The Consultative Committee for Space Data Systems

Draft Recommendation for Space Data System Standards

OPEN ARCHIVAL INFORMATION SYSTEM INTEROPERABILITY FRAMEWORK (OAIS-IF) ARCHITECTURE DESCRIPTION

PROPOSED DRAFT RECOMMENDED STANDARD

CCSDS 000.0-W-0

MAGENTA BOOK
October 2022

AUTHORITY

Issue:	Blue Book, Issue 0
Date:	May 2019
Location:	Not Applicable

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

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

This document is published and maintained by:

CCSDS Secretariat
National Aeronautics and Space Administration
Washington, DC, USA
E-mail: secretariat@mailman.ccsds.org

STATEMENT OF INTENT

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

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

CCSDS Recommendations take two forms: **Recommended Standards** that are prescriptive and are the formal vehicles by which CCSDS Agencies create the standards that specify how elements of their space mission support infrastructure shall operate and interoperate with others; and **Recommended Standards** that are more descriptive in nature and are intended to provide general guidance about how to approach a particular problem associated with space mission support. This **Recommended Standard** is issued by, and represents the consensus of, the CCSDS members. Endorsement of this **Recommended Standard** is entirely voluntary and does not imply a commitment by any Agency or organization to implement its recommendations in a prescriptive sense.

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

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

FOREWORD

This document is a draft technical Recommended Standard for use in developing and maintaining broader consensus on what is required for an archive to provide permanent, or indefinite long term, preservation of digital information.

This draft Recommended Standard establishes a framework of specifications that forms the basis for the Open Archival Information System (OAIS) Interoperability Framework (IF). OAIS is a long-established Process Framework (PF) to enable digital preservation in trustworthy archives. The OAIS-IF supplements OAIS with interoperable technical specifications that will allow interoperability between users and multiple archives, and between multiple archives. The OAIS-IF is not required for an archive to cite compliance with OAIS.

OAIS provides a basis for further standardization within an archival context. OAIS-IF is an example of that further standardization.

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

http://www.ccsds.org/

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

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

Member Agencies

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

Observer Agencies

- Austrian Space Agency (ASA)/Austria.
- Belgian 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.
- Electronics and Telecommunications Research Institute (ETRI)/Korea.
- 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.
- Swiss Space Office (SSO)/Switzerland.
- United States Geological Survey (USGS)/USA.

PREFACE

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

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

DOCUMENT CONTROL

Document	Title and Issue	Date	Status
CCSDS 000.0-W-0	[Document Title], Proposed Draft Recommended Standard, Issue 0	[Month Year]	Current proposed draft

TABLE OF CONTENTS

TABLE OF FIGURES

1 INTRODUCTION

1.1 PURPOSE AND SCOPE

The purpose of this document is to define the CCSDS and International Organization for Standardization (ISO) **Open Archival Information System** (OAIS) Interoperability Framework (IF). An OAIS is an Archive, consisting of an organization, which may be part of a larger organization, of people and systems, that has accepted the responsibility to preserve information and make it available for a **Designated Community**. The OAIS-IF is a supplement to that overarching standard that adds capabilities for system interoperability between users and archives, and between coordinating archives. This document outlines a data system architectural approach and a set of specifications for interfaces required for interoperability and that are visible to Producers and Consumers. This standard is the Architecture Description document that sets the overall architectural framework for the OAIS-IF suite of standards.

The OAIS-IF is an implementable framework for digital repositories that enables international and collaborative research. Its aim is to provide a set of interoperable protocols and interface specifications that will enable the access and re-use of the data, both within and across the operational boundaries of trusted digital repositories. The OAIS-IF is designed to be effectively applied broadly across a spectrum of small, medium, and large use cases and involving a wide variety of stakeholders.

Implementers and system developers that plan to develop systems compliant with the OAIS-IF suite of standards should have a solid grasp of the precepts, concepts and terminology of the Reference Model for an OAIS as described in CCSDS 650.0-M-2.

The information being maintained in these Archives has been deemed to need **Long Term Preservation**, even if the OAIS itself is not permanent. **Long Term** is long enough to be concerned with the impacts of changing technologies, as well as support for new media and data formats, or with a changing Knowledge Base of the Designated Community or changes within the Designated Community or its definition. Long Term may extend indefinitely. Further treatment of the scope of Long Term preservation is in the RM for OAIS, CCSDS 650.0-M-2.-

In terms of scope, the Architecture Description Document is intended to specify normative requirements only for the OAIS-IF components of an OAIS. However to provide context for the OAIS-IF the document describes external components of the archive system and the consumer and producer clients. The interfaces between the OAIS-IF and external components are the key assets specified to achieve interoperability across those interfaces. They are fully specified and normative in this document. However, underlying functions below the interfaces within the client or archive components may be developed differently than this description as long as they support the specified normative functions of the interoperable interfaces.

1.2 APPLICABILITY

Like the OAIS Reference Model in CCSDS 650.0-M-2, this document may be applicable to any Archive that complies with that OAIS standard as well as any archive that wishes to interoperate using the standard. It is specifically applicable to organizations with the responsibility of making information available for the Long Term. This includes organizations with other responsibilities, such as processing and distribution in response to programmatic needs.

This architecture is specifically designed to supplement OAIS Archives. However, this architecture or components of it may be used by archives that are partially or fully non-compliant to the Reference Model for OAIS. The authors of this standard cannot guarantee that these technical approaches will work to fulfill objectives of archives that are not fully OAIS compliant. It is hoped that in these cases partial implementation of the OAIS-IF will encourage greater adoption of the RM for OAIS as archives learn the value of the OAIS practices that enable truly trustworthy Archives for preserving valuable information.

It is intended that the functionality and components in OAIS-IF will exactly mirror the content of the RM for OAIS. However, since these are two separate documents with updates released at different times and different approval cycles, it may be that new functions can be added to OAIS-IF that are not yet in the RM for OAIS. Likewise, there may be new functions in OAIS that are not yet in the OAIS-IF. The intention is to keep the OAIS RM practice and the OAIS-IF specification as closely aligned as possible. However, perfect alignment may not be possible at every given point in time.

These specifications, including the functional and information modeling concepts, are relevant to the comparison and design of facilities which hold information, on a temporary basis, for three reasons:

– When taking into consideration the rapid pace of technology changes or possible changes in a Designated Community, there is the likelihood that facilities, thought to be holding information on a temporary basis, will in fact find that some or much of their information holdings will need Long Term Preservation attention. Stable OAIS-IF standards will help abate the disruption of technology changes.

– Although some facilities holding information may themselves be temporary, some or all of their information may need to be preserved indefinitely. Such facilities need to become active participants in the Long Term Preservation effort and adoption of OAIS-IF will facilitate that transition.

– Regardless of preservation objectives, this architecture enables interoperability for efficiency benefits, preservation benefits, and cross-discipline research benefits.

1.3 OAIS-IF STAKEHOLDERS

In a broad sense, OAIS-IF has applicability to the following stakeholders. This is not an exclusive list, but is intended to illustrate how the document should be of interest to key organization participants.

- **Any organization who has implemented or plans to implement an OAIS-compliant system.** Not all OAIS-compliant systems will have OAIS-IF capabilities. Indeed, as this first version of OAIS-IF is released, none of the OAIS systems in the world will be OAIS-IF compliant. But OAIS

implementers should evaluate the benefits to themselves and their customers from implementing an OAIS-IF compliant interoperable archive. Therefore, they have a stake in OAIS-IF.

- **Managers**, who we assume are key decision makers and determine technology adoption and use. We use the Manager stakeholder broadly for anyone who sets overall OAIS policy.
- **Application Software Developers**, who are those responsible for providing software at an application level (i.e. software implementing any of the six functional entities[1] of an OAIS). Application software is likely to be repository-specific.
- **Infrastructure Software Developers**, who are those responsible for providing the underlying software framework or environment which may be used by application software developers. This software is much less likely to be repository specific. The distinction between application and infrastructure is not necessarily exact but the separation from application software is useful in identifying the parts of OAIS-IF that form part of the underling infrastructure and are more likely to be reused from repository to repository.

1.4 RATIONALE

The rationale for OAIS and the Reference Model for OAIS is captured in CCSDS 650.0-M-2.

The rationale for the OAIS Interoperability Framework includes the rationale for OAIS (not repeated here) because it supports OAIS by augmenting it with capabilities for interoperability.

The rationale and vision for OAIS-IF is that in the long-range future it will provide:

- A common user interface experience for users (providers and consumers) of OAIS Archives when accessing many diverse kinds of archives through the OAIS-IF.

- An efficient standardized way for archives to exchange data between archives using the same standardized OAIS-IF interfaces.

- Given broad acceptance of OAIS-IF in the OAIS community, a better chance that long-term preservation will work because future generations can easily find the interfacing resources (plug-ins, etc.) that can be used to access legacy archives.

- Enhanced capabilities for cross-discipline research when many different disciplines use the same interface, and access to a new archive outside of their Designated Community can be accomplished via OAIS-IF.

1.5 CONFORMANCE

An Archive may conform to the Reference Model for OAIS without conforming to the OAIS-IF.

An OAIS Archive that also conforms to OAIS-IF must implement the normative sections of this document, namely section 3.

While the OAIS Reference Model does not define or require any particular method of implementation, the OAIS-IF must necessarily bound some implementation options in order to insure interoperability. However, the goal of OAIS-IF is to only limit implementation options necessary for interoperability. This is intended to restrict implementation at the interface of systems, but those interfaces are usually characterized to support underlying functionality as required by the OAIS Reference Model. Therefore the definitions at the interfaces and protocols may necessarily imply some underlying functionality as part of the OAIS-IF suite of standards. As described in section 1.1, Purpose and Scope, the description of that functionality outside the OAIS-IF is not normative, and may be implemented in different ways, as long as it supports the specified normative functionality for that interface.

A conformant OAIS-IF Archive may provide additional services that are beyond those required of the OAIS-IF standards.

This document does not assume or endorse any specific computing platform, system environment, system design paradigm, system development methodology, database management system, database design paradigm, data definition language, technology, or media required for implementation.

The OAIS-IF is designed as an interoperability framework to support the development of interoperability between archives, both OAIS Archives and non-OAIS archives, using the OAIS-IF standard. As such, it attempts to address all the major activities of an ***interoperable*** information-preserving Archive in order to define a consistent and useful set of interoperability terms and concepts. A standard or other document that claims to be conformant to the OAIS-IF shall use the terms and concepts defined in the OAIS-IF in the same manner.

1.6 DOCUMENT STRUCTURE

1.6.1 ORGANIZATION BY SECTION

A general description of this document's sections are:

- Section 1 *Purpose and Scope* describes the problem space and rationale for OAIS-IF, and advice on what to expect from the document organization and conventions.
- Section 2 *Overview* provides an informative (non-normative) explanation of the relationships between OAIS-IF components and between them and the environment..
- Section 3 *Interoperability Framework* is a normative description of the requirements on the components of an OAIS-IF architecture. It presents the technical concepts that OAIS-IF uses in order to perform the functions of an OAIS in an interoperable way.
- (Add explanation of annexes once they are solidified)

This Blue Book begins with a description of the context for the creation of OAIS-IF in the form of the motivation and rationale for the framework. Further sections in this Blue Book then offer greater levels of detail about OAIS-IF generated directly from a formal model of the OAIS-IF. This detailed information is presented using the object-oriented paradigm. Each class, attribute, and relationship is formally defined using text and UML diagrams. It is anticipated that these sections will be primarily applicable to system developers but will be of interest to other stakeholders.

It is expected that after this document is approved and published by CCSDS, the model will be made available online by CCSDS. This should be a valuable aid to system developers of OAIS-IF systems.

1.6.2 TYPOGRAPHICAL CONVENTIONS

There are many terms which are used in this framework and which need to have well-defined meanings. These terms are defined in subsection 1.6.2. When first used in the text, they are shown in bold and are capitalized. Subsequent use employs capitalization only. Because of their extensive use in this document, the defined terms 'data' and 'information' will not always be capitalized unless they are part of another defined term. The defined term 'archive' will not be capitalized unless it is used as the equivalent of an 'OAIS Archive'.

Many diagrams are included throughout this reference model, primarily in Sections 4 and 6. In text discussing the diagrams, block names are capitalized and flows are italicized.

1.7 DEFINITIONS

1.7.1 ACRONYMS AND ABBREVIATIONS

AIC Archival Information Collection

AIP Archival Information Package

AIU Archival Information Unit

API Application Programming Interface

ASCII American Standard Code for Information Interchange

CCSDS Consultative Committee for Space Data Systems

CD-ROM Compact Disk - Read Only Memory

CDO Content Data Object

CRC Cyclic Redundancy Check

CSV Comma Separated Value

DBMS Data Base Management System

DIP Dissemination Information Package

DRM Digital Rights Management

FITS	Flexible Image Transport System
FTP	File Transfer Protocol
HFMS	Hierarchical File Management System
IEEE	Institute of Electrical and Electronics Engineers
IETF	Internet Engineering Task Force
ISBN	International Standard Book Number
ISO	International Organization for Standardization
MPEG	Moving Picture Experts Group
OAIS	Open Archival Information System
PDF	Portable Document Format
PDI	Preservation Description Information
QA	Quality Assurance
RFC	Request For Comment
SIP	Submission Information Package
UML	Unified Modeling Language
VHS	Video Home System
WWW	World Wide Web
XFDU	XML Formatted Data unit
XML	eXtensible Markup Language

1.7.2 TERMINOLOGY

There are many terms which are used in this standard and which need to have well-defined meanings. These terms are defined in this subsection. When first used in the text, they are shown in bold and are capitalized. Subsequent use employs capitalization only.

This standard is applicable to all disciplines and organizations that do, or expect to, preserve and provide information in digital form, these terms cannot match all of those familiar to any particular discipline (e.g., traditional archives, digital libraries, science data centers). Rather, the approach taken is to use terms that are not already overloaded with meaning so as to reduce conveying unintended meanings. Therefore, it is expected that all disciplines and organizations will find that they need to map some of their more familiar terms to those of the OAIS Reference Model and OAIS-IF. This should not be difficult and is viewed as a contribution, rather than a deterrent, to the success of these

standards. For example, archival science focuses on preservation of the 'record'. This term is not used in these standards, but one mapping might approximately equate it with 'Content Data Object within an Archival Information Package'.

TERMS TO BE SUPPLIED (Probably after current OAIS Red Book is published)

1.8 REFERENCES

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

[Only references required as part of the specification are listed in the References subsection. See CCSDS A20.0-Y-4, *CCSDS Publications Manual* (Yellow Book, Issue 4, April 2014) for additional information on this subsection.]

Reference Model for an Open Archival Information System (OAIS). Magenta Book. CCSDS 650.0-M-2 Issue 2. June 2012. (to be changed to Issue 3 when issue 3 is released)

Audit and Certification of Trustworthy Digital Repositories. Magenta Book. Recommended Practice CCSDS 652.0-M-1. September 2011. (to be changed when issue 3 is released)

OTHER REFERENCES TO BE SUPPLIED

2 OVERVIEW

The following concepts set the context for normative definitions starting in section 3.

2.1 OAIS INTEROPERABILITY FRAMEWORK (OAIS-IF)

An OAIS Archive is an organization that intends to preserve information for access and use by a Designated Community.

An Open Archival Information System (OAIS) is an Archive, an organization that intends to preserve information for access and use by a Designated Community. It meets a set of responsibilities that allows an OAIS Archive to be distinguished from other uses of the term 'Archive'.

The OAIS Interoperability Framework (OAIS-IF) is a framework based on the concepts presented in the OAIS Reference Model (RM) and augmented with features designed during several decades of digital archive development. The OAIS-IF is designed to be implementable and is an interoperable framework that fosters the acquisition, stewardship, and continuing access to data products, related information resources, and services for a Designated Community.

The environment surrounding an OAIS includes Management, Consumers, and Producers. The resulting environment of the OAIS-IF is illustrated in figure 1.

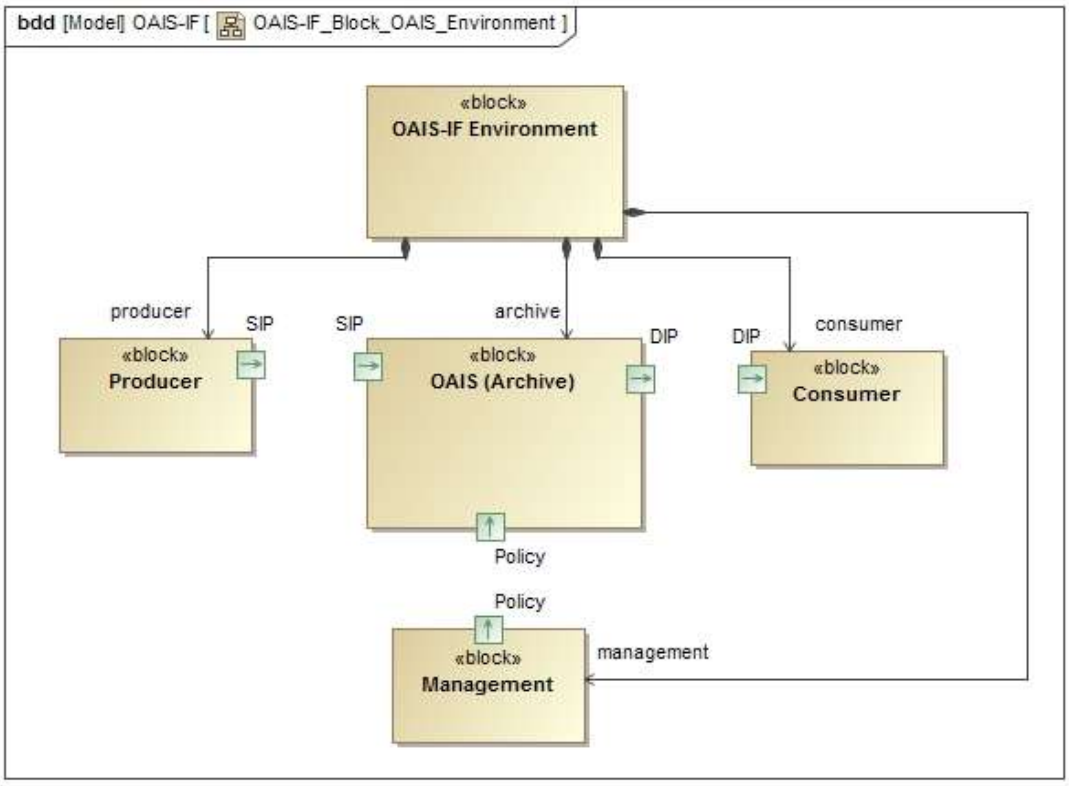

Figure 1 - OAIS Environment

Management is the role played by those who set overall OAIS policy as one component in a broader policy domain, for example as part of a larger organization.

Producer is the role played by those persons or client systems that provide the information to be preserved. This can include other OAISes or internal OAIS persons or systems. A Producer creates a Submission Information Package(s) (SIPs) and submits it to the Archive where it is processed into one or more Archival Information Packages (AIPs).

A Consumer is the role played by those persons, or client systems, who interact with OAIS services to find preserved information of interest and to access that information. A Consumer receives a Dissemination Information Package(s) (DIP) from the Archive.

2.2 OAIS FUNCTIONAL ENTITIES

Within an OAIS (Archive), an OAIS Functional Entity is an entity responsible for an operational function in a specific part of an Open Archive Information System (OAIS). The OAIS functional entities include Access, Administration, Archival Storage, Data Management, Ingest, and Preservation Planning. The OAIS Interoperability Framework, being based on the OAIS Reference Model, has an additional functional entity the Adapter Layer.

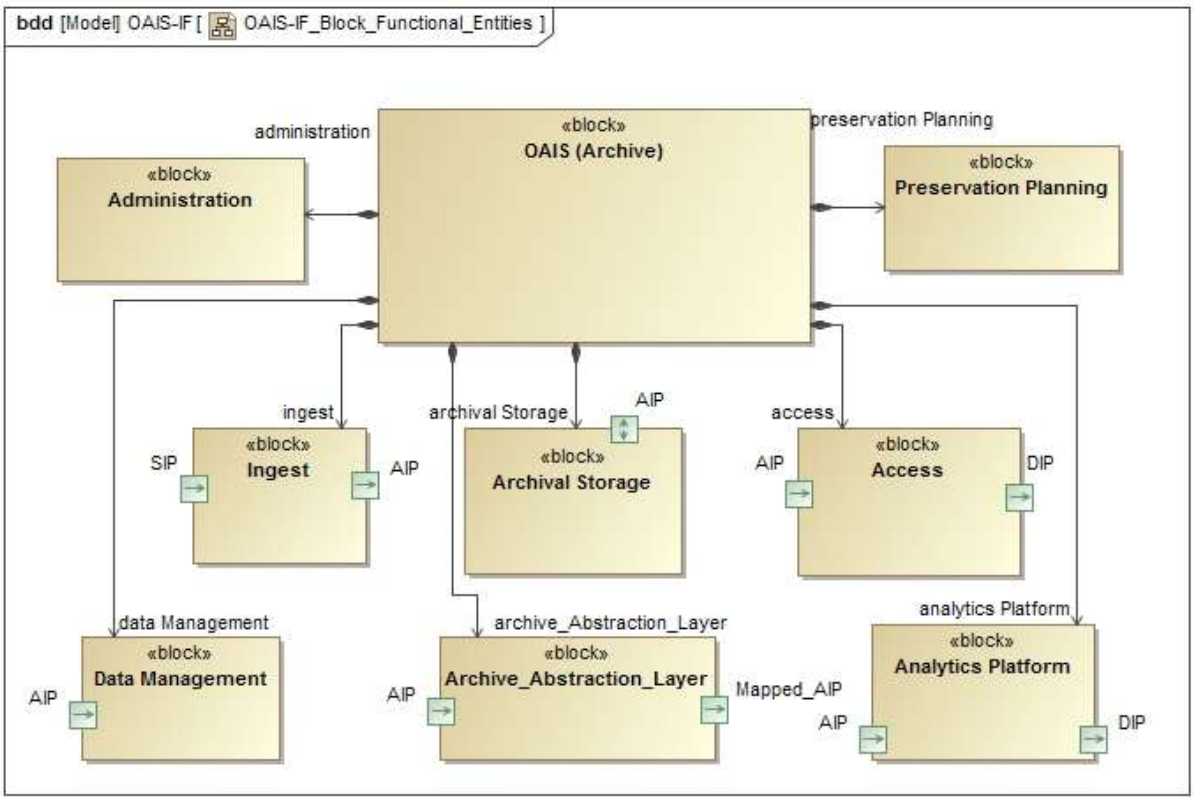

Figure 2 - OAIS Functional Entities

The Access Functional Entity (aka Access) contains the services and functions which make the archival information holdings and related services visible to Consumers. Access generates and provides a Dissemination Information Packages (DIP) to a Consumer, produces a Query Response for a Consumer, and provides Report Assistance to a Consumer.

The Administration Functional Entity (aka Administration) contains the services and functions needed to control the operation of the other OAIS functional entities on a day-to-day basis. For Consumers and Producers Administration sends Information Requests, Bills and Special Request Responses to Consumers. Final Ingest Report and possible liens are sent to a Producer.

The Archival Storage Functional Entity (aka Archival Storage) contains the services and functions used for the storage and retrieval of Archival Information Packages (AIPs).

The Data Management Functional Entity (aka Data Management) contains the services and functions for populating, maintaining, and accessing a wide variety of information. Some examples of this information are catalogs and inventories on what may be retrieved from Archival Storage, processing algorithms that may be run on retrieved data, Consumer access statistics, Consumer billing, Event Based Orders, security controls, and OAIS schedules, policies, and procedures.

The Ingest Functional Entity (aka Ingest) contains the services and functions that accept Submission Information Packages (SIPs) from Producers, prepares Archival Information Packages for storage, and ensures that Archival Information Packages and their supporting Descriptive Information become established within the OAIS. Ingest sends Receipt Confirmation to a Producer.

The Preservation Planning Functional Entity (aka Preservation Planning) provides the services and functions for monitoring the environment of the OAIS and provides recommendations and preservation plans to ensure that the information stored in the OAIS remains accessible to, and understandable by, and sufficiently usable by, the Designated Community over the Long Term, even if the original computing environment becomes obsolete. Preservation Planning surveys a Consumer and surveys a Producer.

The Adapter Layer Functional Entity provides a mapping and possible translation between an object class in the OAIS Information Model and an object class in a non-conforming information model. package. For example a Consumer asking for Provenance Information as defined in in the OAIS Information Model could receive information about a derived product, the source products, and processing software that was grouped and classified as processing history in a non-OAIS information package. This is of course if the Adapter Layer had definitions of the two information models, how their components were related, and how to translate from one to the other if needed.

The Analytical Platform is a unified data analysis solution designed to address the demands of users beyond the data management infrastructure necessary for using a long-term trusted digital repository. In general it provides contextual analyzed data from across the repository and joins different tools for creating analytics systems.

2.3 OAIS INTEROPERABILITY FRAMEWORK DEFINITION

The OAIS Interoperability Framework (OAIS-IF) supports the Producer and Consumer as they perform their roles and interact with the Ingest and Access functional entities of an OAIS. Three viewpoints of the framework are presented, an abstract component architecture, an information model, and an abstract functional interface. The abstract component architecture consists of a hierarchy of three major components: Client, OAIS Interoperability Framework, and OAIS IF Archive. The OAIS Interoperability Framework in turn consists of the Consumer and Producer Interfaces, the OAIS Information Model, and the Adapter layer.
The OAIS Information Model provides the framework's domain of discourse. The entities defined in the domain of discourse are the objects passed between functional entities of the framework. For example, the OAIS Digital Object is passed between a Consumer and an archive.

The abstract functional interface is defined within the Abstraction Layer and consists of formally interfaces and their operations. These interfaces must be implemented by the functional entities. For example, an Adapter for an archive must implement the Adapter Interface, Message Interface, and the Identifier Interface. Due to inheritance, the Adapter also implements the Information Object Interface.

The Negotiate Interface is used by a Client and the archive to identify one or more implemented Adapters to be used by the Consumer Interface and Producer Interface and the archive to interoperate. Once an Adapter has been set, the Consumer and Producer interoperate via Messages

using any implemented protocol. Any entity defined in the Information Model may be passed between any two connected functional entities that implement the Information Object Interface.

.

3 INTEROPERABILITY FRAMEWORK

In this section the environment in which the OAIS Interoperability Framework (OAIS-IF) exists is first diagramed to provide the context for the normative sections of the document, the OAIS-IF Interfaces and the OAIS Information Model. [CN]. The primary components, Client, OAIS Interoperability Framework, and OAIS_IF_Archive, are decomposed into their constituent sub-components down to the level of individual class and interface definitions. The entities comprising the OAIS Information Model and the Access and Ingest functional entities come directly from the OAIS Reference Model. A new component, Negotiate, has been added to address the selection process for finding adapters to interface with specific clients and archives.

The components represent modular part of the system that encapsulates the state and behavior of a set of elements such as attributes and methods. Their behavior is defined in terms of required interfaces.

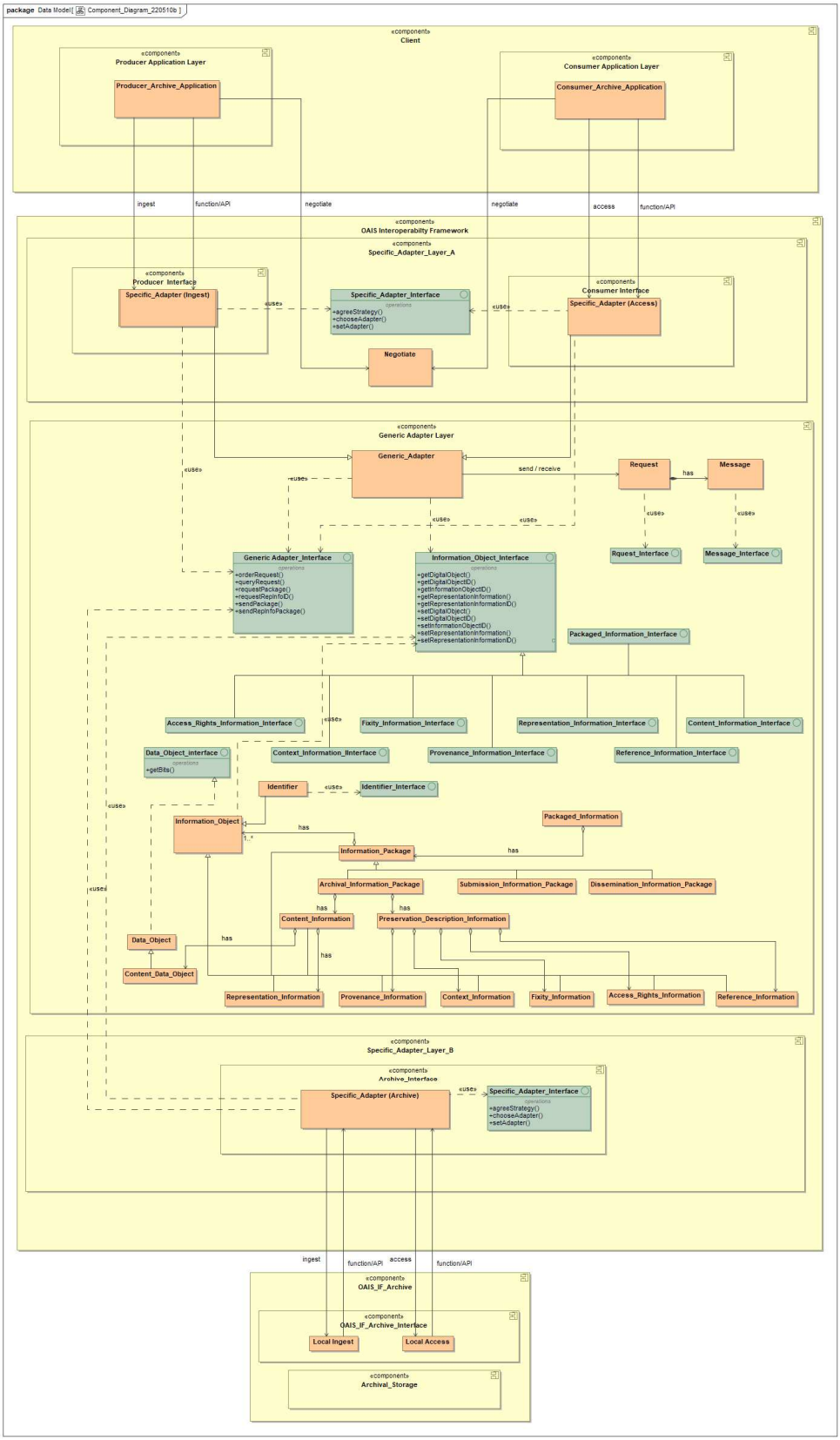

Figure 3 - Component Diagram

3.1 CLIENT

The Client component represents a computer system or process that requests a service of another computer system or process (a server) using a specific protocol and accepts the server's responses. The Client class is a Component. This section is informative.

3.1.1 CONSUMER_APPLICATION_LAYER

The Consumer Application Layer contains a program or group of programs designed for a Consumer. The Consumer Application Layer component is a Component and is an element of Client. This section is informative.

3.1.1.1 Consumer_Archive_Application

The Consumer Archive Application object class represents an application for use by a user performing the role of a consumer in a data archive. The Consumer Archive Application negotiates for Specific Adapters via the Negotiate service and makes requests for Information Packages or Dissemination Information Packages through the Access service. As an application it may invoke OAIS Interoperability Framework services either through functions calls by consuming the framework or through an Application Programming Interface (API) developed as a wrapper for the framework. The Consumer Archive Application is an element of the Consumer Application Layer.

3.1.2 PRODUCER_APPLICATION_LAYER

The Producer Application Layer contains a program or group of programs designed for Producers. The Producer Application Layer component is a subclass of Component and is an element of Client. This section is informative.

3.1.2.1 Producer_Archive_Application

The Producer Archive Application object class represents an application for use by a user performing the role of a producer in a data archive. The Producer Archive Application negotiates for Specific Adapters via the Negotiate service and ingests Submission Information Packages through the Ingest service. As an application it may invoke OAIS Interoperability Framework services either through functions calls by consuming the framework or through an Application Programming Interface (API) developed as a wrapper for the framework. The Producer Archive Application is an element of the Producer Application Layer.

3.2 OAIS_INTEROPERABILITY_FRAMEWORK

The OAIS Interoperability Framework component represents a layered distributed application structure that partitions tasks between the providers and the requesters of services and resources. As seen in Figure 4, a resource of type Information Object is passed between two application stacks over a communication channel. The roles of provider and requester can be held alternately by either stack.

The framework has two types of adapters, the generic and specific adapter. Generic Adapters are components of the Generic Adapter Layer and provide interfaces for communicating between application stacks. The Generic Adapter implements standard interfaces for requesting and sending an Information Object. The Generic Adapter interface is normative.

Specific Adapters are components of the Specific Adapter Layers and provide interfaces for client and archive software. Specific Adapters extend the Generic Adapter and implement the standard generic adapter interfaces. Specific Adapters may also implement custom-made interfaces for clients and archives that are not OAIS compliant. The choice of Specific Adapters to use by the client or archive components are negotiated external to the Generic Adapter layer. The Specific Adapter Interface is informative.

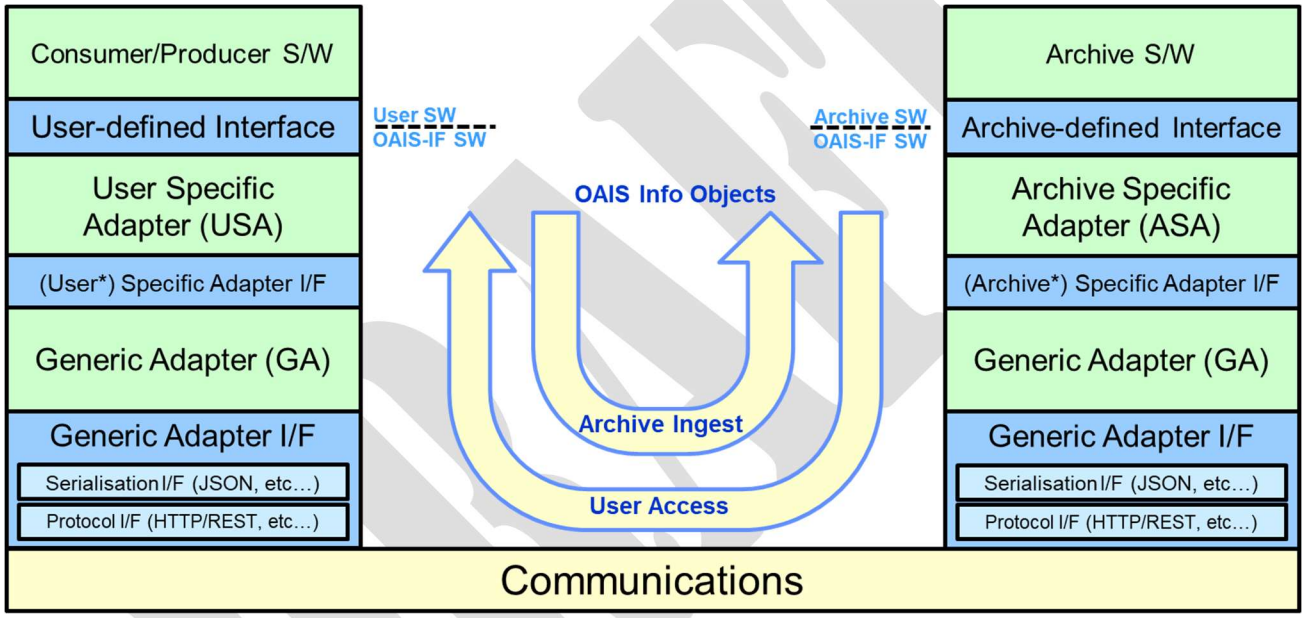

Figure 4 – Framework Adapter Stack

Notes on Figure 4:

- *Specific Adapters may be the same on both User and Archive side, or they may be unique on each side. The DAI WG goal is that they can be the same, but that is not yet settled. (hence the parenthetical, for now.)

- Blue boxes are planned as CCSDS Blue Books (normative), green boxes described in the OAIS-IF Green Book (informative).

- OAIS-IF defines only interfaces as Blue Books. The roles of the adapters are discussed in the (informative) Green Book. Within the adapters, however, functionality may be required for the functions necessary to meet the specified interface. OAIS-IF will initially cite JSON and

HTTP/REST for serialization and protocol functions, but other options may be standardized in later versions or additional documents.

In Figure 4, a communication network is used to pass OAIS Information Objects between two adapter stacks, representing the Access and Ingest data flows. The Protocol interface enables the two stacks to communicate with each other over networked devices. The serialization interface serializes (i.e., translates) the Information Object data structure into a format that can be transmitted over the network and then de-serializes (i.e., reconstructs) the Information Object after the transmission is complete.

The Hypertext Transfer Protocol (HTTP) communication protocol and the JavaScript Object Notation (JSON) data format are both broadly accepted for transporting data over communication networks and are included as default implementations. Additional options for these implementations may be adopted by implementers, or may be added as optional specifications in future iterations of this document (or new documents in the OAIS-IF Document Tree). However, users/archives must settle on the same implementation in order to guarantee interoperability.

The generic adapter provides a set of interfaces for sending and requesting messages between the users and archives. The payload of the message is an Information Object and the message itself is an Information Object. The Information Object interfaces provide methods for manipulating the components of the Information Object and are inherited by the subclasses of Information Object, for example, Information Package, Preservation Description Information, Representation Information, and Content Information.

3.2.1 GENERIC ADAPTER LAYER

The Generic Adapter Layer contains the Generic Adapter and other core framework classes that are designed for requesting and sending an Information Object between software components. The Generic Adapter Layer is a component of the OAIS Interoperability Framework. This section is normative.

3.2.1.1 Adapter

An Adapter (Binding) is a wrapper library that bridges two programming languages so that a library written for one language can be used in another language. An OAIS-IF adapter is a bridge between client and archive software components.

The Adapter provides a set of interfaces for passing Information Objects between the software components. It must implement the Message Interface and provide a standard protocol or interaction pattern for passing Information Objects between the components. It also implements identifier,

protocol, and serialization interfaces that are needed to identify and transfer Information Objects over a communication network.

A software component can implement an adapter by either consuming the adapter or wrapping the adapter in an Application Programming Interface (API). The Adapter is an element of the several Adapter Layers within the OAIS Interoperability Framework component.

3.2.1.2 Generic Adapter

The Generic Adapter is an Adapter that implements the Generic Adapter interfaces. The Generic Adapter is a class designed for requesting and sending the Information Object class and its subclasses. This section is normative.

3.2.1.2.1 Adapter_Interface

All Known Implementing Classes:

AbstractClient

`public interface AdapterInterface`

The Adapter Interface is a well-defined entry point for sending and getting Information Packages.

This interface provides the primary inter-operability interface.

3.2.1.2.1.1 Method Summary

All Methods	
Modifier and Type	**Method and Description**
`void`	`asyncGetPackage(Identifier identifier, Message message)` The asyncGetPackage method enqueues a Request to the specified receiver for one or more Information Objects.
`void`	`asyncSendPackage(Identifier identifier, Message message)` The asyncSendPackage method sends an InfoObject to the specified receiver.
`Identifier []`	`getEndPoints(String source)`

	The getEndPoints method returns a set of End Point Connection as a list of Identifiers.
void	syncSendPackage(Identifier identifier, Message message) The syncSendPackage method sends an InfoObject to the specified receiver.

3.2.1.2.1.2 Method Detail

3.2.1.2.1.2.1 asyncGetPackage

```
void asyncGetPackage(Identifier identifier, Message message)
```

The asyncGetPackage method enqueues a Request to the specified receiver for one or more Information Objects. The method sends the query as an InfoObject by including it as the RepInfo of a Message and using the Request Asynchronous interaction pattern. The results are expected as the Data Object of the Message.

Parameters:

`identifier` - the identifier of the receiver

`message` - the request as the Data Object of a Message

3.2.1.2.1.2.2 asyncSendPackage

```
void asyncSendPackage(Identifier identifier, Message message)
```

The asyncSendPackage method sends an InfoObject to the specified receiver. The method sends the InfoObject as the Data Object of a Message using the Request Asynchronous interaction pattern. The results are expected in the RepInfo of the Message.

Parameters:

`identifier` - the identifier of the receiver

`message` - a message containing the Info Object being sent.

3.2.1.2.1.2.3 syncSendPackage

```
void syncSendPackage(Identifier identifier, Message message)
```

The syncSendPackage method sends an InfoObject to the specified receiver. The method sends the InfoObject as the Data Object of a Message using the Request Synchronous interaction pattern. The results are expected in the RepInfo of the Message.

Parameters:

`identifier` - the identifier of the receiver

`message` - a message containing the Info Object being sent.

3.2.1.2.1.2.4 getEndPoints

```
Identifier[] getEndPoints(String source)
```

The getEndPoints method returns a set of End Point Connection as a list of Identifiers. sends an InfoObject to the specified receiver. The method sends the InfoObject as the Data Object of a Message using the Request Synchronous interaction pattern. The results are expected in the RepInfo of the Message.

Parameters:

`source` - the identifier of the source

Returns:

Identifier [] a list of End Point Connections.

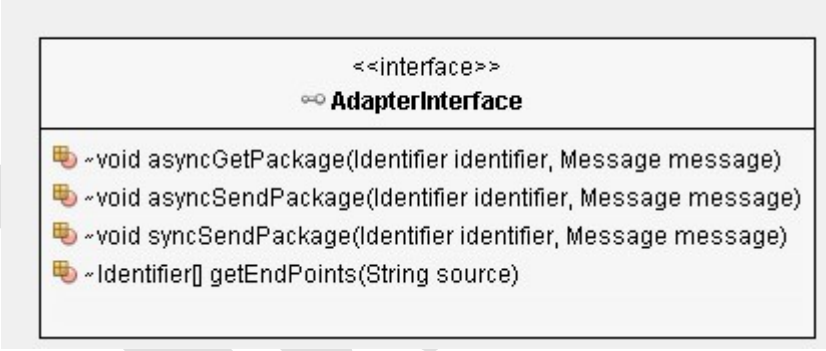

Figure 5 - Adapter Interface

The Adapter interface passes Information Objects between software components through the implemented Message Interface. The Interaction Diagram in Figure 6 illustrates seven interaction behaviors involving the flow of messages between the two adapters. Interactions 1 and 2 set up the communication channel between the two adapters.

In Interactions 3, a Message, a subclass of Information Object, is passed to the second adapter and contains a request. Before being sent, the Message is serialized into a Digital Object. In Interaction 5, the Message is returned to the sender and contains the query result as an Information Object.

Interaction 3 and 5 together implement the "Request" interaction protocol, a two phased interaction where the sent Message is expected to be returned to the sender.

Interaction 4 allows an adapter to navigate the Representation Network to identify and return a specific instance of Representation Information. Interaction 6 and 7 allow for changing the specific adapters and communication protocols.

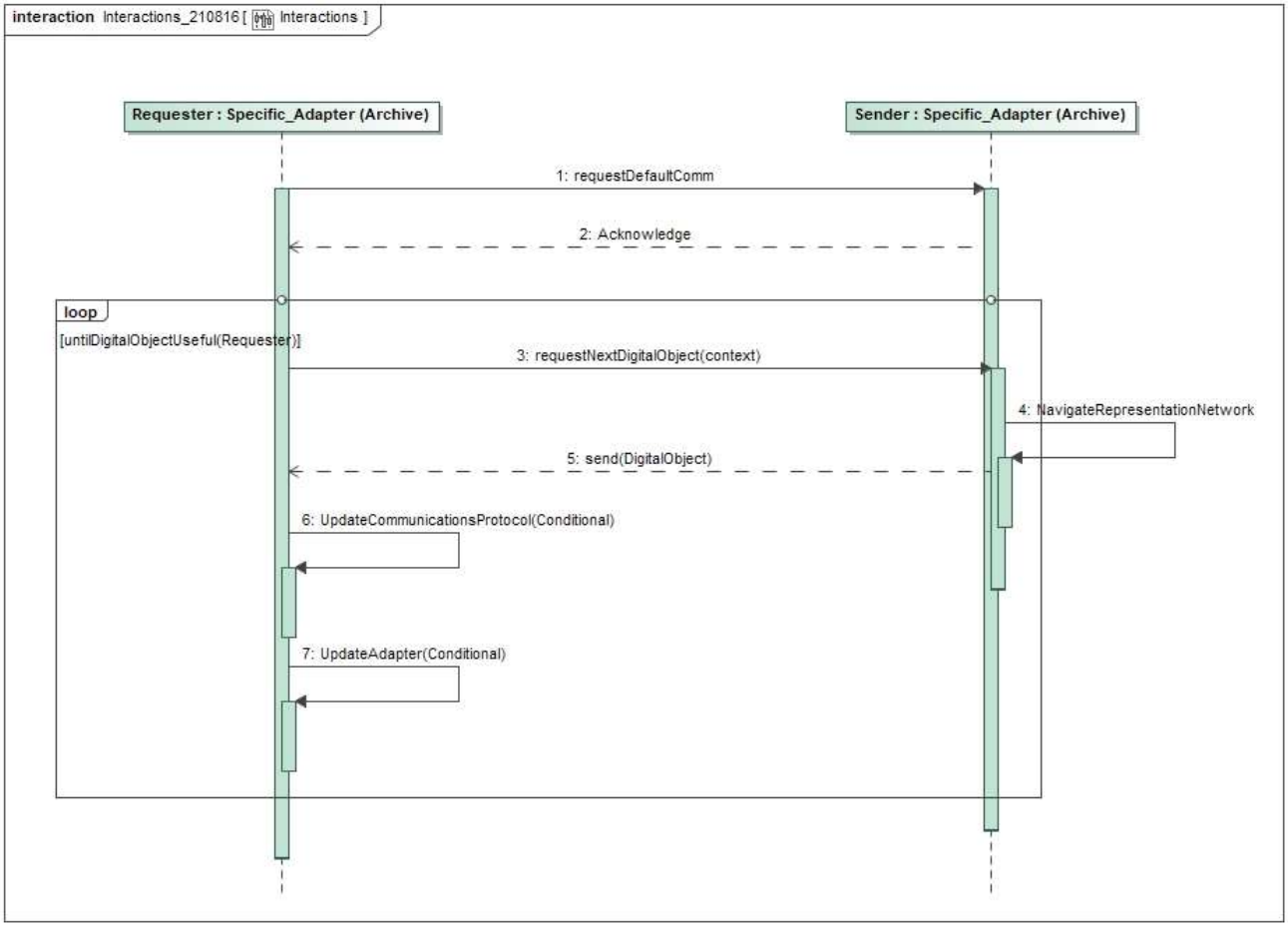

Figure 6 - Interaction Diagram

3.2.1.2.2 Abstract Generic Adapter

An Abstract Generic Adapter is shown in Figure 5. The Adapter object class is an abstract class from which more specialized adapters can be extended. The Adapter object class implements the Message Interface to provide a standard protocol or interaction pattern for passing Information Objects between software components. It also implements identifier, protocol, and serialization interfaces that are

needed to identify and transfer Information Objects over a communication network. The methods required for manipulating the components of an Information Object and its extensions, are defined by the Information Object's interfaces. The Request Interaction Protocol is a necessary and sufficient protocol for message passing. This Abstract client can be used by Consumers, Producers, and Archives.

It is an element of the Adapter Layer components. This section is normative.

Figure 7 - Abstract (Generic) Adapter Implementing the Adapter Interface

3.2.1.3 Information_Object

Information Object: A Data Object together with its Representation Information. [C1]

The Information Object class is composed of a Data Object and a Representation Information class. It implements the Information Object Interface is managed by an Archival Storage functional entity, and is an element of the OAIS Interoperability Framework component. This section is normative.

3.2.1.3.1 Information_Object_Interface

The Information Object Interface is a well-defined entry point for accessing an Information Object. The interface is an element of the Abstraction Layer component. This section is normative.

- All Known Implementing Classes: AccessRightsInformation, ContentInformation, ContextInformation, FixityInformation, InfoObject, PackagedInformation, ProvenanceInformation, ReferenceInformation, RepresentationInformation

```
public interface InfoObjectInterface
```

The InfoObject (Information Object) Interface is a well-defined entry point and contract for getting and putting Information Objects and its components.

3.2.1.3.1.1 Method Summary

All Methods	
Modifier and Type	**Method and Description**
Object	getDataObject() The getDataObject method returns the Data Object component of this Information Object.
Identifier	getDataObjectID() The getDataObjectID method returns the identifier of the Data Object component of this Information Object.
Identifier	getInfoObjectID() The getInfoObjectID method returns the identifier of this Information Object.
InfoObject	getRepInfo() The getRepInfo method returns the Representation Information component of this Information Object.
Object	getRepInfoDataObject() The getRepInfoDataObject method returns a Data Object component of this Information Object.

Identifier	getRepInfoDataObjectID() The getRepInfoDataObjectID method returns the Identifier of a Data Object component of this Information Object.
void	setDataObject(Identifier identifier, DataObject dataObject) The setDataObject method sets the Data Object component of this Information Object.
void	setInfoObjectID(Identifier identifier) The setInfoObjectID method sets the identifier of this Information Object.
void	setRepInfoDataObject(Identifier identifier, RepInfoDataObject repInfoDataObject) The setRepInfoDataObject method sets a Data Object component of this Information Object.

3.2.1.3.1.2 Method Detail

3.2.1.3.1.2.1 getInfoObjectID

```
getInfoObjectID()
```

The getInfoObjectID method returns the identifier of this Information Object.

Returns:
Identifier - the identifier of this InfoObject

3.2.1.3.1.2.2 getDataObjectID

```
getDataObjectID()
```

The getDataObjectID method returns the identifier of the Data Object component of this Information Object.

Returns:
Identifier - the identifier of the DataObject

3.2.1.3.1.2.3 getRepInfoDataObjectID

```
getRepInfoDataObjectID()
```

The getRepInfoDataObjectID method returns the Identifier of a Data Object component of this Information Object. More specifically this method returns the Identifier of the Data Object of the Representation Information for this Information Object.

Returns:
Identifier - the identifier of the RepInfoDataObject

3.2.1.3.1.2.4 getDataObject

```
getDataObject()
```

The getDataObject method returns the Data Object component of this Information Object.

Returns:
Object - the DataObject for this InfoObject

3.2.1.3.1.2.5 getRepInfo

```
InfoObject getRepInfo()
```

The getRepInfo method returns the Representation Information component of this Information Object.

Returns: InfoObject - the RepInfo for this InfoObject

3.2.1.3.1.2.6 getRepInfoDataObject

```
getRepInfoDataObject()
```

The getRepInfoDataObject method returns a Data Object component of this Information Object. More specifically this method returns the Data Object of the Representation Information for this Information Object.

Returns:
Object - the DataObject for this InfoObject's RepInfo

3.2.1.3.1.2.7 setInfoObjectID

```
void setInfoObjectID(Identifier identifier)
```

The setInfoObjectID method sets the identifier of this Information Object.

Parameters:
`identifier` - the identifier for this InfoObject

3.2.1.3.1.2.8 setDataObject

```
void setDataObject(Identifier identifier, DataObject dataObject)
```

The setDataObject method sets the Data Object component of this Information Object.

Parameters:
`identifier` - the identifier for this DataObject
`dataObject` - the DataObject for this InfoObject

3.2.1.3.1.2.9 setRepInfoDataObject

```
void setRepInfoDataObject(Identifier identifier,
                          RepInfoDataObject repInfoDataObject)
```

The setRepInfoDataObject method sets a Data Object component of this Information Object. More specifically this method sets the Data Object of the Representation Information for this Information Object.

Parameters:
`identifier` - the identifier for this dataObject
`repInfoDataObject` - the RepInfoDataObject for this InfoObject's RepInfo

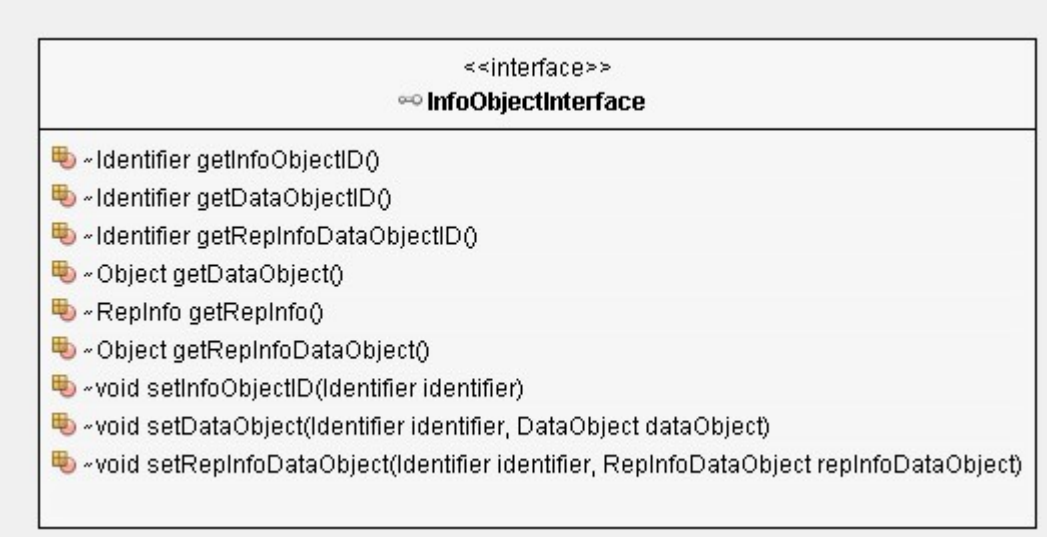

Figure 8 - Information Object Interface

3.2.1.4 Data_Object

Information Object: A Data Object together with its Representation Information. [C1]

The Information Object class is composed of a Data Object and a Representation Information class. It implements the Information Object Interface is managed by an Archival Storage functional entity, and is an element of the OAIS Interoperability Framework component. This section is normative.

3.2.1.4.1 Data_Object_Interface.

The Data Object Interface is a well-defined entry point for accessing a Data Object. The interface requires a getObject method and is an element of the Abstraction Layer component. This section is normative.

- All Known Implementing Classes: DataObject

```
public interface DataObjectInterface
```

The Data Object Interface is a well-defined entry point for getting and putting the Identifier and Object of a Data Object.

3.2.1.4.1.1 Method Summary

All Methods	
Modifier and Type	**Method and Description**
`Identifier`	`getIdentifier()` The getIdentifier method returns the Identifier of this DataObject.
`Object`	`getObject()` The getObject method returns the Object for this Data Object
`void`	`setObject(Identifier identifier, Object object)` The setObject method sets the object for this Data Object.

3.2.1.4.1.2 Method Detail

3.2.1.4.1.2.1 getIdentifier

```
Identifier getIdentifier()
```

The getIdentifier method returns the Identifier of this DataObject.

Returns:
Identifier - the identifier for this DataObject

3.2.1.4.1.2.2 getObject

`getObject()`

The getObject method returns the Object for this Data Object

Returns:
`Object` - the object for this dataObject

3.2.1.4.1.2.3 setObject

```
void setObject(Identifier identifier, Object  object)
```

The setObject method sets the object for this Data Object.

Parameters:
`identifier` - the identifier for this dataObject
`object` - the object for this dataObject

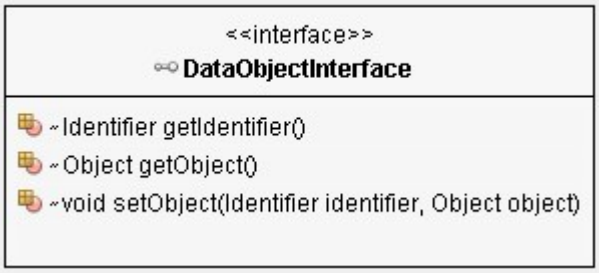

Figure 9 - Digital Object Interface

3.2.1.5 Representation_Information_Data_Object

The Representation Information Data Object class implements the Representation Information Data Object Interface and returns Representation Information as a Data Object. In other words the Data Object returned is the representation information component of an Information Object, not Representation_Information as a subclass of Information Object. It is managed by an Archival Storage functional entity and is an element of the OAIS Interoperability Framework component. This section is normative.

3.2.1.5.1 Representation Information Data Object Interface

The Representation Information Data Object Interface is a well-defined entry point for accessing the Data Object of an Information Object's Representation. The interface is an element of the Abstraction Layer component. This section is normative.

- All Known Implementing Classes: RepInfoDataObject

```
public interface RepInfoDataObjectInterface
```

The Representation Information Data Object Interface is a well-defined entry point for getting and putting the Identifier and Object of a Data Object, the Data Object of an Information Object's Representation Information.

3.2.1.5.1.1 Method Summary

All Methods	
Modifier and Type	**Method and Description**
`Identifier`	`getIdentifier()` The getIdentifier method returns the Identifier of this RepInfoDataObject.
`Object`	`getObject()` The getObject method returns the Object for this RepInfoDataObject
`void`	`setObject(Identifier identifier, Object object)` The setObject method sets the Object for this RepInfoDataObject.

3.2.1.5.1.2 Method Detail

3.2.1.5.1.2.1 getIdentifier

```
Identifier getIdentifier()
```

The getIdentifier method returns the Identifier of this RepInfoDataObject.

Returns:
Identifier - the identifier for this repInfoDataObject

3.2.1.5.1.2.2 getObject

```
getObject()
```

The getObject method returns the Object for this RepInfoDataObject

Returns:
Object - the object for this repInfoDataObject

3.2.1.5.1.2.3 setObject

```
void setObject(Identifier identifier, Object object)
```

The setObject method sets the Object for this RepInfoDataObject.

Parameters:
`identifier` - the identifier for this repInfoDataObject
`object` - the object for this repInfoDataObject

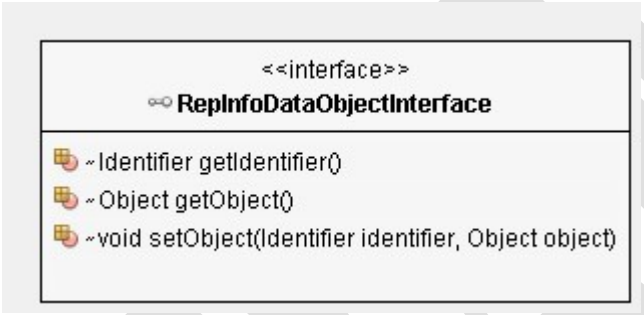

Figure 10 - Representation Information Data Object Interface

3.2.1.6 Identifier

The Identifier class provides a unique name to an entity to identify it as distinct from other entities.

3.2.1.6.1 Identifier_Interface

The Identifier Interface is a well-defined entry point for accessing the identifier. The interface requires a setAdapter method and is an element of the Abstraction Layer component. This section is normative.

All Known Implementing Classes:
EndPointConnection

```
public interface IdentifierInterface
```

The Identifier Interface is a well-defined entry point and contract for an Identifier. An Identifier names an Object.

3.2.1.6.1.1 Method Summary

All Methods	
Modifier and Type	**Method and Description**
`Object`	`getIdentifier()` The getIdentifier method returns the identifier.
`void`	`setIdentifier(java.lang.Object id)` The setIdentifier method stores an Identifier.

3.2.1.6.1.2 Method Detail

3.2.1.6.1.2.1 getIdentifier

```
Object getIdentifier()
```

The getIdentifier method returns the identifier.

Returns:

Object the identifier of an Object.

3.2.1.6.1.2.2 setIdentifier

```
void setIdentifier(java.lang.Object id)
```

The setIdentifier method stores an Identifier.

Parameters:

`id` - the identifier of an Object.

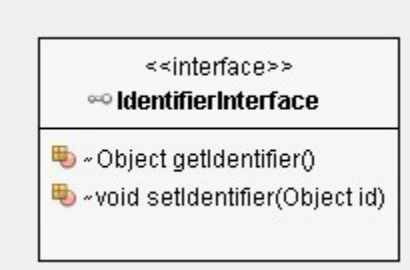

Figure 11 - Identifier Interface

3.2.1.7 End Point Connection

The End Point Connection class is an extension of the Identifier class. It provide a name of a connection to a physical device that connects to and exchanges information over a computer network.

```
Public class EndPointConnection extends Identifier

    implements IdentifierInterface
```

3.2.1.8 Message

The Message class a message is an object, which is sent by a sender, to a receiver.

3.2.1.8.1 Message_Interface

The Message Interface is a well-defined entry point for accessing the components of a Message. The interface is an element of the Adapter Layer component. This section is normative.

- All Superinterfaces:

  InfoObjectInterface

  All Known Implementing Classes: Message

  ```
  public interface MessageInterface

  extends InfoObjectInterface
  ```

  The Message Interface is a well-defined entry point for getting and putting the components of a Message.

  The Message class is defined as an extension of the Info Object class.

  The methods defined are for the local implementation the Message class.

3.2.1.8.1.1 Method Summary

All Methods	
Modifier and Type	**Method and Description**
`Identifier`	`getIdentifier()`

	The getIdentifier method returns the identifier of this message.
Identifier	getReceiverId() The getReceiverId method returns the identifier of the receiver
Object	getReceiverIO() The getReceiverIO method returns the object from the receiver.
Identifier	getSenderId() The getSenderId method returns the identifier of the sender
Object	getSenderIO() The getSenderIO method returns the object from the sender.
void	setIdentifier(Identifier identifier) The setIdentifier method sets the identifier of this message
void	setReceiverId(Identifier receiverId) The setReceiverId method sets the identifier of the receiver
void	setReceiverIO(Object receiverObject) The setReceiverIO method sets the object from the receiver receiver.
void	setSenderId(Identifier senderId) The setSenderId method sets the identifier of the sender receiver.
void	setSenderIO(Object senderObject) The setSenderIO method sets the object from the sender

3.2.1.8.1.2 Methods inherited from interface InfoObjectInterface

getDataObject, getDataObjectID, getInfoObjectID, getRepInfo, getRepInfoDataObject, getRepInfoDataObjectID, setDataObject, setInfoObjectID, setRepInfoDataObject

3.2.1.8.1.3 Method Detail

3.2.1.8.1.3.1 getIdentifier

```
Identifier getIdentifier()
```

The getIdentifier method returns the identifier of this message.

Returns: Identifier the identifier of this message

3.2.1.8.1.3.2 getSenderId

```
Identifier getSenderId()
```

The getSenderId method returns the identifier of the sender

Returns: Identifier the identifier of the sender

3.2.1.8.1.3.3 getReceiverId

```
Identifier getReceiverId()
```

The getReceiverId method returns the identifier of the receiver

Returns: Identifier the identifier of the receiver

3.2.1.8.1.3.4 getSenderIO

```
Object getSenderIO()
```

The getSenderIO method returns the object from the sender.

Returns: Object the Info Object from the sender.

3.2.1.8.1.3.5 getReceiverIO

```
Object getReceiverIO()
```

The getReceiverIO method returns the object from the receiver. receiver.

Returns: Object the Info Object from the receiver.

3.2.1.8.1.3.6 setIdentifier

```
void setIdentifier(Identifier identifier)
```

The setIdentifier method sets the identifier of this message

Parameters: `identifier` - the identifier of the message

3.2.1.8.1.3.7 setSenderId

```
void setSenderId(Identifier senderId)
```

The setSenderId method sets the identifier of the sender receiver.

Parameters: `senderId` - the identifier of the sender

3.2.1.8.1.3.8 setReceiverId

```
void setReceiverId(Identifier receiverId)
```

The setReceiverId method sets the identifier of the receiver

Parameters: `receiverId` - the identifier of the receiver

3.2.1.8.1.3.9 setSenderIO

```
void setSenderIO(Object senderObject)
```

The setSenderIO method sets the object from the sender

Parameters: `senderObject` - the object being sent from the sender to the receiver

3.2.1.8.1.3.10 setReceiverIO

```
void setReceiverIO(Object receiverObject)
```

The setReceiverIO method sets the object from the receiver receiver.

Parameters: `receiverObject` - the object being sent from the receiver to the sender

3.2.1.9 Request

The Request class provides is the "Request" messaging interaction protocol where a message is sent to a receiver and a response is expected from the receiver.

3.2.1.9.1 Request_Interface

The Request Interface is a well-defined entry point for an interaction protocol that sends a message and receives a response. The interface is an element of the Adapter Layer component. This section is normative.

- All Known Implementing Classes: Request

```
public interface RequestInterface
```

The Request Interface is a well-defined entry point for message enqueue and dequeue.

3.2.1.9.1.1 Method Summary

All Methods	
Modifier and Type	**Method and Description**
`boolean`	`RequestDequeue(Identifier messageID)` The RequestDequeue method dequeues the provided message
`void`	`RequestEnqueue(Identifier messageID, Message message)` The Request Enqueue method enqueues the provided message

3.2.1.9.1.2 Method Detail

3.2.1.9.1.2.1 RequestEnqueue

```
void RequestEnqueue(Identifier messageID, Message message)
```

The Request Enqueue method enqueues the message to be sent.

Parameters:
`messageID` - the identifier of the Message to enqueue
`message` - the Message to enqueue

3.2.1.9.1.2.2 RequestDequeue

```
boolean RequestDequeue(Identifier messageID)
```

The RequestDequeue method dequeues the message returned.

Parameters:
`messageID` - the identifier of the Message to dequeue

Returns:
boolean - true if the dequeue was successful

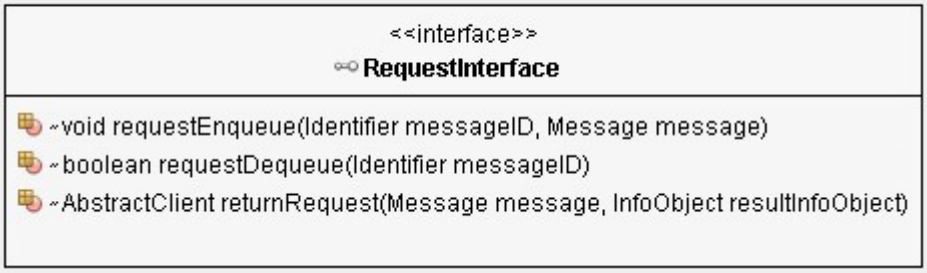

Figure 12 - Request Interface

3.2.2 INFORMATION MODEL

The Information Model describe the types of information that are exchanged and managed within an OAIS. This subsection also defines the specific Information Objects that are used within the OAIS to preserve and access the information entrusted to the Archive. This detailed model of OAIS-related Information Objects is intended to aid the architect or designer of future OAIS systems. The objects discussed in this subsection are conceptual and should not be taken to imply any specific implementations. [C2] However the Interfaces described within this document defined normative methods for getting and putting each Information Object.

An information model is a representation of concepts and the relationships, constraints, rules, and operations to specify data semantics for a chosen domain of discourse. This section is normative.

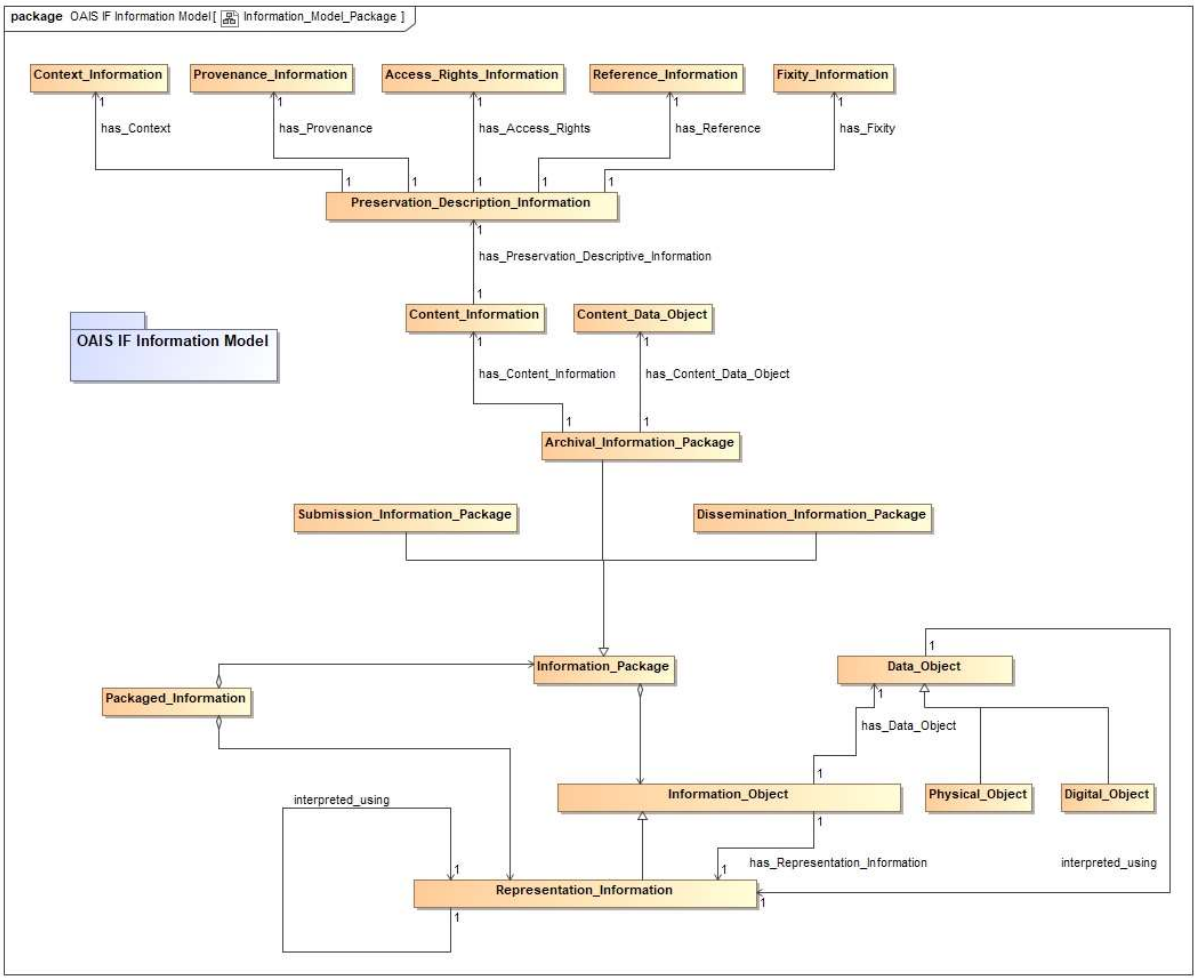

Figure 13 - OAIS Information Model

3.2.2.1 Access_Rights_Information

Access Rights Information: The information that identifies the access restrictions pertaining to the Content Information, including the legal framework, licensing terms, and access control. It contains the access and distribution conditions stated within the Submission Agreement, related to both preservation (by the OAIS) and final usage (by the Consumer). It also includes the specifications for the application of rights enforcement measures. [C1]

The Access Rights Information object class implements the Access Rights Information Interface. The Access Rights Information class is also a subclass of the Information Object class and inherits its properties. It implements the Information Object Interface, is composed of a Data Object and Representation Information, is managed by an Archival Storage functional entity, and is an element of the OAIS Interoperability Framework component. This section is normative.

3.2.2.2 Archival_Information_Package

Archival Information Package (AIP): An Information Package, consisting of the Content Information and the associated Preservation Description Information (PDI), which is preserved within an OAIS. [C1]

The Archival Information Package object class is a subclass of the Information Package class and is composed of a Content Information class and a Preservation Description Information (PDI) class. It is also indirectly a subclass of the Information Object class and inherits its properties. It implements the Information Object Interface, is composed of a Data Object and Representation Information, is managed by an Archival Storage functional entity, and is an element of the OAIS Interoperability Framework component. This section is normative.

3.2.2.3 Content_Data_Object

Content Data Object: The Data Object, that together with associated Representation Information, comprises the Content Information. [C1]

The Content Data Object is the Data object of Content Information. It has associated Representation Information, is managed by an Archival Storage functional entity, and is an element of the OAIS Interoperability Framework component. This section is normative.

3.2.2.4 Content_Information

Content Information: A set of information that is the original target of preservation or that includes part or all of that information. It is an Information Object composed of its Content Data Object and its Representation Information. [C1]

The Content Information object class is composed of a Content Data Object and a Representation Information class. It implements the Information Object Interface, is managed by an Archival Storage functional entity, and is an element of the OAIS Interoperability Framework component. Content Information is further described by Preservation Descriptive Information. This section is normative.

3.2.2.5 Context_Information

Context Information: The information that documents the relationships of the Content Information to its environment. This includes why the Content Information was created and how it relates to other Content Information objects. [CC]

The Context Information object class implements the Context Information Interface. The Context Information class is also a subclass of the Information Object class and inherits its properties. It implements the Information Object Interface, is composed of a Data Object and Representation Information, is managed by an Archival Storage functional entity, and is an element of the OAIS Interoperability Framework component. This section is normative.

3.2.2.6 Dissemination_Information_Package

Dissemination Information Package (DIP): An Information Package, derived from one or more AIPs, and sent by Archives to the Consumer in response to a request to the OAIS. [C1]

The Dissemination Information Package object class is a subclass of the Information Package class and is composed of a Content Information class and a Preservation Description Information (PDI) class. It is also indirectly a subclass of the Information Object class and inherits its properties. It implements the Information Object Interface, is composed of a Data Object and Representation Information, is managed by an Archival Storage functional entity, and is an element of the OAIS Interoperability Framework component. This section is normative.

3.2.2.7 Fixity_Information

Fixity Information: The information which documents the mechanisms that ensure that the Content Information object has not been altered in an undocumented manner. An example is a Cyclical Redundancy Check (CRC) code for a file. [C1]

The Fixity Information object class implements the Fixity Information Interface. The Fixity Information class is also a subclass of the Information Object class and inherits its properties. It implements the Information Object Interface, is composed of a Data Object and Representation Information, is managed by an Archival Storage functional entity, and is an element of the OAIS Interoperability Framework component. This section is normative.

3.2.2.8 Information_Package

Information Package: A logical 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. [C1]

The Information Package object class is composed of a Content Information class and a Preservation Description Information (PDI) class. It is a subclass of the Information Object class and inherits its properties. It implements the Information Object Interface, is composed of a Data Object and Representation Information, is managed by an Archival Storage functional entity, and is an element of the OAIS Interoperability Framework component. This section is normative.

3.2.2.9 Preservation_Description_Information

Preservation Description Information (PDI): The information which is necessary for adequate preservation of the Content Information and which can be categorized as Provenance, Reference, Fixity, Context, and Access Rights Information. [C1]

The Preservation Description Information object class is composed of Access Rights Information, Context Information, Fixity Information, Provenance Information, and Reference Information. It provides preservation description for Content Information. It is also a subclass of the Information Object class and inherits its properties. It implements the Information Object Interface, is managed by

an Archival Storage functional entity, and is an element of the OAIS Interoperability Framework component. This section is normative.

3.2.2.10 Provenance_Information

Provenance Information: The information that documents the history of the Content Information. This information tells the origin or source of the Content Information, any changes that may have taken place since it was originated, and who has had custody of it since it was originated. The Archive is responsible for creating and preserving Provenance Information from the point of Ingest; however, earlier Provenance Information should be provided by the Producer. Provenance Information adds to the evidence to support Authenticity. [C1]

The Provenance Information object class implements the Provenance Information Interface. The Provenance Information class is also a subclass of the Information Object class and inherits its properties. It implements the Information Object Interface, is composed of a Data Object and Representation Information, is managed by an Archival Storage functional entity, and is an element of the OAIS Interoperability Framework component. This section is normative.

3.2.2.11 Reference_Information

Reference Information: The information that is used as an identifier for the Content Information. It also includes identifiers that allow outside systems to refer unambiguously to a particular Content Information. An example of Reference Information is an ISBN. [C1]

The Reference Information object class implements the Reference Information Interface. The Reference Information class is also a subclass of the Information Object class and inherits its properties. It implements the Information Object Interface, is composed of a Data Object and Representation Information, is managed by an Archival Storage functional entity, and is an element of the OAIS Interoperability Framework component. This section is normative.

3.2.2.12 Representation_Information

Representation Information: The information that maps a Data Object into more meaningful concepts. An example of Representation Information for a bit sequence which is a FITS file might consist of the FITS standard which defines the format plus a dictionary which defines the meaning in the file of keywords which are not part of the standard. Another example is JPEG software which is used to render a JPEG file; rendering the JPEG file as bits is not very meaningful to humans but the software, which embodies an understanding of the JPEG standard, maps the bits into pixels which can then be rendered as an image for human viewing. [C1]

The Representation Information object class implements the Representation Information Interface. The Representation Information class is also a subclass of the Information Object class and inherits its properties. It implements the Information Object Interface, is composed of a Data Object and Representation Information, is managed by an Archival Storage functional entity, and is an element of the OAIS Interoperability Framework component. This section is normative.

3.2.2.13 Submission_Information_Package

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. [C1]

The Submission Information Package object class is a subclass of the Information Package class and is composed of a Content Information class and a Preservation Description Information (PDI) class. It is also indirectly a subclass of the Information Object class and inherits its properties. It implements the Information Object Interface, is composed of a Data Object and Representation Information, is managed by an Archival Storage functional entity, and is an element of the OAIS Interoperability Framework component. This section is normative.

3.2.3 INTERFACES

An Interface is the abstraction of a service that only defines the operations supported by that service, but not their implementations. This section is normative.

3.2.3.1.1 Access_Rights_Information_Interface

The Access Rights Information Interface is a well-defined entry point for accessing Access Rights Information. The interface is a subclass of the Information Object Interface and inherits its methods. The interface is an element of the Abstraction Layer component. This section is normative.

All Superinterfaces: InfoObjectInterface

```
public interface AccessRightsInformationInterface extends
InfoObjectInterface
```

The Access Rights Information Interface is a well-defined entry point and contract for getting and putting this Information Object and its components.

Access Rights Information is a subclass of Information Object and inherits methods for getting and putting the component Representation Information and Digital Object of this Information Object.

Access Rights Information: The information that identifies the access restrictions pertaining to the Content Information, including the legal framework, licensing terms, and access control. It contains the access and distribution conditions stated within the Submission Agreement, related to both preservation (by the OAIS) and final usage (by the Consumer). It also includes the specifications for the application of rights enforcement measures.

3.2.3.1.1.1 Method Summary

3.2.3.1.1.2 Methods inherited from interface InfoObjectInterface

getDataObject, getDataObjectID, getInfoObjectID, getRepInfo, getRepInfoDataObject, getRepInfoDataObjectID, setDataObject, setInfoObjectID, setRepInfoDataObject

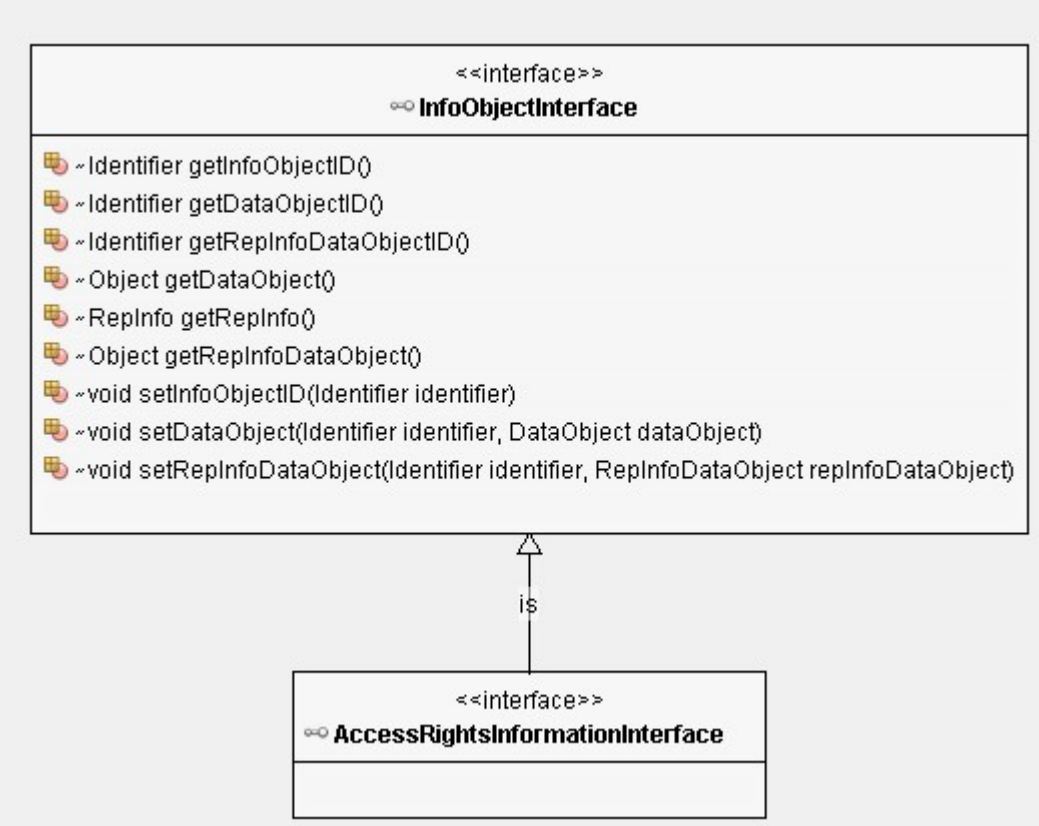

Figure 14 - Access Rights Information Interface

3.2.3.1.2 Archival_Information_Package_Interface

All Superinterfaces: InfoObjectInterface, InformationPackageInterface

```
public interface ArchivalInformationPackageInterface extends
InformationPackageInterface
```

The Archival Information Package Interface is a well-defined entry point and contract for getting and putting this Information Object and its components.

Archival Information Package is a subclass of Information Object and inherits methods for getting and putting the component Representation Information and Data Object of this Information Object. The Data Object is a container for the Content Information and Preservation Description Information.

Archival Information Package (AIP): An Information Package, consisting of the Content Information and the associated Preservation Description Information (PDI), which is preserved within an OAIS.

3.2.3.1.2.1 Method Summary

All Methods	
Modifier and Type	**Method and Description**
Object	getContentInformation() The getContentInformation method returns the Content Information component of this Archival Information Package.
Object	getPreservationDescriptionInformation() The getPreservationDescriptionInformation method returns the Preservation Description Information component of this Archival Information Package.
void	setContentInformation(Identifier identifier, InfoObject infoObject) The setContentInformation method sets the Content Information component of this Archival Information Package.
void	setPreservationDescriptionInformation(Identifier identifier, InfoObject infoObject) The setPreservationDescriptionInformation method sets the Preservation Description Information component of this Archival Information Package.

3.2.3.1.2.2 Methods inherited from interface InfoObjectInterface

getDataObject, getDataObjectID, getInfoObjectID, getRepInfo, getRepInfoDataObject, getRepInfoDataObjectID, setDataObject, setInfoObjectID, setRepInfoDataObject

3.2.3.1.2.3 Method Detail

3.2.3.1.2.3.1 getContentInformation

```
Object getContentInformation()
```

The getContentInformation method returns the Content Information component of this Archival Information Package.

Returns:
Object - the Content Information for this InfoObject

3.2.3.1.2.3.2 getPreservationDescriptionInformation

```
Object getPreservationDescriptionInformation()
```

The getPreservationDescriptionInformation method returns the Preservation Description Information component of this Archival Information Package.

Returns:
Object - the Preservation Description Information for this InfoObject

3.2.3.1.2.3.3 setContentInformation

```
void setContentInformation(Identifier identifier,
                           InfoObject infoObject)
```

The setContentInformation method sets the Content Information component of this Archival Information Package.

Parameters:
`identifier` - the identifier for this infoObject
`infoObject` - the Content Information component for this InfoObject

3.2.3.1.2.3.4 setPreservationDescriptionInformation

```
void setPreservationDescriptionInformation(
      Identifier identifier, InfoObject infoObject)
```

The setPreservationDescriptionInformation method sets the Preservation Description Information component of this Archival Information Package.

Parameters:
`identifier` - the identifier for this infoObject
`infoObject` - the Preservation Description Information component for this InfoObject

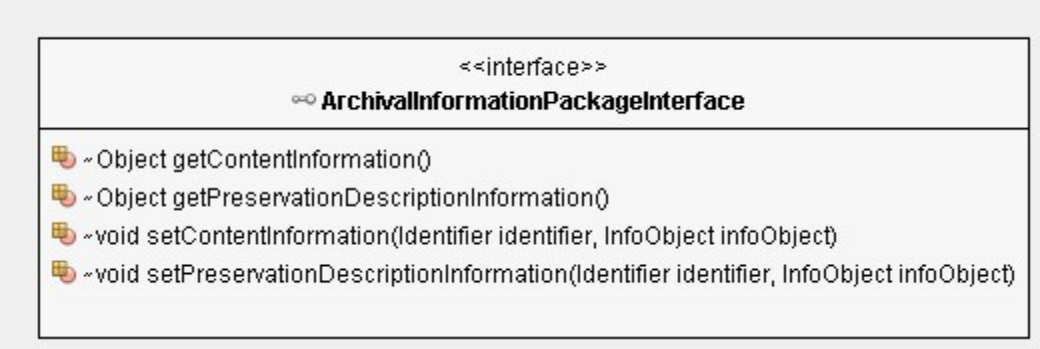

Figure 15 - Archival Information Package Interface

3.2.3.1.3 Content_Data_Object_Interface

The Content Data Object Interface is a well-defined entry point for getting and putting the Identifier and Object of a Content Data Object. The interface is a subclass of the Information Object Interface and inherits its methods. The interface is an element of the Abstraction Layer component. This section is normative.

All Superinterfaces: DataObjectInterface

```
public interface ContentDataObjectInterface extends DataObjectInterface
```

Content Data Object is a subclass of Data Object and inherits methods for getting and putting the Object and Identifier of this Content Data Object.

Content Data Object: The Data Object, that together with associated Representation Information, comprises the Content Information.

3.2.3.1.3.1 Method Summary

3.2.3.1.3.2 Methods inherited from interface DataObjectInterface
`getIdentifier`, `getObject`, `setObject`

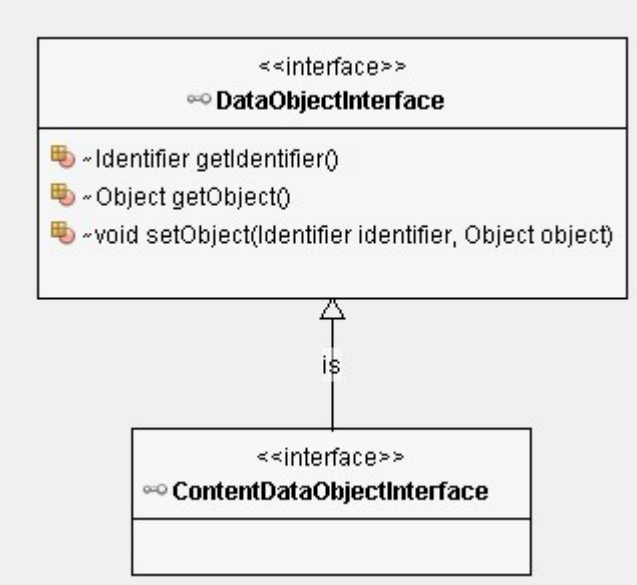

Figure 16 - Content Data Object Interface

3.2.3.1.4 Content_Information_Interface

The Content Information Interface is a well-defined entry point for accessing Content Information. The interface is a subclass of the Information Object Interface and inherits its methods. The interface is an element of the Abstraction Layer component. This section is normative.

All Superinterfaces: InfoObjectInterface

```
public interface ContentInformationInterface extends InfoObjectInterface
```

The Content Information Interface is a well-defined entry point and contract for getting and putting this Information Object and its components.

Content Information is a subclass of Information Object and inherits methods for getting and putting the component Representation Information and Digital Object of this Information Object.

Content Information: A set of information that is the original target of preservation or that includes part or all of that information. It is an Information Object composed of its Content Data Object and its Representation Information.

3.2.3.1.4.1 Method Summary

All Methods	
Modifier and Type	**Method and Description**
Object	getContentDataObject() The getContentDataObject method returns the Content Data Object for this Content Information.
void	setContentDataObject(Identifier identifier, Object object) The setContentDataObject method sets the Content Data Object for this Content Information.

3.2.3.1.4.2 Methods inherited from interface InfoObjectInterface
getDataObject, getDataObjectID, getInfoObjectID, getRepInfo, getRepInfoDataObject, getRepInfoDataObjectID, setDataObject, setInfoObjectID, setRepInfoDataObject

3.2.3.1.4.3 Method Detail

3.2.3.1.4.3.1 getContentDataObject

```
Object getContentDataObject()
```

The getContentDataObject method returns the Content Data Object for this Content Information.

Returns:
Object - the contentDataObject for this contentInformation

3.2.3.1.4.3.2 setContentDataObject

```
void setContentDataObject(Identifier identifier, Object object)
```

The setContentDataObject method sets the Content Data Object for this Content Information.

Parameters:
`identifier` - the identifier for this contentDataObject
`object` - the contentDataObject for this contentInformation

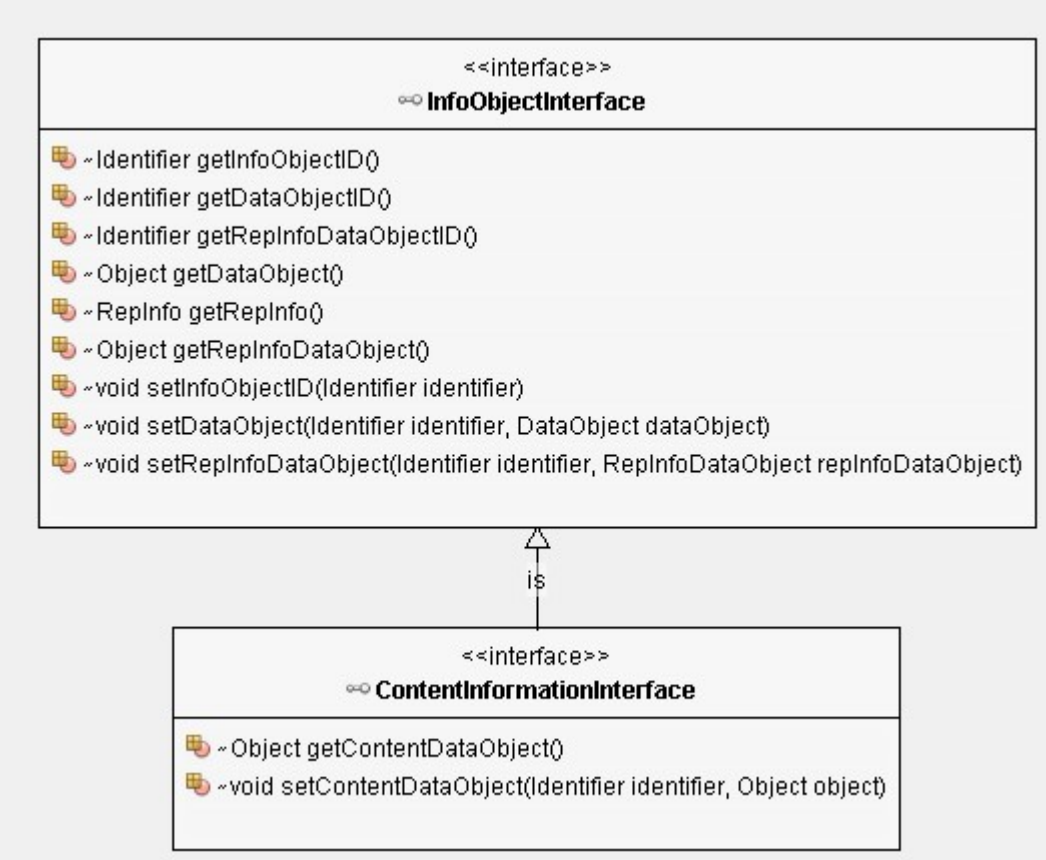

Figure 16 - Content Information Interface

3.2.3.1.5 Context_Information_Interface

The Context Information Interface is a well-defined entry point for accessing Context Information. The interface is a subclass of the Information Object Interface and inherits its methods. The interface is an element of the Abstraction Layer component. This section is normative.

All Superinterfaces: InfoObjectInterface

```
public interface ContextInformationInterface extends InfoObjectInterface
```

The Context Information Interface is a well-defined entry point and contract for getting and putting this Information Object and its components.

Context Information is a subclass of Information Object and inherits methods for getting and putting the component Representation Information and Digital Object of this Information Object.

Context Information is a subclass of Information Object and inherits Context Information: The information that documents the relationships of the Content Information to its environment.

This includes why the Content Information was created and how it relates to other Content Information objects.

3.2.3.1.5.1 Method Summary

3.2.3.1.5.2 Methods inherited from interface InfoObjectInterface

getDataObject, getDataObjectID, getInfoObjectID, getRepInfo, getRepInfoDataObject, getRepInfoDataObjectID, setDataObject, setInfoObjectID, setRepInfoDataObject

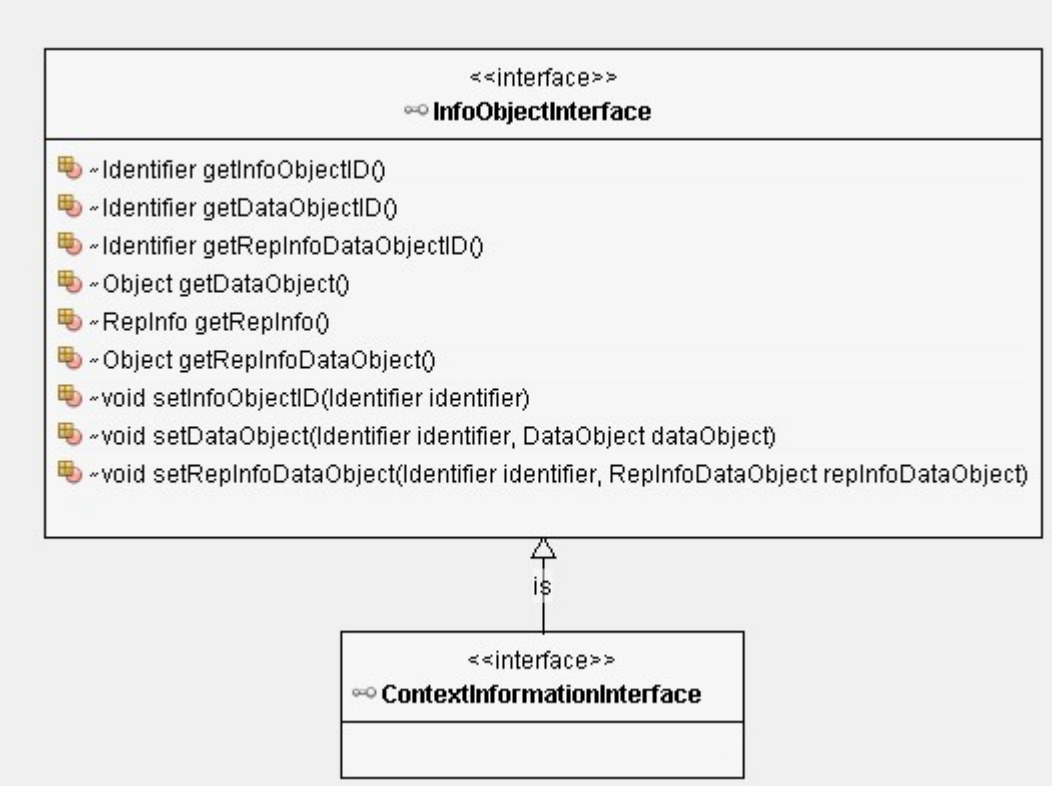

Figure 17 - Context Information Interface

3.2.3.1.6 Fixity_Information_Interface

The Fixity Information Interface is a well-defined entry point for accessing Fixity Information. The interface is a subclass of the Information Object Interface and inherits its methods. The interface is an element of the Abstraction Layer component. This section is normative.

All Superinterfaces: InfoObjectInterface

```
public interface FixityInformationInterface extends InfoObjectInterface
```

The Fixity Information Interface is a well-defined entry point and contract for getting and putting this Information Object and its components.

Fixity Information is a subclass of Information Object and inherits methods for getting and putting the component Representation Information and Digital Object of this Information Object.

Fixity Information: The information which documents the mechanisms that ensure that the Content Information object has not been altered in an undocumented manner. An example is a Cyclical Redundancy Check (CRC) code for a file.

3.2.3.1.6.1 Method Summary

3.2.3.1.6.2 Methods inherited from interface InfoObjectInterface

getDataObject, getDataObjectID, getInfoObjectID, getRepInfo, getRepInfoDataObject, getRepInfoDataObjectID, setDataObject, setInfoObjectID, setRepInfoDataObject

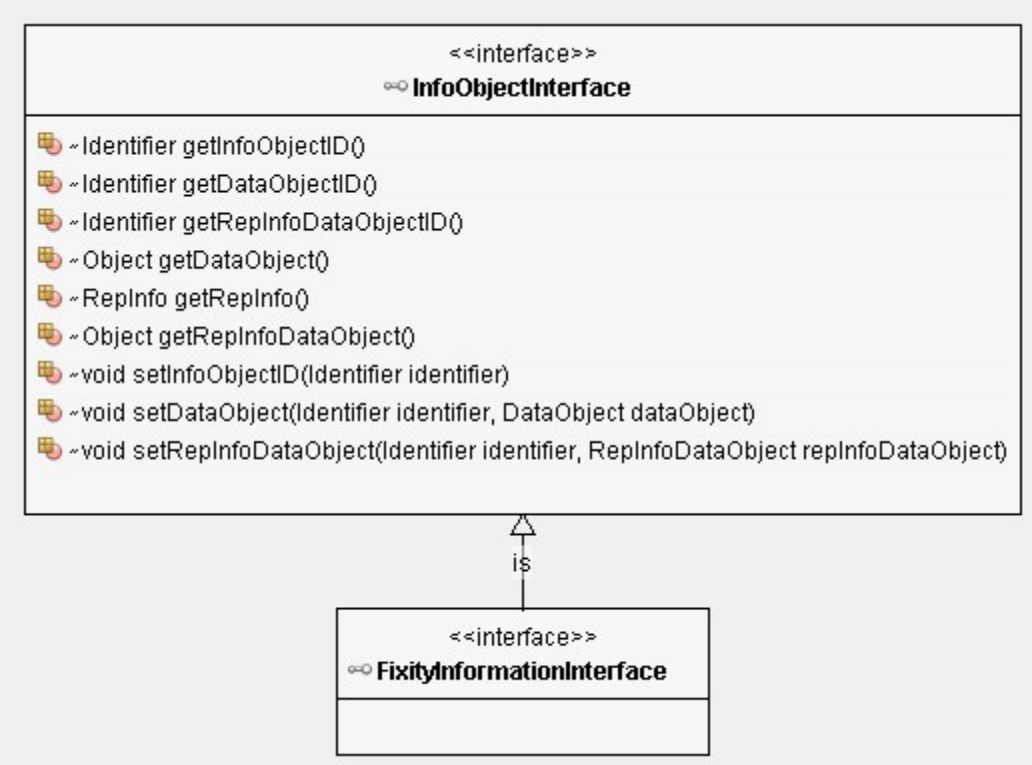

Figure 18 - Fixity Information Interface

3.2.3.1.7 Information_Package_Interface

All Superinterfaces: InfoObjectInterface

All Known Subinterfaces: ArchivalInformationPackageInterface

```
public interface InformationPackageInterface extends InfoObjectInterface
```

The Information Package Interface is a well-defined entry point and contract for getting and putting this Information Object and its components.

Information Package is a subclass of Information Object and inherits methods for getting and putting the component Representation Information and Data Object of this Information Object.

Information Package: A logical container composed of optional Content Information and optional associated Preservation Description Information.

3.2.3.1.7.1 Method Summary

3.2.3.1.7.2 Methods inherited from interface InfoObjectInterface

getDataObject, getDataObjectID, getInfoObjectID, getRepInfo, getRepInfoDataObject, getRepInfoDataObjectID, setDataObject, setInfoObjectID, setRepInfoDataObject

3.2.3.1.8 Packaged_Information_Interface

The Packaged Information Interface is a well-defined entry point for accessing Packaged Information. The interface is a subclass of the Information Object Interface and inherits its methods. The interface is an element of the Abstraction Layer component. This section is normative.

All Superinterfaces: InfoObjectInterface

```
public interface PackagedInformationInterface extends InfoObjectInterface
```

The Packaged Information Interface is a well-defined entry point and contract for getting and putting this Information Object and its components.

Packaged Information is a subclass of Information Object and inherits methods for getting and putting the component Representation Information and Digital Object of this Information Object.

3.2.3.1.8.1 Method Summary

3.2.3.1.8.2 Methods inherited from interface InfoObjectInterface

getDataObject, getDataObjectID, getInfoObjectID, getRepInfo, getRepInfoDataObject, getRepInfoDataObjectID, setDataObject, setInfoObjectID, setRepInfoDataObject

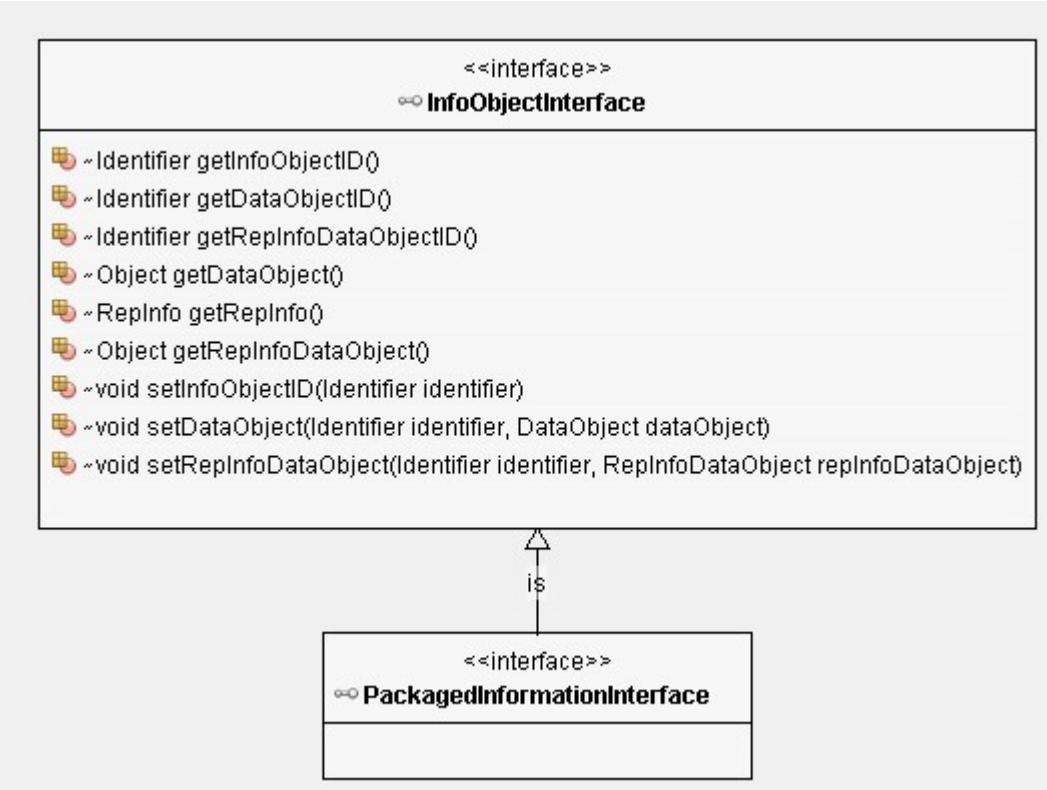

Figure 19 - Packaged Information Interface

3.2.3.1.9 Preservation_Description_Information_Interface

All Superinterfaces: InfoObjectInterface

```
public interface PreservationDescriptionInformationInterface extends
InfoObjectInterface
```

The Preservation Description Information Interface is a well-defined entry point and contract for getting and putting this Information Object and its components.

Preservation Description Information is a subclass of Information Object and inherits methods for getting and putting the component Representation Information and Data Object of this Information Object. The Data Object is a container for Provenance, Reference, Fixity, Context, and Access Rights Information.

Preservation Description Information (PDI): The information which is necessary for adequate preservation of the Content Information and which can be categorized as Provenance, Reference, Fixity, Context, and Access Rights Information.

3.2.3.1.9.1 Method Summary

All Methods	
Modifier and Type	**Method and Description**
Object	getAccessRightsInformation() The getAccessRightsInformation method returns the Access Rights Information component of this Preservation Description Information Object.
Object	getContextInformation() The getContextInformation method returns the Context Information component of this Preservation Description Information Object.
Object	getFixityInformation() The getFixityInformation method returns the Fixity Information component of this Preservation Description Information Object.
Object	getProvenanceInformation() The getProvenanceInformation method returns the Provenance Information component of this Preservation Description Information Object.
Object	getReferenceInformation() The getReferenceInformation method returns the Reference Information component of this Preservation Description Information Object.
void	setAccessRightsInformation(Identifier identifier, InfoObject infoObject) The setAccessRightsInformation method sets the Access Rights Information component of this Preservation Description Information Object.
void	setContextInformation(Identifier identifier, InfoObject infoObject)

	The setContextInformation method sets the Context Information component of this Preservation Description Information Object.
void	setFixityInformation(Identifier identifier, InfoObject infoObject) The setFixityInformation method sets the Fixity Information component of this Preservation Description Information Object.
void	setProvenanceInformation(Identifier identifier, InfoObject infoObject) The setProvenanceInformation method sets the Provenance Information component of this Preservation Description Information Object.
void	setReferenceInformation(Identifier identifier, InfoObject infoObject) The setReferenceInformation method sets the Reference Information component of this Preservation Description Information Object.

3.2.3.1.9.2 Methods inherited from interface InfoObjectInterface
getDataObject, getDataObjectID, getInfoObjectID, getRepInfo, getRepInfoDataObject, getRepInfoDataObjectID, setDataObject, setInfoObjectID, setRepInfoDataObject

3.2.3.1.9.3 Method Detail

3.2.3.1.9.3.1 getProvenanceInformation

```
Object getProvenanceInformation()
```

The getProvenanceInformation method returns the Provenance Information component of this Preservation Description Information Object.

Returns:
Object - the Provenance Information for this InfoObject

3.2.3.1.9.3.2 getReferenceInformation

```
Object getReferenceInformation()
```

The getReferenceInformation method returns the Reference Information component of this Preservation Description Information Object.

Returns:
Object - the Reference Information for this InfoObject

3.2.3.1.9.3.3 getFixityInformation

```
Object getFixityInformation()
```

The getFixityInformation method returns the Fixity Information component of this Preservation Description Information Object.

Returns:
Object - the Fixity Information for this InfoObject

3.2.3.1.9.3.4 getContextInformation

```
Object getContextInformation()
```

The getContextInformation method returns the Context Information component of this Preservation Description Information Object.

Returns:
Object - the Context Information for this InfoObject

3.2.3.1.9.3.5 getAccessRightsInformation

```
Object getAccessRightsInformation()
```

The getAccessRightsInformation method returns the Access Rights Information component of this Preservation Description Information Object.

Returns:
Object - the Access Rights Information for this InfoObject

3.2.3.1.9.3.6 setProvenanceInformation

```
void setProvenanceInformation(Identifier identifier,
                              InfoObject infoObject)
```

The setProvenanceInformation method sets the Provenance Information component of this Preservation Description Information Object.

Parameters:
`identifier` - the identifier for this infoObject
`infoObject` - the Provenance Information component for this InfoObject

3.2.3.1.9.3.7 setReferenceInformation

```
void setReferenceInformation(Identifier identifier,
                             InfoObject infoObject)
```

The setReferenceInformation method sets the Reference Information component of this Preservation Description Information Object.

Parameters:
`identifier` - the identifier for this infoObject
`infoObject` - the Reference Information component for this InfoObject

3.2.3.1.9.3.8 setFixityInformation

```
void setFixityInformation(Identifier identifier,
                          InfoObject infoObject)
```

The setFixityInformation method sets the Fixity Information component of this Preservation Description Information Object.

Parameters:
`identifier` - the identifier for this infoObject
`infoObject` - the Fixity Information component for this InfoObject

3.2.3.1.9.3.9 setContextInformation

```
void setContextInformation(Identifier identifier,
                           InfoObject infoObject)
```

The setContextInformation method sets the Context Information component of this Preservation Description Information Object.

Parameters:
`identifier` - the identifier for this infoObject
`infoObject` - the Context Information component for this InfoObject

3.2.3.1.9.3.10 setAccessRightsInformation

```
void setAccessRightsInformation(Identifier identifier,
                                InfoObject infoObject)
```

The setAccessRightsInformation method sets the Access Rights Information component of this Preservation Description Information Object.

Parameters:

`identifier` - the identifier for this infoObject
`infoObject` - the Access Rights Information component for this InfoObject

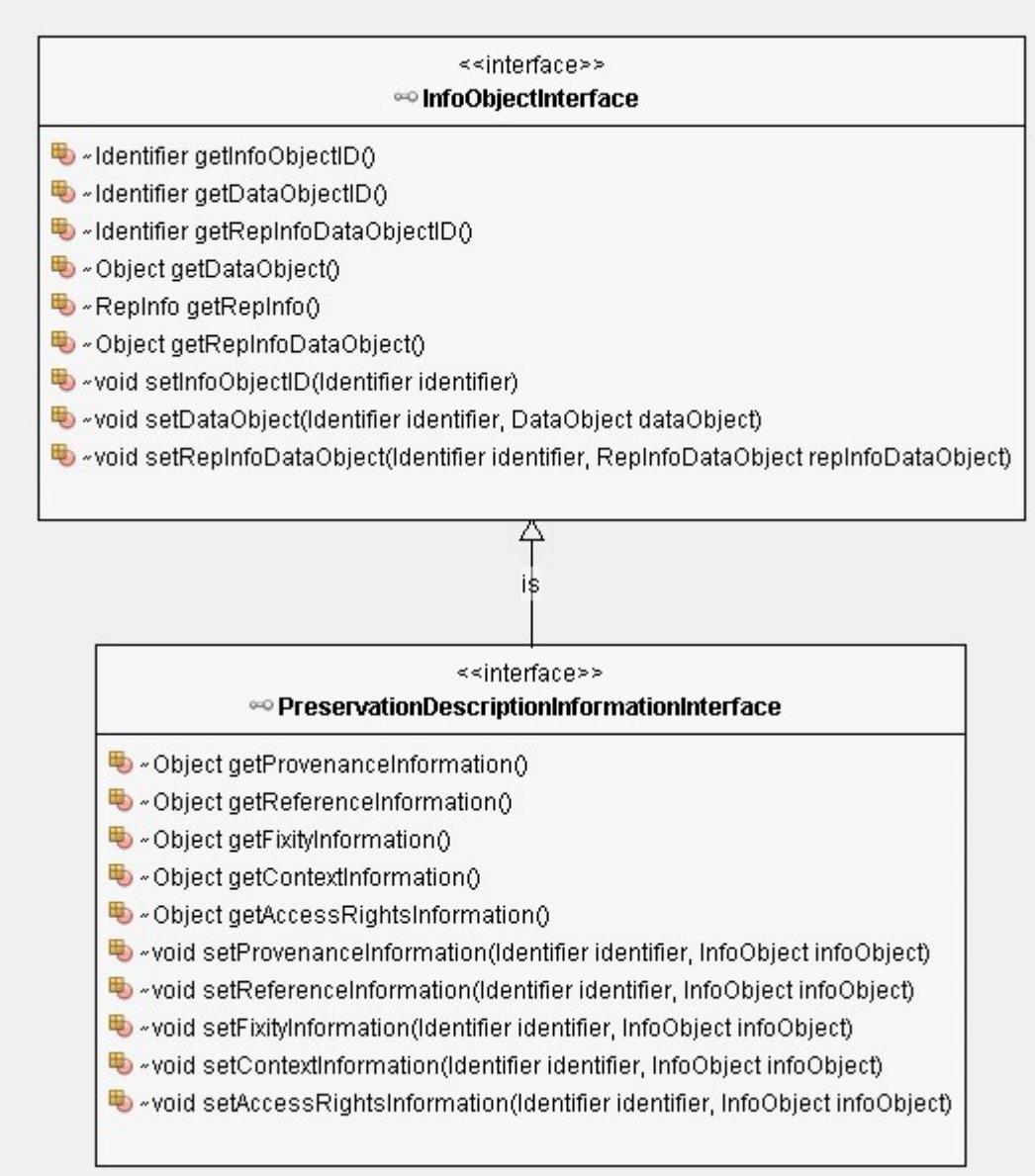

Figure 20 - Preservation Description Information Interface

3.2.3.1.10 Provenance_Information_Interface

The Provenance Information Interface is a well-defined entry point for accessing Provenance Information. The interface is a subclass of the Information Object Interface and inherits its methods. The interface is an element of the Abstraction Layer component. This section is normative.

All Superinterfaces: InfoObjectInterface

```
public interface ProvenanceInformationInterface extends InfoObjectInterface
```

The Provenance Information Interface is a well-defined entry point and contract for getting and putting this Information Object and its components.

Provenance Information is a subclass of Information Object and inherits methods for getting and putting the component Representation Information and Digital Object of this Information Object.

Provenance Information: The information that documents the history of the Content Information. This information tells the origin or source of the Content Information, any changes that may have taken place since it was originated, and who has had custody of it since it was originated. The Archive is responsible for creating and preserving Provenance Information from the point of Ingest; however, earlier Provenance Information should be provided by the Producer. Provenance Information adds to the evidence to support Authenticity.

3.2.3.1.10.1 Method Summary

3.2.3.1.10.2 Methods inherited from interface InfoObjectInterface

getDataObject, getDataObjectID, getInfoObjectID, getRepInfo, getRepInfoDataObject, getRepInfoDataObjectID, setDataObject, setInfoObjectID, setRepInfoDataObject

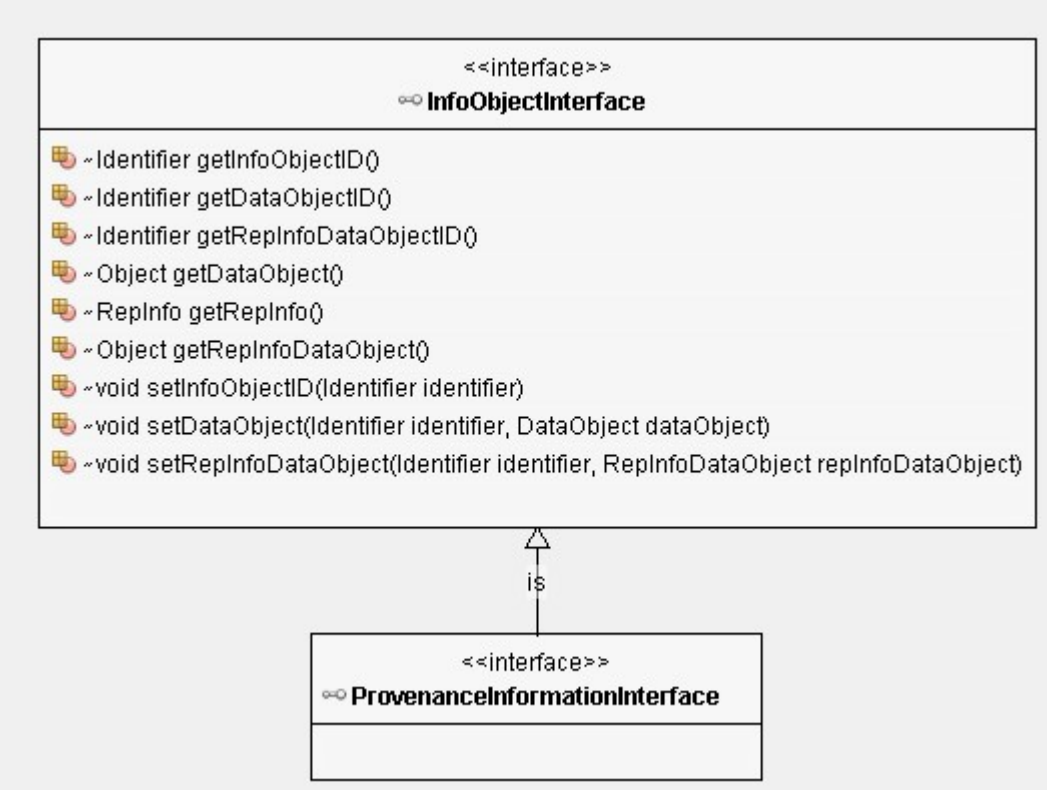

Figure 21 - Provenance Information Interface

3.2.3.1.11 Reference_Information_Interface

The Reference Information Interface is a well-defined entry point for accessing Reference Information. The interface is a subclass of the Information Object Interface and inherits its methods. The interface is an element of the Abstraction Layer component. This section is normative.

All Superinterfaces: InfoObjectInterface

```
public interface ReferenceInformationInterface extends InfoObjectInterface
```

The Reference Information Interface is a well-defined entry point and contract for getting and putting this Information Object and its components.

Reference Information is a subclass of Information Object and inherits methods for getting and putting the component Representation Information and Digital Object of this Information Object.

Reference Information: The information that is used as an identifier for the Content Information. It also includes identifiers that allow outside systems to refer unambiguously to a particular Content Information. An example of Reference Information is an ISBN.

3.2.3.1.11.1 Method Summary

3.2.3.1.11.2 Methods inherited from interface InfoObjectInterface

getDataObject, getDataObjectID, getInfoObjectID, getRepInfo, getRepInfoDataObject, getRepInfoDataObjectID, setDataObject, setInfoObjectID, setRepInfoDataObject

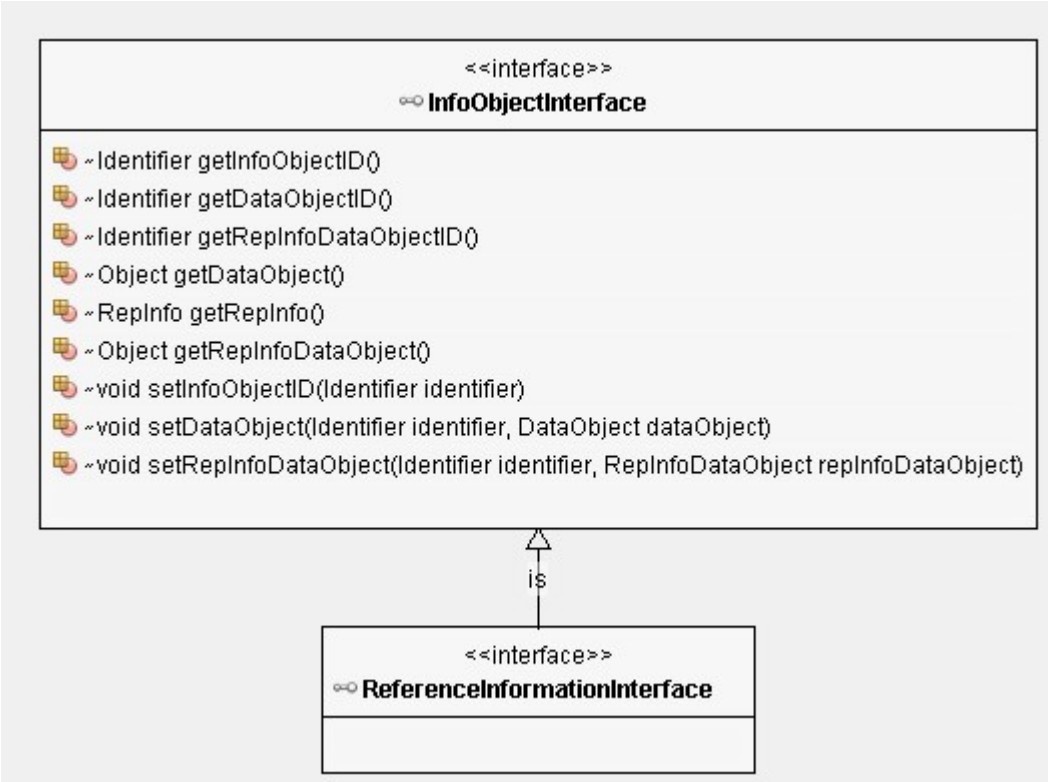

Figure 22 – Reference Information Interface

3.2.3.1.12 Representation_Information_Interface

The Representation Information Interface is a well-defined entry point for accessing Representation Information. The interface is a subclass of the Information Object Interface and inherits its methods. The interface is an element of the Abstraction Layer component. This section is normative.

All Superinterfaces: InfoObjectInterface

```
public interface RepresentationInformationInterface extends
InfoObjectInterface
```

The Representation Information Interface is a well-defined entry point and contract for getting and putting this Information Object and its components.

Representation Information is a subclass of Information Object and inherits methods for getting and putting the component Representation Information and Digital Object of this Information Object.

Representation Information: The information that maps a Data Object into more meaningful concepts. An example of Representation Information for a bit sequence which is a FITS file might consist of the FITS standard which defines the format plus a dictionary which defines the meaning in the file of keywords which are not part of the standard. Another example is JPEG software which is used to render a JPEG file; rendering the JPEG file as bits is not very meaningful to humans but the software, which embodies an understanding of the JPEG standard, maps the bits into pixels which can then be rendered as an image for human viewing.

3.2.3.1.12.1 Method Summary

3.2.3.1.12.2 Methods inherited from interface InfoObjectInterface

getDataObject, getDataObjectID, getInfoObjectID, getRepInfo, getRepInfoDataObject, getRepInfoDataObjectID, setDataObject, setInfoObjectID, setRepInfoDataObject

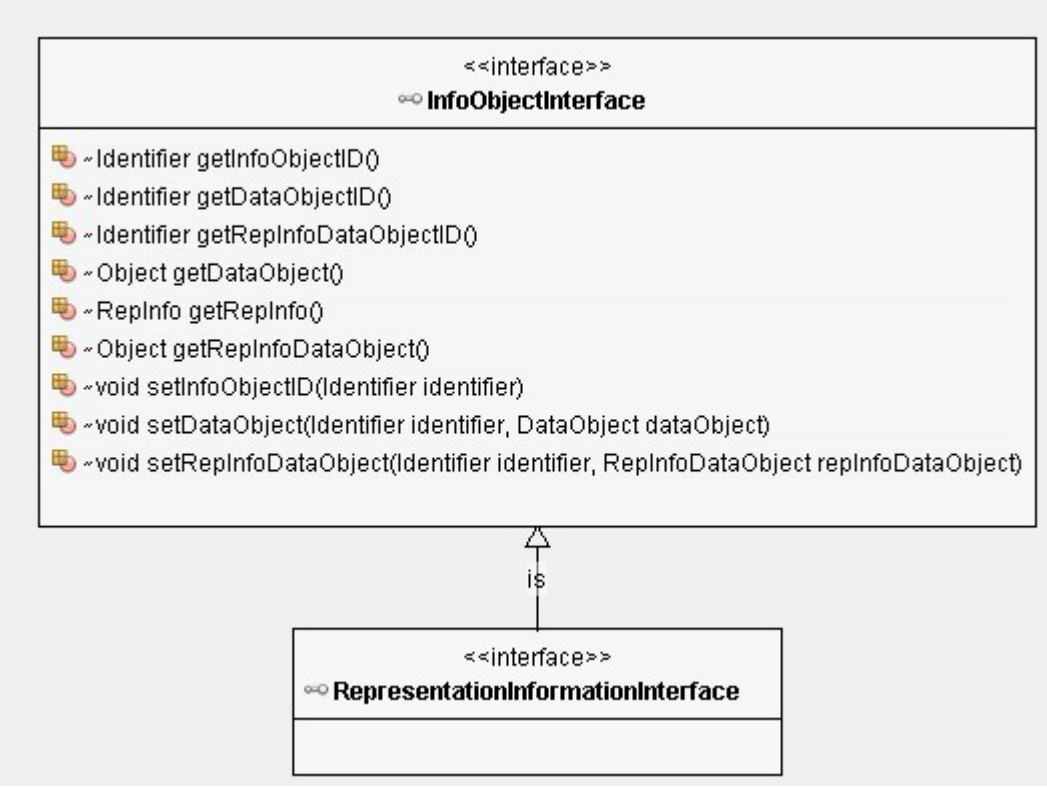

Figure 23 - Representation Information Interface

3.2.4 SPECIFIC ADAPTER LAYER

The Specific Adapter Layers contains the Specific Adapters and other classes necessary for requesting and sending an Information Objects to and from non-OAIS clients and archives. The Specific Adapter Layer is a component and is an element of the OAIS Interoperability Framework. This section is informative..

3.2.4.1 Specific Adapter

The Specific Adapter is an Adapter that extends the Generic Adapter and implements the Adapter and Information Object Interfaces. Specific Adapters also implement custom-made interfaces for clients and archives that are not OAIS compliant. The choice of Specific Adapters to use is negotiated external to Generic Adapter layer. This section is informative.

3.2.4.2 Producer Interface

The Producer Interface object class is a well-defined entry point for producer services. The Producer Interface class is a subclass of Component and is an element of the OAIS Interoperability Framework. This section is informative.

3.2.4.2.1 Ingest

Ingest Functional Entity (aka Ingest): The OAIS functional entity that contains the services and functions that accept Submission Information Packages from Producers, prepares Archival Information Packages for storage, and ensures that Archival Information Packages and their supporting Descriptive Information become established within the OAIS.

The Ingest object class provides the functions necessary to accept Submission Information Packages and register them through the Archive Interface. It implements the Message Service Interface that provides a protocol or interaction pattern for communication. It also implements the Access Interface that uses an Adapter to interoperate with the archive. The Adapter to be used is negotiated with the archive. The Ingest class is an element of the Consumer and Producer Interface components. This section is informative.

3.2.4.3 Consumer Interface

The Consumer Interface provides abstractions of consumer services.

The Consumer Interface object class is a well-defined entry point for consumer services. The Consumer Interface class is a subclass of Component and is an element of the OAIS Interoperability Framework. This section is informative.

3.2.4.3.1 Access

Access Functional Entity (aka Access): The OAIS functional entity that contains the services and functions which make the archival information holdings and related services visible to Consumers.

The Access object class provides the functions necessary to locate and retrieve Information Packages and Dissemination Information Packages through the Archive Interface. It implements the Message Service Interface that provides a protocol or interaction pattern for communication. It also implements the Access Interface that uses an Adapter to interoperate with the archive. The Adapter to be used is negotiated with the archive. The Ingest class is an element of the Consumer and Producer Interface components. This section is informative.

3.2.4.4 Negotiate

The Negotiate function allows a user and the OAIS to negotiate adapters that enable them to interoperate.

The Negotiate object class provides the functions necessary to identify and choose specialized Adapters that are mutually acceptable to interoperate between the client and the archive. The Negotiate class implements the Negotiate Interface which provides agreeStrategy and chooseAdapter methods. It also implements the Message Service Interface that provides a protocol or interaction pattern for communication. This section is informative.

3.2.4.4.1 Negotiate_Interface

The Negotiate Interface is a well-defined entry point for the Negotiate object class. The interface requires an agreeStrategy and a chooseAdapter method. It is an element of the OAIS Interoperability Framework component. This section is normative.

```
public interface Negotiate_Interface
```

The Negotiate Interface is used by clients and/or archives to identify a specific adapter with which they will be able to inter-operate.

3.2.4.4.1.1 Method Summary

All Methods	
Modifier and Type	**Method and Description**
`Boolean`	`agreeStrategy(InfoObject infoObject)` The agreeStrategy method provides the description of a candidate adapter for consideration as means for passing information objects.
`ArrayList <InfoObject>`	`chooseAdapter(ArrayList<InfoObject> infoObjectArr)` The chooseAdapter method enable a component to select a subset of adapters from a set of adapters

void	setAdapter(InfoObject infoObject) The setAdapter method makes a negotiated adapter active.

3.2.4.4.1.2 Method Detail

3.2.4.4.1.2.1 agreeStrategy

```
Boolean agreeStrategy(InfoObject infoObject)
```

The agreeStrategy method provides the description of a candidate adapter for consideration as means for passing information objects.

Parameters:

`infoObject` - an Information Object that describing a candidate adapter

Returns:

Boolean True indicates that the candidate adapter is a valid candidate.

3.2.4.4.1.2.2 chooseAdapter

```
ArrayList<InfoObject> chooseAdapter(ArrayList<InfoObject> infoObjectArr)
```

The chooseAdapter method enable a component to select a subset of adapters from a set of adapters

Parameters:

`infoObjectArr` - an array of Information Objects that describe existing adapters

Returns:

ArrayList an array of Information Objects that describe candidate adapters

3.2.4.4.1.2.3 setAdapter

```
void setAdapter(InfoObject infoObject)
```

The setAdapter method makes a negotiated adapter active.

Parameters:

`infoObject` - an Information Object that describes the adapter to use.

3.3 OAIS_IF_ARCHIVE

An OAIS IF Archive is an organization that intends to preserve information for access and use by a Designated Community and acknowledges the OAIS IF can be used to interoperate with other OAIS IF Archives. This section is informative.

3.3.1 OAIS_IF_ARCHIVE_INTERFACE

The OAIS IF Archive Interface is a well-defined entry point for the OAIS_IF_Archive and provides a contract for the exchange of information.

3.3.1.1 Local_Access

The Local Access class provides an access capability for an Archive.

The Local Access object class provides the functions to locate and retrieve information packages in the Archive. The Local Access class is an element of the Archive Interface component. A special Adapter must address any OAIS-IF requirement that is not met by a corresponding Local Access instance. This section is informative.

3.3.1.2 Local_Ingest

The Local Ingest class provides an ingest capability for an Archive.

The Local Ingest object class provides the functions to accept information packages and register them in the Archive. A special Adapter must address any OAIS-IF requirement that is not met by a corresponding Local Ingest instance. The Local Ingest class is an element of the Archive Interface component. This section is informative.

3.3.2 ARCHIVAL_STORAGE

Archival Storage Functional Entity (aka Archival Storage): The OAIS functional entity that contains the services and functions used for the storage and retrieval of Archival Information Packages.

The Archival Storage class is a subclass of Component. This section is informative.

4 FRAMEWORK IMPLEMENTATION

The implementation of the normative definitions of the generic adapter and information object interfaces should be done using design patterns and technologies that provide the broadest level of interoperability possible, namely the ability to send and receive Information Objects between two generic adapters.

The Abstract Generic Adapter shown in Figure 5 must implement the Message Interface to provide a standard protocol or interaction pattern to pass Information Objects between software components. It also implements the identifier, communication protocol, and serialization interfaces that are needed to identify and transfer Information Objects over a communication network. The interfaces for manipulating the Information Object, its components, and its extensions, are defined by the Information Object's interfaces.

In the following sections are method signatures extracted from a running prototype. All interfaces and their respective classes have been implemented. The source code will be made available through github.

4.1 INTERFACE SPECIFICATIONS - CORE

Interface specifications consist of a set of rules and conventions that define how different software components interact with each other. They act as a contract or boundary that specifies the methods, functions, or procedures that a software component exposes to the outside world, as well as the format and types of data that can be exchanged.

The following are the core interfaces of the OAIS Interoperability Framework.

4.1.1 INTERFACE ADAPTER INTERFACE

All Known Implementing Classes:

`GenericAdapter`, `OAISIFClient`, `SpecificAdapter`

---

public interface AdapterInterface

The Adapter Interface is a well-defined entry point for sending and getting Information Packages.

This interface provides the primary inter-operability interface.

4.1.1.1 Method Summary

```
Void asyncGetPackage(Identifier identifier, Message message)
```

The asyncGetPackage method enqueues a Request to the specified receiver for one or more Information Objects.

`Void asyncSendPackage(Identifier identifier, Message message)`

The asyncSendPackage method sends an InfoObject to the specified receiver.

`Identifier[]`

Package

4.1.2 INTERFACE INFO OBJECT INTERFACE

All Known Subinterfaces:

AccessRightsInformationInterface, ArchivalInformationPackageInterface, ContentInformationInterface, ContextInformationInterface, FixityInformationInterface, InformationPackageInterface, MessageInterface, PackagedInformationInterface, PreservationDescriptionInformationInterface, ProvenanceInformationInterface, ReferenceInformationInterface, RepresentationInformationInterface

All Known Implementing Classes:

AccessRightsInformation, ContentInformation, ContextInformation, FixityInformation, InfoObject, Message, PackagedInformation, PreservationDescriptionInformation, ProtocolDataUnit, ProvenanceInformation, ReferenceInformation, RepresentationInformation

---

public interface InfoObjectInterface

The InfoObjectInterface interface represents an information object in the OAIS (Open Archival Information System). It provides methods to retrieve and set various properties of the information object.

4.1.2.1 Method Details

4.1.2.1.1 getInfoObjectID

Identifier getInfoObjectID()

Retrieves the identifier of the info object.

Returns:The identifier of the info object.

4.1.2.1.2 getDataObjectID

Identifier getDataObjectID()

Retrieves the identifier of the associated data object.

Returns:The identifier of the data object.

4.1.2.1.3 getRepInfoDataObjectID

Identifier getRepInfoDataObjectID()

Retrieves the identifier of the associated representation information data object.

Returns:The identifier of the representation information data object.

4.1.2.1.4 getDataObject

Object getDataObject()

Retrieves the data object associated with the info object.

Returns:

The data object associated with the info object.

4.1.2.1.5 getRepInfoDataObject

Object getRepInfoDataObject()

Retrieves the representation information data object associated with the info object.

Returns:The representation information data object associated with the info object.

4.1.2.1.6 setInfoObjectID

void setInfoObjectID(Identifier identifier)

Sets the identifier of the info object.

Parameters: `identifier` - The identifier to set for the info object.

4.1.2.1.7 setDataObject

void setDataObject(Identifier identifier, DataObject dataObject)

Sets the data object associated with the info object.

Parameters: `identifier` - The identifier of the data object to set.

`dataObject` - The data object to associate with the info object.

4.1.2.1.8 setRepInfoDataObject

void setRepInfoDataObject(Identifier identifier, RepInfoDataObject repInfoDataObject)

Sets the representation information data object associated with the info object.

Parameters: `identifier` - The identifier of the representation information data object to set.

`repInfoDataObject` - The representation information data object to associate with the info object.

4.1.2.1.9 serialize

Object serialize()

Serializes the info object into an object.

Returns: The serialized representation of the info object.

4.1.2.1.10 deserialize

Object deserialize(String infoObjSer)

Deserializes the info object from a serialized string representation.

Parameters: `infoObjSer` - The serialized string representation of the info object.

Returns: The deserialized info object.

Package

4.1.3 INTERFACE DATA OBJECT INTERFACE

All Known Subinterfaces:

`ContentDataObjectInterface`, `RepInfoDataObjectInterface`

All Known Implementing Classes:

`DataObject`, `RepInfoDataObject`

---

public interface DataObjectInterface

The Data Object Interface is a well-defined entry point for getting and putting the Identifier and Object of a Data Object.

4.1.3.1 Method Details

4.1.3.1.1 getIdentifier

Identifier getIdentifier()

The getIdentifier method returns the Identifier of this DataObject.

Returns: Identifier the identifier for this DataObject

4.1.3.1.2 getObject

Object getObject()

The getObject method returns the Object for this Data Object

Returns: Object the object for this dataObject

4.1.3.1.3 setObject

void setObject(Identifier identifier, Object object)

The setObject method sets the object for this Data Object.

Parameters: `identifier` - the identifier for this dataObject

`object` - the object for this dataObject

Package

4.1.4 INTERFACE DIGITALOBJECT INTERFACE

All Known Implementing Classes:

`DigitalObject`

---

public interface DigitalObjectInterface

The Digital Object Interface is a well-defined entry point and contract for accessing the sequence of bits that comprise a digital object.

A Digital Object is an object composed of a set of bit sequences.

4.1.4.1 Method Details

4.1.4.1.1 getBits

Object getBits()

The getBits method returns the sequence of bits that comprise a digital object

Returns: BitSet a sequence of bits

Package

4.1.5 INTERFACE REPINFO DATA OBJECT INTERFACE

All Superinterfaces:

`DataObjectInterface`

All Known Implementing Classes:

`RepInfoDataObject`

---

public interface RepInfoDataObjectInterface extends DataObjectInterface

The Representation Information Data Object Interface is a well-defined entry point for getting and putting the Identifier and Object of an Information Object's Representation Information (a Data Object).

4.1.5.1 Methods inherited from interface **DataObjectInterface**

`getIdentifier`, `getObject`, `setObject`

Package

4.1.6 INTERFACE IDENTIFIER INTERFACE

All Known Implementing Classes:

`EndPointConnection`, `Identifier`

---

public interface IdentifierInterface

The Identifier Interface is a well-defined entry point and contract for an Identifier. An Identifier names an Object.

4.1.6.1 Method Details

4.1.6.1.1 getIdentifier

Object getIdentifier()

The getIdentifier method returns the identifier. .

Returns: Object the identifier of an Object.

4.1.6.1.2 setIdentifier

void setIdentifier(Object id)

The setIdentifier method stores an Identifier.

Parameters: `id` - the identifier of an Object.

4.2 INTERFACE SPECIFICATIONS – OAIS INFORMATION MODEL

Interface specifications consist of a set of rules and conventions that define how different software components interact with each other. They act as a contract or boundary that specifies the methods, functions, or procedures that a software component exposes to the outside world, as well as the format and types of data that can be exchanged.

The following are the interfaces for the OAIS Information Model Entities

Package

4.2.1 INTERFACE ACCESS RIGHTS INFORMATION INTERFACE

All Superinterfaces:

`InfoObjectInterface`

---

public interface AccessRightsInformationInterface extends InfoObjectInterface

The Access Rights Information Interface is a well-defined entry point and contract for getting and putting this Information Object and its components.

Access Rights Information is a subclass of Information Object and inherits methods for getting and putting the component Representation Information and Digital Object of this Information Object.

Access Rights Information: The information that identifies the access restrictions pertaining to the Content Information, including the legal framework, licensing terms, and access control. It

contains the access and distribution conditions stated within the Submission Agreement, related to both preservation (by the OAIS) and final usage (by the Consumer). It also includes the specifications for the application of rights enforcement measures.

4.2.1.1 Methods inherited from interface **InfoObjectInterface**

deserialize, getDataObject, getDataObjectID, getInfoObjectID, getRepInfoDataObject, getRepInfoDataObjectID, serialize, setDataObject, setInfoObjectID, setRepInfoDataObject

Package

4.2.2 INTERFACE CONTEXT INFORMATION INTERFACE

All Superinterfaces:

InfoObjectInterface

---

public interface ContextInformationInterface extends InfoObjectInterface

The Context Information Interface is a well-defined entry point and contract for getting and putting this Information Object and its components.

Context Information is a subclass of Information Object and inherits methods for getting and putting the component Representation Information and Digital Object of this Information Object.

Context Information is a subclass of Information Object and inherits Context Information: The information that documents the relationships of the Content Information to its environment. This includes why the Content Information was created and how it relates to other Content Information objects.

4.2.2.1 **Methods inher**ited from interface **InfoObjectInterface**

deserialize, getDataObject, getDataObjectID, getInfoObjectID, getRepInfoDataObject, getRepInfoDataObjectID, serialize, setDataObject, setInfoObjectID, setRepInfoDataObject

Package

4.2.3 INTERFACE FIXITY INFORMATION INTERFACE

All Superinterfaces:

InfoObjectInterface

---

public interface FixityInformationInterface extends InfoObjectInterface

The Fixity Information Interface is a well-defined entry point and contract for getting and putting this Information Object and its components.

Fixity Information is a subclass of Information Object and inherits methods for getting and putting the component Representation Information and Digital Object of this Information Object.

Fixity Information: The information which documents the mechanisms that ensure that the Content Information object has not been altered in an undocumented manner. An example is a Cyclical Redundancy Check (CRC) code for a file.

4.2.3.1 Methods inherited from interface **InfoObjectInterface**

deserialize, getDataObject, getDataObjectID, getInfoObjectID, getRepInfoDataObject, getRepInfoDataObjectID, serialize, setDataObject, setInfoObjectID, setRepInfoDataObject

Package

4.2.4 INTERFACE PROVENANCE INFORMATION INTERFACE

All Superinterfaces:

InfoObjectInterface

---

public interface ProvenanceInformationInterface extends InfoObjectInterface

The Provenance Information Interface is a well-defined entry point and contract for getting and putting this Information Object and its components.

Provenance Information is a subclass of Information Object and inherits methods for getting and putting the component Representation Information and Digital Object of this Information Object.

Provenance Information: The information that documents the history of the Content Information. This information tells the origin or source of the Content Information, any changes that may have taken place since it was originated, and who has had custody of it since it was originated. The Archive is responsible for creating and preserving Provenance Information from the point of Ingest; however, earlier Provenance Information should be provided by the Producer. Provenance Information adds to the evidence to support Authenticity.

4.2.4.1 Methods inherited from interface **InfoObjectInterface**

deserialize, getDataObject, getDataObjectID, getInfoObjectID, getRepInfoDataObject, getRepInfoDataObjectID, serialize, setDataObject, setInfoObjectID, setRepInfoDataObject

Package

4.2.5 INTERFACE REFERENCE INFORMATION INTERFACE

All Superinterfaces:

InfoObjectInterface

---

public interface ReferenceInformationInterface extends InfoObjectInterface

The Reference Information Interface is a well-defined entry point and contract for getting and putting this Information Object and its components.

Reference Information is a subclass of Information Object and inherits methods for getting and putting the component Representation Information and Digital Object of this Information Object.

Reference Information: The information that is used as an identifier for the Content Information. It also includes identifiers that allow outside systems to refer unambiguously to a particular Content Information. An example of Reference Information is an ISBN.

4.2.5.1 Methods inherited from interface **InfoObjectInterface**

deserialize, getDataObject, getDataObjectID, getInfoObjectID, getRepInfoDataObject, getRepInfoDataObjectID, serialize, setDataObject, setInfoObjectID, setRepInfoDataObject

Package

4.2.6 INTERFACE REPRESENTATION INFORMATION INTERFACE

All Superinterfaces:

InfoObjectInterface

---

public interface RepresentationInformationInterface extends InfoObjectInterface

The Representation Information Interface is a well-defined entry point and contract for getting and putting this Information Object and its components.

Representation Information is a subclass of Information Object and inherits methods for getting and putting the component Representation Information and Digital Object of this Information Object.

Representation Information: The information that maps a Data Object into more meaningful concepts. An example of Representation Information for a bit sequence which is a FITS file might consist of the FITS standard which defines the format plus a dictionary which defines the meaning in the file of keywords which are not part of the standard. Another example is JPEG software which is used to render a JPEG file; rendering the JPEG file as bits is not very meaningful to humans but the software, which embodies an understanding of the JPEG standard, maps the bits into pixels which can then be rendered as an image for human viewing.

4.2.6.1 Methods inherited from interface **InfoObjectInterface**

deserialize, getDataObject, getDataObjectID, getInfoObjectID, getRepInfoDataObject, getRepInfoDataObjectID, serialize, setDataObject, setInfoObjectID, setRepInfoDataObject

Package

4.2.7 INTERFACE INFORMATION PACKAGE INTERFACE

All Superinterfaces:

InfoObjectInterface

All Known Subinterfaces:

ArchivalInformationPackageInterface

---

public interface InformationPackageInterface extends InfoObjectInterface

The Information Package Interface is a well-defined entry point and contract for getting and putting this Information Object and its components.

Information Package is a subclass of Information Object and inherits methods for getting and putting the component Representation Information and Data Object of this Information Object.

Information Package: A logical container composed of optional Content Information and optional associated Preservation Description Information.

4.2.7.1 Methods inherited from interface **InfoObjectInterface**

deserialize, getDataObject, getDataObjectID, getInfoObjectID, getRepInfoDataObject, getRepInfoDataObjectID, serialize, setDataObject, setInfoObjectID, setRepInfoDataObject

Package

4.2.8 INTERFACE PRESERVATION DESCRIPTION INFORMATION INTERFACE

All Superinterfaces: `InfoObjectInterface`

---

public interface PreservationDescriptionInformationInterface extends InfoObjectInterface

The Preservation Description Information Interface is a well-defined entry point and contract for getting and putting this Information Object and its components.

Preservation Description Information is a subclass of Information Object and inherits methods for getting and putting the component Representation Information and Data Object of this Information Object. The Data Object is a container for Provenance, Reference, Fixity, Context, and Access Rights Information.

Preservation Description Information (PDI): The information which is necessary for adequate preservation of the Content Information and which can be categorized as Provenance, Reference, Fixity, Context, and Access Rights Information.

4.2.8.1 Method Details

4.2.8.1.1 getProvenanceInformation

Object getProvenanceInformation()

The getProvenanceInformation method returns the Provenance Information component of this Preservation Description Information Object.

Returns:

Object the Provenance Information for this InfoObject

4.2.8.1.2 getReferenceInformation

Object getReferenceInformation()

The getReferenceInformation method returns the Reference Information component of this Preservation Description Information Object.

Returns: Object the Reference Information for this InfoObject

4.2.8.1.3 getFixityInformation

Object getFixityInformation()

The getFixityInformation method returns the Fixity Information component of this Preservation Description Information Object.

Returns: Object the Fixity Information for this InfoObject

4.2.8.1.4 getContextInformation

Object getContextInformation()

The getContextInformation method returns the Context Information component of this Preservation Description Information Object.

Returns: Object the Context Information for this InfoObject

4.2.8.1.5 getAccessRightsInformation

Object getAccessRightsInformation()

The getAccessRightsInformation method returns the Access Rights Information component of this Preservation Description Information Object.

Returns: Object the Access Rights Information for this InfoObject

4.2.8.1.6 setProvenanceInformation

void setProvenanceInformation(Identifier identifier, InfoObject infoObject)

The setProvenanceInformation method sets the Provenance Information component of this Preservation Description Information Object.

Parameters: `identifier` - the identifier for this infoObject

`infoObject` - the Provenance Information component for this InfoObject

4.2.8.1.7 setReferenceInformation

void setReferenceInformation(Identifier identifier, InfoObject infoObject)

The setReferenceInformation method sets the Reference Information component of this Preservation Description Information Object.

Parameters: `identifier` - the identifier for this infoObject

`infoObject` - the Reference Information component for this InfoObject

4.2.8.1.8 setFixityInformation

void setFixityInformation(Identifier identifier, InfoObject infoObject)

The setFixityInformation method sets the Fixity Information component of this Preservation Description Information Object.

Parameters: `identifier` - the identifier for this infoObject

`infoObject` - the Fixity Information component for this InfoObject

4.2.8.1.9 setContextInformation

void setContextInformation(Identifier identifier, InfoObject infoObject)

The setContextInformation method sets the Context Information component of this Preservation Description Information Object.

Parameters: `identifier` - the identifier for this infoObject

`infoObject` - the Context Information component for this InfoObject

4.2.8.1.10 setAccessRightsInformation

void setAccessRightsInformation(Identifier identifier, InfoObject infoObject)

The setAccessRightsInformation method sets the Access Rights Information component of this Preservation Description Information Object.

Parameters: `identifier` - the identifier for this infoObject

`infoObject` - the Access Rights Information component for this InfoObject

Package

4.2.9 INTERFACE ARCHIVAL INFORMATION PACKAGE INTERFACE

All Superinterfaces: `InfoObjectInterface`, `InformationPackageInterface`

---

public interface ArchivalInformationPackageInterface extends InformationPackageInterface

The Archival Information Package Interface is a well-defined entry point and contract for getting and putting this Information Object and its components.

Archival Information Package is a subclass of Information Object and inherits methods for getting and putting the component Representation Information and Data Object of this Information Object. The Data Object is a container for the Content Information and Preservation Description Information.

Archival Information Package (AIP): An Information Package, consisting of the Content Information and the associated Preservation Description Information (PDI), which is preserved within an OAIS.

4.2.9.1 Method Details

4.2.9.1.1 getContentInformation

Object getContentInformation()

The getContentInformation method returns the Content Information component of this Archival Information Package.

Returns: Object the Content Information for this InfoObject

4.2.9.1.2 getPreservationDescriptionInformation

Object getPreservationDescriptionInformation()

The getPreservationDescriptionInformation method returns the Preservation Description Information component of this Archival Information Package.

Returns: Object the Preservation Description Information for this InfoObject

4.2.9.1.3 setContentInformation

void setContentInformation(Identifier identifier, InfoObject infoObject)

The setContentInformation method sets the Content Information component of this Archival Information Package.

Parameters: `identifier` - the identifier for this infoObject

`infoObject` - the Content Information component for this InfoObject

4.2.9.1.4 setPreservationDescriptionInformation

void setPreservationDescriptionInformation(Identifier identifier, InfoObject in foObject)

The setPreservationDescriptionInformation method sets the Preservation Description Information component of this Archival Information Package.

Parameters: `identifier` - the identifier for this infoObject

`infoObject` - the Preservation Description Information component for this InfoObject

Package

4.2.10 INTERFACE CONTENT DATA OBJECT INTERFACE

All Superinterfaces: `DataObjectInterface`

---

public interface ContentDataObjectInterface extends DataObjectInterface

The Content Data Object Interface is a well-defined entry point for getting and putting the Identifier and Object of a Content Data Object.

Content Data Object is a subclass of Data Object and inherits methods for getting and putting the Object and Identifier of this Content Data Object.

Content Data Object: The Data Object, that together with associated Representation Information, comprises the Content Information.

4.2.10.1 Methods inherited from interface **DataObjectInterface**

`getIdentifier`, `getObject`, `setObject`

Package

4.2.11 INTERFACE CONTENT INFORMATION INTERFACE

All Superinterfaces: `InfoObjectInterface`

---

public interface ContentInformationInterface extends InfoObjectInterface

The Content Information Interface is a well-defined entry point and contract for getting and putting this Information Object and its components.

Content Information is a subclass of Information Object and inherits methods for getting and putting the component Representation Information and Digital Object of this Information Object.

Content Information: A set of information that is the original target of preservation or that includes part or all of that information. It is an Information Object composed of its Content Data Object and its Representation Information.

4.2.11.1 Method Details

4.2.11.1.1 getContentDataObject

Object getContentDataObject()

The getContentDataObject method returns the Content Data Object for this Content Information.

Returns: Object the contentDataObject for this contentInformation

4.2.11.1.2 setContentDataObject

void setContentDataObject(Identifier identifier, Object object)

The setContentDataObject method sets the Content Data Object for this Content Information.

Parameters: `identifier` - the identifier for this contentDataObject

`object` - the contentDataObject for this contentInformation

Package

4.2.12 INTERFACE PACKAGED INFORMATION INTERFACE

All Superinterfaces:

`InfoObjectInterface`

---

public interface PackagedInformationInterface extends InfoObjectInterface

The Packaged Information Interface is a well-defined entry point and contract for getting and putting this Information Object and its components.

Packaged Information is a subclass of Information Object and inherits methods for getting and putting the component Representation Information and Digital Object of this Information Object.

4.2.12.1 Methods inherited from interface **InfoObjectInterface**

`deserialize`, `getDataObject`, `getDataObjectID`, `getInfoObjectID`, `getRepInfoDataObject`, `getRepInfoDataObjectID`, `serialize`, `setDataObject`, `setInfoObjectID`, `setRepInfoDataObject`

4.3 INTERFACE SPECIFICATIONS – OAISIF SERVICES

Interface specifications consist of a set of rules and conventions that define how different software components interact with each other. They act as a contract or boundary that specifies the methods, functions, or procedures that a software component exposes to the outside world, as well as the format and types of data that can be exchanged.

The following interfaces specify the OAIS Interoperability services.

Package

4.3.1 INTERFACE ARCHIVALSTORAGE INTERFACE

All Known Implementing Classes: `ArchivalStorage`

---

public interface ArchivalStorageInterface

4.3.1.1 Method Details

4.3.1.1.1 getInfoObjectQueryRequest

InfoObject getInfoObjectQueryRequest(InfoObject infoObjectQueryString)

The getInfoObjectQueryRequest method queries the Archival Storage for one or more Information Objects using the provided query string.

Parameters: `infoObjectQueryString` - a query string that identifies one or more Information Objects.

Returns: InfoObject an Information Object.

4.3.1.1.2 getDataObject

DataObject getDataObject(Identifier identifier)

The getDataObject method get a Data Object from Archival Storage using an identifier.

Parameters: `identifier` - the identifier of the receiver

Returns: DataObject the Data Object component of an Information Object

4.3.1.1.3 getRepInfo

RepInfoDataObject getRepInfo(Identifier identifier)

The getRepInfo method get the Representation Information component of an Information Object from Archival Storage using an identifier.

Parameters: `identifier` - the identifier of the Representation Information

Returns: RepInfoDataObject the Representation Information component

4.3.1.1.4 setInfoObject

void setInfoObject(Identifier identifier, InfoObject infoObject)

The setInfoObject method set an Information Object in Archival Storage using an identifier.

Parameters: `identifier` - the identifier of the receiver

`infoObject` - the Information Object to be stored.

4.3.1.1.5 setDataObject

void setDataObject(Identifier identifier, DataObject dataObject)

The setDataObject method set a Data Object in Archival Storage using an identifier.

Parameters: `identifier` - the identifier of the Data Object

`dataObject` - the Data Object to be stored

4.3.1.1.6 setRepInfo

void setRepInfo(Identifier identifier, RepInfoDataObject repInfo)

The setRepInfo method set a Representation Information component of an Information Object to Archival Storage using an identifier. .

Parameters: `identifier` - the identifier of the Representation Information

`repInfo` - The Representation Information to be stored

4.3.1.1.7 getObjectStore

ObjectStore getObjectStore()

The getObjectStore method gets this client's Object Store

Returns: ObjectStore The Object Store for this client.

4.3.1.1.8 putObjectStore

void putObjectStore(ObjectStore objectStore)

The putObjectStore method put this client's Object Store

Parameters: `objectStore` - the object store for this client

Package

4.3.2 INTERFACE MESSAGE INTERFACE

All Superinterfaces: `InfoObjectInterface`

All Known Implementing Classes: `Message`, `ProtocolDataUnit`

---

public interface MessageInterface extends InfoObjectInterface

The Message Interface is a well-defined entry point for getting and putting the components of a Message.

The Message class is defined as an extension of the Info Object class.

The methods defined are for the local implementation the Message class.

4.3.2.1 Method Details

4.3.2.1.1 getIdentifier

Identifier getIdentifier()

The getIdentifier method returns the identifier of this message.

Returns: Identifier the identifier of this message

4.3.2.1.2 getSenderId

Identifier getSenderId()

The getSenderId method returns the identifier of the sender

Returns: Identifier the identifier of the sender

4.3.2.1.3 getReceiverId

Identifier getReceiverId()

The getReceiverId method returns the identifier of the receiver

Returns: Identifier the identifier of the receiver

4.3.2.1.4 getSenderIO

Object getSenderIO()

The getSenderIO method returns the object from the sender.

Returns: Object the Info Object from the sender.

4.3.2.1.5 getReceiverIO

Object getReceiverIO()

The getReceiverIO method returns the object from the receiver. receiver.

Returns: Object the Info Object from the receiver.

4.3.2.1.6 setIdentifier

void setIdentifier(Identifier identifier)

The setIdentifier method sets the identifier of this message

Parameters: `identifier` - the identifier of the message

4.3.2.1.7 setSenderId

void setSenderId(Identifier senderId)

The setSenderId method sets the identifier of the sender receiver.

Parameters: `senderId` - the identifier of the sender

4.3.2.1.8 setReceiverId

void setReceiverId(Identifier receiverId)

The setReceiverId method sets the identifier of the receiver

Parameters: `receiverId` - the identifier of the receiver

4.3.2.1.9 setSenderIO

void setSenderIO(Object senderObject)

The setSenderIO method sets the object from the sender

Parameters: `senderObject` - the object being sent from the sender to the receiver

4.3.2.1.10 setReceiverIO

void setReceiverIO(Object receiverObject)

The setReceiverIO method sets the object from the receiver receiver.

Parameters: `receiverObject` - the object being sent from the receiver to the sender

Package

4.3.3 INTERFACE OBJECTSTORE INTERFACE

All Known Implementing Classes: `ObjectStore`

---

public interface ObjectStoreInterface

The ObjectStoreInterface interface represents an object store in the OAIS (Open Archival Information System). It provides methods to retrieve and store various types of objects, such as InfoObjects, DataObjects, and RepInfoDataObjects.

4.3.3.1 Method Details

4.3.3.1.1 getInfoObject

InfoObject getInfoObject(Identifier identifier)

Retrieves the InfoObject associated with the specified identifier.

Parameters: `identifier` - The identifier of the InfoObject to retrieve.

Returns:

The InfoObject associated with the identifier, or null if not found.

4.3.3.1.2 getDataObject

DataObject getDataObject(Identifier identifier)

Retrieves the DataObject associated with the specified identifier.

Parameters: `identifier` - The identifier of the DataObject to retrieve.

Returns: The DataObject associated with the identifier, or null if not found.

4.3.3.1.3 getRepInfo

RepInfoDataObject getRepInfo(Identifier identifier)

Retrieves the RepInfoDataObject associated with the specified identifier.

Parameters: `identifier` - The identifier of the RepInfoDataObject to retrieve.

Returns: The RepInfoDataObject associated with the identifier, or null if not found.

4.3.3.1.4 setInfoObject

void setInfoObject(Identifier identifier, InfoObject infoObject)

Stores the specified InfoObject with the given identifier.

Parameters: `identifier` - The identifier to associate with the InfoObject.

`infoObject` - The InfoObject to store.

4.3.3.1.5 setDataObject

void setDataObject(Identifier identifier, DataObject dataObject)

Stores the specified DataObject with the given identifier.

Parameters: `identifier` - The identifier to associate with the DataObject.

`dataObject` - The DataObject to store.

4.3.3.1.6 setRepInfo

void setRepInfo(Identifier identifier, RepInfoDataObject repInfo)

Stores the specified RepInfoDataObject with the given identifier.

Parameters: `identifier` - The identifier to associate with the RepInfoDataObject.

`repInfo` - The RepInfoDataObject to store.

Package

4.3.4 INTERFACE REQUEST INTERFACE

All Known Implementing Classes: `Request`

---

public interface RequestInterface

The Request Interface is a well-defined entry point for message enqueue and dequeue.

4.3.4.1 Method Details

4.3.4.1.1 requestEnqueue

void requestEnqueue(Identifier messageID, Message message)

The Request Enqueue method enqueues the provided message

Parameters: `messageID` - the identifier of the Message to enqueue

`message` - the Message to enqueue

4.3.4.1.2 requestDequeue

boolean requestDequeue(Identifier messageID)

The requestDequeue method dequeues the provided message

Parameters: `messageID` - the identifier of the Message to dequeue

Returns: boolean true if the dequeue was successful

4.3.4.1.3 returnRequest

OAISIFClient returnRequest(Message message)

The requestDequeue method dequeues the provided message

Parameters: `message` - the original Message containing the query, an InfoObject

Returns: OAISIFClient the client receiving the message

4.4 INTERFACE SPECIFICATIONS - SWITCHBOARD

Interface specifications consist of a set of rules and conventions that define how different software components interact with each other. They act as a contract or boundary that specifies the methods, functions, or procedures that a software component exposes to the outside world, as well as the format and types of data that can be exchanged.

The following specify the interfaces for the OAIS Interoperability Framework's switchboard.

Package

4.4.1 INTERFACE SWITCHBOARD INTERFACE

All Known Implementing Classes:

`SwitchBoardState`

---

public interface SwitchBoardInterface

The SwitchBoardInterface is a well-defined entry point for accessing the Switch Board.

4.4.1.1 Method Details

4.4.1.1.1 getSwitchBoardEntry

SwitchBoardEntry getSwitchBoardEntry(String identifier)

Retrieves a SwitchBoardEntry from the Switch Board based on the given identifier.

Parameters: `identifier` - The identifier of the Switch Board entry.

Returns: The SwitchBoardEntry associated with the identifier.

4.4.1.1.2 getActiveClientById

OAISIFClient getActiveClientById(String identifier)

Retrieves an active client from the Switch Board based on the given identifier.

Parameters: `identifier` - The identifier of the active client.

Returns: The active OAISIFClient associated with the identifier.

4.4.1.1.3 getActiveClientIdArr

ArrayList<String> getActiveClientIdArr()

Retrieves an array list of identifiers for active clients in the Switch Board.

Returns: An ArrayList containing the identifiers of active clients.

4.4.1.1.4 getCandidateClientIdArr

ArrayList<String> getCandidateClientIdArr()

Retrieves an array list of identifiers for candidate clients in the Switch Board.

Returns: An ArrayList containing the identifiers of candidate clients.

4.4.1.1.5 getSwitchBoardStateAsString

String getSwitchBoardStateAsString()

Retrieves the state of the Switch Board as a string representation.

Returns: The state of the Switch Board as a string.

4.4.1.1.6 setSwitchBoardState

void setSwitchBoardState(OAISIFClient lOAISIFClient)

Sets the state of the Switch Board using the given OAISIFClient.

Parameters: `lOAISIFClient` - The OAISIFClient to set as the Switch Board state.

4.5 CLASS SPECIFICATIONS - CORE

Class specifications are blueprints or templates that define the structure and behavior of objects. They serve as blueprints for creating instances of objects, which are individual occurrences based on the class specification. These class specifications have been abbreviated to include only the class name, definition, and methods/constructors.

The following class specifications are for the core classes of the OAIS Interoperability Framework.

Package

4.5.1 CLASS GENERIC ADAPTER

GenericAdapter

All Implemented Interfaces: `AdapterInterface`

Direct Known Subclasses: `SpecificAdapter`

---

public class GenericAdapter extends Object implements AdapterInterface

The GenericAdapter class represents an abstract client implementation that serves as a base for specific client implementations. It implements the AdapterInterface and provides common functionality and properties for clients.

4.5.1.1 Method Details

4.5.1.1.1 asyncGetPackage

public void asyncGetPackage(Identifier receiverId, Message message)

Description copied from interface: `AdapterInterface`

The asyncGetPackage method enqueues a Request to the specified receiver for one or more Information Objects. The method sends the query as an InfoObject by including it as the RepInfo of a Message and using the Request

Asynchronous interaction pattern. The results are expected as the Data Object of the Message.

Specified by: `asyncGetPackage` in interface `AdapterInterface`

Parameters: `receiverId` - the identifier of the receiver

`message` - the request as the Data Object of a Message

4.5.1.1.2 asyncSendPackage

public void asyncSendPackage(Identifier receiverId, Message message)

Description copied from interface: `AdapterInterface`

The asyncSendPackage method sends an InfoObject to the specified receiver. The method sends the InfoObject as the Data Object of a Message using the Request Asynchronous interaction pattern. The results are expected in the RepInfo of the Message.

Specified by: `asyncSendPackage` in interface `AdapterInterface`

Parameters: `receiverId` - the identifier of the receiver

`message` - a message containing the Info Object being sent.

4.5.1.1.3 syncSendPackage

public void syncSendPackage(Identifier receiverId, Message message)

Description copied from interface: `AdapterInterface`

The syncSendPackage method sends an InfoObject to the specified receiver. The method sends the InfoObject as the Data Object of a Message using the Request Synchronous interaction pattern. The results are expected in the RepInfo of the Message.

Specified by: `syncSendPackage` in interface `AdapterInterface`

Parameters: `receiverId` - the identifier of the receiver

`message` - a message containing the Info Object being sent.

4.5.1.1.4 getEndPoints

public Identifier[] getEndPoints(String source)

The getEndPoints method returns a set of End Point Connection as a list of Identifiers. sends an InfoObject to the specified receiver. The method sends the InfoObject as the Data Object of a Message using the Request Synchronous interaction pattern. The results are expected in the RepInfo of the Message.

Specified by: `getEndPoints` in interface `AdapterInterface`

Parameters: `source` - the identifier of the source

Returns: Identifier [] a list of End Point Connections.

4.5.1.1.5 getIdentifier

public Identifier getIdentifier()

Retrieves the identifier of the client.

Returns: The identifier of the client.

4.5.1.1.6 getArchivalStorage

public ArchivalStorage getArchivalStorage()

Retrieves the Archival Storage for the client.

Returns: Archival Storage for the client.

4.5.1.1.7 getSwitchBoardEntry

public SwitchBoardEntry getSwitchBoardEntry()

Retrieves the Switch Board Entry of the client.

Returns: The Switch Board Entry of the client.

4.5.1.1.8 getTitle

public String getTitle()

Retrieves the title of the client.

Returns: The title of the client.

4.5.1.1.9 getClientType

public String getClientType()

Retrieves the client type of the client.

Returns: The client type of the client.

4.5.1.1.10 getClientTypeInd

public String getClientTypeInd()

Retrieves the client type indicator of the client.

Returns: The client type indicator of the client.

4.5.1.1.11 getClientPortId

public String getClientPortId()

Retrieves the port ID of the client.

Returns: The port ID of the client.

Package

4.5.2 CLASS INFO OBJECT

InfoObject

All Implemented Interfaces: `InfoObjectInterface`

Direct Known Subclasses:

`AccessRightsInformation`, `ContentInformation`, `ContextInformation`, `FixityInformation`, `Message`, `PackagedInformation`, `PreservationDescriptionInformation`, `ProvenanceInformation`, `ReferenceInformation`, `RepresentationInformation`

---

public class InfoObject extends Object implements InfoObjectInterface

The InfoObject class represents an information object.

4.5.2.1 Field Details

4.5.2.1.1 infoObjectID

protected Identifier infoObjectID

4.5.2.1.2 dataObjectID

protected Identifier dataObjectID

4.5.2.1.3 repInfoDataObjectID

protected Identifier repInfoDataObjectID

4.5.2.1.4 objectStore

protected ObjectStore objectStore

4.5.2.1.5 Constructor Details

4.5.2.1.5.1 InfoObject

public InfoObject(Identifier infoObjectID, Identifier dataObjectID, Identifier r epInfoID, ObjectStore objectStore)

Constructs an InfoObject object with the provided identifiers and object store.

Parameters: `infoObjectID` - the identifier of the info object

`dataObjectID` - the identifier of the data object

`repInfoID` - the identifier of the rep info data object

`objectStore` - the object store

4.5.2.1.5.2 InfoObject

public InfoObject(ObjectStore objectStore)

Constructs an InfoObject object with the provided object store.

Parameters: `objectStore` - the object store

4.5.2.1.5.3 InfoObject

public InfoObject()

Constructs an empty InfoObject object.

4.5.2.2 Method Details

4.5.2.2.1 getInfoObjectID

public Identifier getInfoObjectID()

Description copied from interface: `InfoObjectInterface`

Retrieves the identifier of the info object.

Specified by: `getInfoObjectID` in interface `InfoObjectInterface`

Returns: The identifier of the info object.

4.5.2.2.2 getDataObjectID

public Identifier getDataObjectID()

Description copied from interface: `InfoObjectInterface`

Retrieves the identifier of the associated data object.

Specified by: `getDataObjectID` in interface `InfoObjectInterface`

Returns: The identifier of the data object.

4.5.2.2.3 getRepInfoDataObjectID

public Identifier getRepInfoDataObjectID()

Description copied from interface: `InfoObjectInterface`

Retrieves the identifier of the associated representation information data object.

Specified by: `getRepInfoDataObjectID` in interface `InfoObjectInterface`

Returns: The identifier of the representation information data object.

4.5.2.2.4 getDataObject

public Object getDataObject()

Description copied from interface: `InfoObjectInterface`

Retrieves the data object associated with the info object.

Specified by: `getDataObject` in interface `InfoObjectInterface`

Returns: The data object associated with the info object.

4.5.2.2.5 getRepInfoDataObject

public Object getRepInfoDataObject()

Description copied from interface: `InfoObjectInterface`

Retrieves the representation information data object associated with the info object.

Specified by: `getRepInfoDataObject` in interface `InfoObjectInterface`

Returns: The representation information data object associated with the info object.

4.5.2.2.6 getDataObjectStr

public String getDataObjectStr()

Retrieves the string representation of the data object.

Returns: the string representation of the data object

4.5.2.2.7 getRepInfoDataObjectStr

public String getRepInfoDataObjectStr()

Retrieves the string representation of the rep info data object.

Returns: the string representation of the rep info data object

4.5.2.2.8 getRepInfoDataObjectStr2

public String getRepInfoDataObjectStr2()

Retrieves the string representation of the rep info data object.

Returns: the string representation of the rep info data object

4.5.2.2.9 setInfoObjectID

public void setInfoObjectID(Identifier identifier)

Description copied from interface: `InfoObjectInterface`

Sets the identifier of the info object.

Specified by: `setInfoObjectID` in interface `InfoObjectInterface`

Parameters: `identifier` - The identifier to set for the info object.

4.5.2.2.10 setDataObject

public void setDataObject(Identifier identifier, DataObject dataObject)

Description copied from interface: `InfoObjectInterface`

Sets the data object associated with the info object.

Specified by: `setDataObject` in interface `InfoObjectInterface`

Parameters: `identifier` - The identifier of the data object to set.

`dataObject` - The data object to associate with the info object.

4.5.2.2.11 setRepInfoDataObject

public void setRepInfoDataObject(Identifier identifier, RepInfoDataObject repInfoDataObject)

Description copied from interface: `InfoObjectInterface`

Sets the representation information data object associated with the info object.

Specified by: `setRepInfoDataObject` in interface `InfoObjectInterface`

Parameters: `identifier` - The identifier of the representation information data object to set.

`repInfoDataObject` - The representation information data object to associate with the info object.

4.5.2.2.12 serialize

public Object serialize()

Description copied from interface: `InfoObjectInterface`

Serializes the info object into an object.

Specified by: `serialize` in interface `InfoObjectInterface`

Returns: The serialized representation of the info object.

4.5.2.2.13 deserialize

public Object deserialize(String InfoObjSer)

Description copied from interface: `InfoObjectInterface`

Deserializes the info object from a serialized string representation.

Specified by: `deserialize` in interface `InfoObjectInterface`

Parameters: `InfoObjSer` - The serialized string representation of the info object.

Returns: The deserialized info object.

4.5.2.2.14 setObjectStore

public void setObjectStore(ObjectStore objectStore)

Sets the object store for the InfoObject.

Parameters: `objectStore` - the object store to be set

4.5.2.2.15 dump

public void dump()

Dumps the contents of the InfoObject to the console.

Package

4.5.3 CLASS SPECIFIC ADAPTER

SpecificAdapter

All Implemented Interfaces:

`AdapterInterface`

Direct Known Subclasses:

`OAISIFClient`

---

public class SpecificAdapter extends GenericAdapter

The SpecificAdapter class represents a specific adapter implementation that extends the GenericAdapter.

4.5.3.1.1 Fields inherited from class GenericAdapter

`archivalStorage`, `oaisRequest`, `switchBoardEntry`

4.5.3.2 Method Details

4.5.3.2.1 getNextIdentifierRawStr

public String getNextIdentifierRawStr()

Retrieves the next raw identifier string.

Returns: The next identifier string.

4.5.3.2.2 getIdentifierSpecific

public static Identifier getIdentifierSpecific(String type, String identifierRaw)

Creates a specific identifier.

Parameters: `type` - The identifier type.

`identifierRaw` - The raw identifier value.

Returns: The created Identifier object.

4.5.3.2.3 getObjectStore

public ObjectStore getObjectStore()

Retrieves the object store associated with the SpecificAdapter.

Returns: The ObjectStore instance.

4.5.3.2.4 putInfoObjectGeneric

public void putInfoObjectGeneric(String lIdentifierRawStr, String clientType, Object lDOContent, Object lRIContent, String objectTypeId)

Puts an information object in the archival storage using a generic method.

Parameters: `lIdentifierRawStr` - The raw identifier string.

`clientType` - The client type.

`lDOContent` - The content of the data object.

`lRIContent` - The content of the representation information.

`objectTypeId` - The object type ID.

Package

4.5.4 CLASS IDENTIFIER

Identifier

All Implemented Interfaces:

`IdentifierInterface`

Direct Known Subclasses:

`EndPointConnection`

---

public class Identifier extends Object implements IdentifierInterface

An identifier which names an instance. (Identifier)

4.5.4.1.1 Constructor Summary

Description

`Identifier(Object identifier)`

Constructs an Identifier object with the provided identifier.

4.5.4.2 Method Details

4.5.4.2.1 getIdentifier

public Object getIdentifier()

Description copied from interface: `IdentifierInterface`

The getIdentifier method returns the identifier. .

Specified by: `getIdentifier` in interface `IdentifierInterface`

Returns: Object the identifier of an Object.

4.5.4.2.2 setIdentifier

public void setIdentifier(Object identifier)

Description copied from interface: `IdentifierInterface`

The setIdentifier method stores an Identifier.

Specified by: `setIdentifier` in interface `IdentifierInterface`

Parameters: `identifier` - the identifier of an Object.

4.5.4.2.3 getIdentifierStr

public String getIdentifierStr()

Retrieves the string representation of the identifier.

Returns: the string representation of the identifier

Package

4.5.5 CLASS DATA OBJECT

DataObject

All Implemented Interfaces:

`DataObjectInterface`

---

public class DataObject extends Object implements DataObjectInterface

The DataObject class represents either a Physical Object or a Data Object.

4.5.5.1 Method Details

4.5.5.1.1 getIdentifier

public Identifier getIdentifier()

Description copied from interface: `DataObjectInterface`

The getIdentifier method returns the Identifier of this DataObject.

Specified by: `getIdentifier` in interface `DataObjectInterface`

Returns: Identifier the identifier for this DataObject

4.5.5.1.2 getObject

public Object getObject()

Description copied from interface: `DataObjectInterface`

The getObject method returns the Object for this Data Object

Specified by: `getObject` in interface `DataObjectInterface`

Returns: Object the object for this dataObject

4.5.5.1.3 getObjectStr

public String getObjectStr()

Gets the string representation of the object associated with the data object.

Returns: the string representation of the object

4.5.5.1.4 setObject

public void setObject(Identifier identifier, Object object)

Description copied from interface: `DataObjectInterface`

The setObject method sets the object for this Data Object.

Specified by: `setObject` in interface `DataObjectInterface`

Parameters: `identifier` - the identifier for this dataObject

`object` - the object for this dataObject

4.5.5.1.5 dump

public void dump(String title)

Prints the details of the data object.

Parameters: `title` - the title of the dump

Package

4.5.6 CLASS DIGITAL OBJECT

DigitalObject

All Implemented Interfaces:

`DigitalObjectInterface`

---

public class DigitalObject extends Object implements DigitalObjectInterface

An object composed of a set of bit sequences.

4.5.6.1 Method Details

4.5.6.1.1 getBits

public Object getBits()

Retrieves the bit sequences of the digital object.

Specified by: `getBits` in interface `DigitalObjectInterface`

Returns: the bit sequences of the digital object

Package

4.5.7 CLASS REPINFO DATA OBJECT

java.lang.Object

RepInfoDataObject

All Implemented Interfaces:

`DataObjectInterface`, `RepInfoDataObjectInterface`

---

public class RepInfoDataObject extends Object implements RepInfoDataObjectInterface

Represents the relationship between representation information and a data object.

4.5.7.1 Method Details

4.5.7.1.1 getIdentifier

public Identifier getIdentifier()

Retrieves the identifier associated with the RepInfoDataObject.

Specified by: `getIdentifier` in interface `DataObjectInterface`

Returns: The identifier.

4.5.7.1.2 getObject

public Object getObject()

Retrieves the object associated with the RepInfoDataObject.

Specified by: `getObject` in interface `DataObjectInterface`

Returns: The object.

4.5.7.1.3 getObjectStr

public String getObjectStr()

Retrieves the object as a string.

Returns: The object as a string.

4.5.7.1.4 setObject

public void setObject(Identifier identifier, Object object)

Sets the identifier and object for the RepInfoDataObject.

Specified by: `setObject` in interface `DataObjectInterface`

Parameters: `identifier` - The identifier to set.

`object` - The object to set.

4.5.7.1.5 dump

public void dump(String title)

Prints the RepInfoDataObject information to the console.

Parameters: `title` - The title of the dump.

4.6 CLASS SPECIFICATIONS – OAIS INFORMATION MODEL

Class specifications are blueprints or templates that define the structure and behavior of objects. They serve as blueprints for creating instances of objects, which are individual occurrences based on the class specification. These class specifications have been abbreviated to include only the class name, definition, and methods/constructors.

The following class specifications are for the entities defined in the OAIS Information model.

Package

4.6.1 CLASS ACCESS RIGHTS INFORMATION

InfoObject

AccessRightsInformation

All Implemented Interfaces: `InfoObjectInterface`

public class AccessRightsInformation extends InfoObject

AccessRightsInformation represents the information that identifies the access restrictions pertaining to the Content Information, including the legal framework, licensing terms, and access control.

4.6.1.1 Methods inherited from class InfoObject

deserialize, dump, getDataObject, getDataObjectID, getDataObjectStr, getInfoObjectID, getRepInfoDataObject, getRepInfoDataObjectID, getRepInfoDataObjectStr, getRepInfoDataObjectStr2, serialize, setDataObject, setInfoObjectID, setObjectStore, setRepInfoDataObject

4.6.1.2 Constructor Details

public AccessRightsInformation(Identifier infoObjectID, Identifier dataObjectID, Identifier repInfoID, ObjectStore objectStore)

Constructs a new instance of AccessRightsInformation.

Parameters: `infoObjectID` - the identifier of the information object

`dataObjectID` - the identifier of the associated data object

`repInfoID` - the identifier of the associated representation information

`objectStore` - the object store where the object will be stored

Package

4.6.2 CLASS CONTEXT INFORMATION

InfoObject

ContextInformation

All Implemented Interfaces:

`InfoObjectInterface`

public class ContextInformation extends InfoObject

The ContextInformation class represents the information that documents the relationships of the Content Information to its environment.

4.6.2.1 Methods inherited from class InfoObject

deserialize, dump, getDataObject, getDataObjectID, getDataObjectStr, getInfoObjectID, getRepInfoDataObject, getRepInfoDataObjectID, getRepInfoDataObjectStr, getRepInfoDataObjectStr2, serialize, setDataObject, setInfoObjectID, setObjectStore, setRepInfoDataObject

4.6.2.2 Constructor Details

public ContextInformation(Identifier infoObjectID, Identifier dataObjectID, Identifier repInfoID, ObjectStore objectStore)

Constructs a new instance of ContextInformation.

Parameters: `infoObjectID` - the identifier of the information object

`dataObjectID` - the identifier of the associated data object

`repInfoID` - the identifier of the associated representation information

`objectStore` - the object store where the object will be stored

Package

4.6.3 CLASS FIXITY INFORMATION

InfoObject

FixityInformation

All Implemented Interfaces:

`InfoObjectInterface`

---

public class FixityInformation extends InfoObject

The information which documents the mechanisms that ensure that the Content Information object has not been altered in an undocumented manner.

4.6.3.1 Methods inherited from class InfoObject

deserialize, dump, getDataObject, getDataObjectID, getDataObjectStr, getInfoObjectID, getRepInfoDataObject, getRepInfoDataObjectID, getRepInfoDataObjectStr, getRepInfoDataObjectStr2, serialize, setDataObject, setInfoObjectID, setObjectStore, setRepInfoDataObject

4.6.3.2 Constructor Details

public FixityInformation(Identifier infoObjectID, Identifier dataObjectID, Identifier repInfoID, ObjectStore objectStore)

Constructs a new FixityInformation with the specified identifiers and object store.

Parameters: `infoObjectID` - the identifier of the FixityInformation object

`dataObjectID` - the identifier of the associated Data Object

`repInfoID` - the identifier of the associated Representation Information Object

`objectStore` - the ObjectStore used for storage and retrieval

Package

4.6.4 CLASS PROVENANCE INFORMATION

InfoObject

ProvenanceInformation

All Implemented Interfaces:

`InfoObjectInterface`

---

public class ProvenanceInformation extends InfoObject

The information that documents the history of the Content Information. This information tells the origin or source of the Content Information, any changes that may have taken place since it was originated, and who has had custody of it since it was originated.

4.6.4.1 Methods inherited from class **InfoObject**

`deserialize`, `dump`, `getDataObject`, `getDataObjectID`, `getDataObjectStr`, `getInfoObjectID`, `getRepInfoDataObject`, `getRepInfoDataObjectID`, `getRepInfoDataObjectStr`, `getRepInfoDataObjectStr2`, `serialize`, `setDataObject`, `setInfoObjectID`, `setObjectStore`, `setRepInfoDataObject`

4.6.4.2 Constructor Details

public ProvenanceInformation(Identifier infoObjectID, Identifier dataObjectID, Identifier repInfoID, ObjectStore objectStore)

Constructs a new instance of the ProvenanceInformation class.

Parameters: `infoObjectID` - The identifier of the InfoObject.

`dataObjectID` - The identifier of the DataObject.

`repInfoID` - The identifier of the RepInfo.

`objectStore` - The object store used to store the ProvenanceInformation object.

Package

4.6.5 CLASS REFERENCE INFORMATION

InfoObject

ReferenceInformation

All Implemented Interfaces:

`InfoObjectInterface`

---

public class ReferenceInformation extends InfoObject

Represents the reference information used as an identifier for the Content Information.

4.6.5.1 Methods inherited from class InfoObject

`deserialize`, `dump`, `getDataObject`, `getDataObjectID`, `getDataObjectStr`, `getInfoObjectID`, `getRepInfoDataObject`, `getRepInfoDataObjectID`, `getRepInfoDataObjectStr`, `getRepInfoDataObjectStr2`, `serialize`, `setDataObject`, `setInfoObjectID`, `setObjectStore`, `setRepInfoDataObject`

4.6.5.2 Constructor Details

public ReferenceInformation(Identifier infoObjectID, Identifier dataObjectID, Identifier repInfoID, ObjectStore objectStore)

Constructs a ReferenceInformation object with the provided identifiers and object store.

Parameters: `infoObjectID` - The identifier for the info object.

`dataObjectID` - The identifier for the data object.

`repInfoID` - The identifier for the representation information.

`objectStore` - The object store to associate with the reference information.

Package

4.6.6 CLASS REPRESENTATION INFORMATION

InfoObject

RepresentationInformation

All Implemented Interfaces:

`InfoObjectInterface`

---

public class RepresentationInformation extends InfoObject

Representation Information: The information that maps a Data Object into more meaningful concepts.

4.6.6.1 Methods inherited from class InfoObject

`deserialize`, `dump`, `getDataObject`, `getDataObjectID`, `getDataObjectStr`, `getInfoObjectID`, `getRepInfoDataObject`, `getRepInfoDataObjectID`, `getRepInfoDataObjectStr`, `getRepInfoDataObjectStr2`, `serialize`, `setDataObject`, `setInfoObjectID`, `setObjectStore`, `setRepInfoDataObject`

4.6.6.2 Constructor Details

public RepresentationInformation(Identifier infoObjectID, Identifier dataObjectID, Identifier repInfoID, ObjectStore objectStore)

Constructs a new RepresentationInformation instance with the specified parameters.

Parameters: `infoObjectID` - The identifier of the information object.

`dataObjectID` - The identifier of the data object.

`repInfoID` - The identifier of the representation information.

`objectStore` - The object store associated with the representation information.

Package

4.6.7 CLASS PRESERVATION DESCRIPTION INFORMATION

InfoObject

PreservationDescriptionInformation

All Implemented Interfaces:

InfoObjectInterface

---

public class PreservationDescriptionInformation extends InfoObject

Preservation Description Information (PDI): The information which is necessary for adequate preservation of the Content Information and which can be categorized as Provenance, Reference, Fixity, Context, and Access Rights Information.

4.6.7.1 Methods inherited from class InfoObject

deserialize, dump, getDataObject, getDataObjectID, getDataObjectStr, getInfoObjectID, getRepInfoDataObject, getRepInfoDataObjectID, getRepInfoDataObjectStr, getRepInfoDataObjectStr2, serialize, setDataObject, setInfoObjectID, setObjectStore, setRepInfoDataObject

4.6.7.2 Constructor Details

public PreservationDescriptionInformation()

Constructs a new instance of the PreservationDescriptionInformation class.

Package

4.6.8 CLASS CONTENT INFORMATION

InfoObject

ContentInformation

All Implemented Interfaces:

InfoObjectInterface

---

public class ContentInformation extends InfoObject

The ContentInformation class represents the Content Information Interface, which is a well-defined entry point and contract for getting and putting this Information Object and its components.

Content Information is a subclass of Information Object and inherits methods for getting and putting the component Representation Information and Digital Object of this Information Object.

4.6.8.1 Methods inherited from class InfoObject

deserialize, dump, getDataObject, getDataObjectID, getDataObjectStr, getInfoObjectID, getRepInfoDataObject, getRepInfoDataObjectID, getRepInfoDataObjectStr, getRepInfoDataObjectStr2, serialize, setDataObject, setInfoObjectID, setObjectStore, setRepInfoDataObject

4.6.8.2 Constructor Details

public ContentInformation(Identifier infoObjectID, Identifier dataObjectID, Identifier repInfoID, ObjectStore objectStore)

Constructs a new instance of ContentInformation.

Parameters: infoObjectID - the identifier of the information object

dataObjectID - the identifier of the associated data object

repInfoID - the identifier of the associated representation information

objectStore - the object store where the object will be stored

Package

4.6.9 CLASS PACKAGED INFORMATION

InfoObject

PackagedInformation

All Implemented Interfaces:

InfoObjectInterface

---

public class PackagedInformation extends InfoObject

A logical container composed of optional Information Object(s).

4.6.9.1 Methods inherited from class InfoObject

deserialize, dump, getDataObject, getDataObjectID, getDataObjectStr, getInfoObjectID, getRepInfoDataObject, getRepInfoDataObjectID, getRepInfoDataObjectStr, getRepInfoDataObjectStr2, serialize, setDataObject, setInfoObjectID, setObjectStore, setRepInfoDataObject

4.6.9.2 Constructor Details

public PackagedInformation(Identifier infoObjectID, Identifier dataObjectID, Identifier repInfoID, ObjectStore objectStore)

Package

4.6.10 CLASS PACKAGING INFORMATION

PackagingInformation

---

public class PackagingInformation extends Object

A logical container composed of optional Information Object(s).

4.6.10.1 Constructor Details

public PackagingInformation()

Constructs a PackagingInformation instance.

4.7 CLASS SPECIFICATIONS – OAIS IF SERVICES

Class specifications are blueprints or templates that define the structure and behavior of objects. They serve as blueprints for creating instances of objects, which are individual occurrences based on the class specification. These class specifications have been abbreviated to include only the class name, definition, and methods/constructors.

The following class specifications are for the OAIS Interoperability Framework's services.

Package

4.7.1 CLASS ARCHIVAL STORAGE

ArchivalStorage

All Implemented Interfaces:

`ArchivalStorageInterface`

---

public class ArchivalStorage extends Object implements ArchivalStorageInterface

ArchivalStorage is a class that represents the archival storage component and implements the ArchivalStorageInterface.

4.7.1.1 Method Details

4.7.1.1.1 getInfoObjectQueryRequest

public InfoObject getInfoObjectQueryRequest(InfoObject infoObjectQueryString)

Description copied from interface: `ArchivalStorageInterface`

The getInfoObjectQueryRequest method queries the Archival Storage for one or more Information Objects using the provided query string.

Specified by: `getInfoObjectQueryRequest` in interface `ArchivalStorageInterface`

Parameters: `infoObjectQueryString` - a query string that identifies one or more Information Objects.

Returns: InfoObject an Information Object.

4.7.1.1.2 getDataObject

public DataObject getDataObject(Identifier identifier)

Description copied from interface: `ArchivalStorageInterface`

The getDataObject method get a Data Object from Archival Storage using an identifier.

Specified by: `getDataObject` in interface `ArchivalStorageInterface`

Parameters: `identifier` - the identifier of the receiver

Returns: DataObject the Data Object component of an Information Object

4.7.1.1.3 getRepInfo

public RepInfoDataObject getRepInfo(Identifier identifier)

Description copied from interface: ArchivalStorageInterface

The getRepInfo method get the Representation Information component of an Information Object from Archival Storage using an identifier.

Specified by: getRepInfo in interface ArchivalStorageInterface

Parameters: identifier - the identifier of the Representation Information

Returns: RepInfoDataObject the Representation Information component

4.7.1.1.4 getObjectStore

public ObjectStore getObjectStore()

Description copied from interface: ArchivalStorageInterface

The getObjectStore method gets this client's Object Store

Specified by: getObjectStore in interface ArchivalStorageInterface

Returns: ObjectStore The Object Store for this client.

4.7.1.1.5 setInfoObject

public void setInfoObject(Identifier identifier, InfoObject infoObject)

Description copied from interface: ArchivalStorageInterface

The setInfoObject method set an Information Object in Archival Storage using an identifier.

Specified by: setInfoObject in interface ArchivalStorageInterface

Parameters: identifier - the identifier of the receiver

infoObject - the Information Object to be stored.

4.7.1.1.6 setDataObject

public void setDataObject(Identifier identifier, DataObject dataObject)

Description copied from interface: ArchivalStorageInterface

The setDataObject method set a Data Object in Archival Storage using an identifier.

Specified by: setDataObject in interface ArchivalStorageInterface

Parameters: identifier - the identifier of the Data Object

dataObject - the Data Object to be stored

4.7.1.1.7 setRepInfo

public void setRepInfo(Identifier identifier, RepInfoDataObject repInfo)

Description copied from interface: ArchivalStorageInterface

The setRepInfo method set a Representation Information component of an Information Object to Archival Storage using an identifier. .

Specified by: setRepInfo in interface ArchivalStorageInterface

Parameters: identifier - the identifier of the Representation Information

repInfo - The Representation Information to be stored

4.7.1.1.8 putObjectStore

public void putObjectStore(ObjectStore objectStore)

Description copied from interface: ArchivalStorageInterface

The putObjectStore method put this client's Object Store

Specified by: putObjectStore in interface ArchivalStorageInterface

Parameters: objectStore - the object store for this client

4.7.1.1.9 getCurrInfoObject

public InfoObject getCurrInfoObject()

Gets the current InfoObject from the object store.

Returns: the current InfoObject

4.7.1.1.10 getInfoObjectStoreArr

public ArrayList<InfoObject> getInfoObjectStoreArr()

Gets the ArrayList of InfoObjects from the object store.

Returns: the ArrayList of InfoObjects

4.7.1.1.11 getInfoObjectIdArr

public ArrayList<String> getInfoObjectIdArr()

Gets the ArrayList of InfoObject IDs from the object store.

Returns: the ArrayList of InfoObject IDs

4.7.1.1.12 putInfoObjectGeneric

public void putInfoObjectGeneric(String lIdentifierRawStr, String clientType, Object lDOContent, Object lRIContent, String objectTypeId)

Stores a generic InfoObject with the given parameters in the object store.

Parameters: `lIdentifierRawStr` - the raw string identifier

`clientType` - the client type

`lDOContent` - the content of the data object

`lRIContent` - the content of the representation information

`objectTypeId` - the object type ID

4.7.1.1.13 putInfoObject

public InfoObject putInfoObject(Identifier lIOId, Identifier lDOId, Object lDOContent, Identifier lRIID, Object lRIContent)

Creates an InfoObject with the given parameters, stores it in the object store, and returns it.

Parameters: `lIOId` - the identifier of the InfoObject

`lDOId` - the identifier of the DataObject

`lDOContent` - the content of the DataObject

`lRIID` - the identifier of the RepInfoDataObject

`lRIContent` - the content of the RepInfoDataObject

Returns: the created InfoObject

4.7.1.1.14 getNextClientIdentifierRawStr

public String getNextClientIdentifierRawStr()

Gets the next client identifier raw string.

Returns: the next client identifier raw string

Package

4.7.2 CLASS MESSAGE

InfoObject

Message

All Implemented Interfaces:

`InfoObjectInterface`, `MessageInterface`

Direct Known Subclasses: `ProtocolDataUnit`

---

public class Message extends InfoObject implements MessageInterface

The Message class represents a message.

4.7.2.1 Method Details

4.7.2.1.1 getIdentifier

public Identifier getIdentifier()

Description copied from interface: `MessageInterface`

The getIdentifier method returns the identifier of this message.

Specified by: `getIdentifier` in interface `MessageInterface`

Returns: Identifier the identifier of this message

4.7.2.1.2 getSenderId

public Identifier getSenderId()

Retrieves the identifier of the sender.

Specified by: `getSenderId` in interface `MessageInterface`

Returns: the identifier of the sender

4.7.2.1.3 getSenderIdStr

public String getSenderIdStr()

Retrieves the string representation of the sender identifier.

Returns: the string representation of the sender identifier

4.7.2.1.4 getReceiverId

public Identifier getReceiverId()

Retrieves the identifier of the receiver.

Specified by: `getReceiverId` in interface `MessageInterface`

Returns: the identifier of the receiver

4.7.2.1.5 getReceiverIdStr

public String getReceiverIdStr()

Retrieves the string representation of the receiver identifier.

Returns: the string representation of the receiver identifier

4.7.2.1.6 getSenderIO

public Object getSenderIO()

Retrieves the object representing the sender.

Specified by: `getSenderIO` in interface `MessageInterface`

Returns: the object representing the sender

4.7.2.1.7 getReceiverIO

public Object getReceiverIO()

Retrieves the object representing the receiver.

Specified by: `getReceiverIO` in interface `MessageInterface`

Returns: the object representing the receiver

4.7.2.1.8 getIsActive

public Boolean getIsActive()

Retrieves the status of the message.

Returns: the status of the message

4.7.2.1.9 resetIsActive

public void resetIsActive()

Resets the status of the message to inactive.

4.7.2.1.10 setIdentifier

public void setIdentifier(Identifier identifier)

Description copied from interface: `MessageInterface`

The setIdentifier method sets the identifier of this message

Specified by: `setIdentifier` in interface `MessageInterface`

Parameters: `identifier` - the identifier of the message

4.7.2.1.11 setSenderId

public void setSenderId(Identifier senderId)

Description copied from interface: `MessageInterface`

The setSenderId method sets the identifier of the sender receiver.

Specified by: `setSenderId` in interface `MessageInterface`

Parameters: `senderId` - the identifier of the sender

4.7.2.1.12 setReceiverId

public void setReceiverId(Identifier receiverId)

Description copied from interface: `MessageInterface`

The setReceiverId method sets the identifier of the receiver

Specified by: `setReceiverId` in interface `MessageInterface`

Parameters: `receiverId` - the identifier of the receiver

4.7.2.1.13 setSenderIO

public void setSenderIO(Object senderObject)

Description copied from interface: `MessageInterface`

The setSenderIO method sets the object from the sender

Specified by: `setSenderIO` in interface `MessageInterface`

Parameters: `senderObject` - the object being sent from the sender to the receiver

4.7.2.1.14 setReceiverIO

public void setReceiverIO(Object receiverObject)

Description copied from interface: `MessageInterface`

The setReceiverIO method sets the object from the receiver receiver.

Specified by: `setReceiverIO` in interface `MessageInterface`

Parameters: `receiverObject` - the object being sent from the receiver to the sender

4.7.2.1.15 dump

public void dump(String note)

Dumps the contents of the message to the console.

Parameters: `note` - additional notes to include in the dump

Package

4.7.3 CLASS OBJECT STORE

ObjectStore

All Implemented Interfaces:

`ObjectStoreInterface`

---

public class ObjectStore extends Object implements ObjectStoreInterface

A class representing an Object Store that stores InfoObjects, DataObjects, and RepInfoDataObjects.

4.7.3.1 Method Details

4.7.3.1.1 getCurrInfoObject

public InfoObject getCurrInfoObject()

Retrieves the current InfoObject.

Returns: the current InfoObject

4.7.3.1.2 getInfoObject

public InfoObject getInfoObject(Identifier identifier)

Description copied from interface: `ObjectStoreInterface`

Retrieves the InfoObject associated with the specified identifier.

Specified by: `getInfoObject` in interface `ObjectStoreInterface`

Parameters: `identifier` - The identifier of the InfoObject to retrieve.

Returns: The InfoObject associated with the identifier, or null if not found.

4.7.3.1.3 getInfoObject

public InfoObject getInfoObject(String identifierStr)

Retrieves the InfoObject associated with the given identifier string.

Parameters: `identifierStr` - the identifier string

Returns: the InfoObject

4.7.3.1.4 getDataObject

public DataObject getDataObject(Identifier identifier)

Description copied from interface: `ObjectStoreInterface`

Retrieves the DataObject associated with the specified identifier.

Specified by: `getDataObject` in interface `ObjectStoreInterface`

Parameters: `identifier` - The identifier of the DataObject to retrieve.

Returns: The DataObject associated with the identifier, or null if not found.

4.7.3.1.5 getRepInfo

public RepInfoDataObject getRepInfo(Identifier identifier)

Retrieves the RepInfoDataObject associated with the given identifier.

Specified by: `getRepInfo` in interface `ObjectStoreInterface`

Parameters: `identifier` - the identifier

Returns: the RepInfoDataObject

4.7.3.1.6 setInfoObject

public void setInfoObject(Identifier identifier, InfoObject infoObject)

Description copied from interface: `ObjectStoreInterface`

Stores the specified InfoObject with the given identifier.

Specified by: `setInfoObject` in interface `ObjectStoreInterface`

Parameters: `identifier` - The identifier to associate with the InfoObject.

`infoObject` - The InfoObject to store.

4.7.3.1.7 setDataObject

public void setDataObject(Identifier identifier, DataObject dataObject)

Description copied from interface: `ObjectStoreInterface`

Stores the specified DataObject with the given identifier.

Specified by: `setDataObject` in interface `ObjectStoreInterface`

Parameters: `identifier` - The identifier to associate with the DataObject.

`dataObject` - The DataObject to store.

4.7.3.1.8 setRepInfo

public void setRepInfo(Identifier identifier, RepInfoDataObject repInfo)

Description copied from interface: `ObjectStoreInterface`

Stores the specified RepInfoDataObject with the given identifier.

Specified by: `setRepInfo` in interface `ObjectStoreInterface`

Parameters: `identifier` - The identifier to associate with the RepInfoDataObject.

`repInfo` - The RepInfoDataObject to store.

4.7.3.1.9 getObjectIdentifierStrArr

public ArrayList<String> getObjectIdentifierStrArr()

Retrieves an ArrayList of object identifier strings.

Returns: the ArrayList of object identifier strings

4.7.3.1.10 getInfoObjectIDArr

public ArrayList<String> getInfoObjectIDArr()

Retrieves an ArrayList of InfoObject identifier strings.

Returns: the ArrayList of InfoObject identifier strings

4.7.3.1.11 getInfoObjectStoreArr

public ArrayList<InfoObject> getInfoObjectStoreArr()

Retrieves an ArrayList of InfoObjects stored in the ObjectStore.

Returns: the ArrayList of InfoObjects

Package

4.7.4 CLASS REQUEST

Request

All Implemented Interfaces: `RequestInterface`

---

public class Request extends Object implements RequestInterface

4.7.4.1 Method Details

4.7.4.1.1 requestEnqueue

public void requestEnqueue(Identifier messageID, Message message)

Description copied from interface: RequestInterface

The Request Enqueue method enqueues the provided message

Specified by: requestEnqueue in interface RequestInterface

Parameters: messageID - the identifier of the Message to enqueue

message - the Message to enqueue

4.7.4.1.2 requestEnqueueMap

public void requestEnqueueMap(Identifier messageID, Message message)

4.7.4.1.3 requestEnqueueHTTP

public void requestEnqueueHTTP(Identifier messageID, Message message)

4.7.4.1.4 requestDequeue

public boolean requestDequeue(Identifier messageID)

Description copied from interface: RequestInterface

The requestDequeue method dequeues the provided message

Specified by: requestDequeue in interface RequestInterface

Parameters: messageID - the identifier of the Message to dequeue

Returns: boolean true if the dequeue was successful

4.7.4.1.5 returnRequest

public OAISIFClient returnRequest(Message lMessage)

Description copied from interface: RequestInterface

The requestDequeue method dequeues the provided message

Specified by: returnRequest in interface RequestInterface

Parameters: lMessage - the original Message containing the query, an InfoObject

Returns: OAISIFClient the client receiving the message

4.8 CLASS SPECIFICATIONS – OAIS IF SWITCHBOARD

Class specifications are blueprints or templates that define the structure and behavior of objects. They serve as blueprints for creating instances of objects, which are individual occurrences based on the class specification. These class specifications have been abbreviated to include only the class name, definition, and methods/constructors.

The following class specifications are for the OAIS Interoperability Framework's switchboard.

Package

4.8.1 CLASS SWITCH BOARD ENTRY

SwitchBoardEntry

---

public class SwitchBoardEntry extends Object

The SwitchBoardEntry class represents an entry in the switchboard of the OAIS (Open Archival Information System). It contains various properties related to the switchboard entry.

4.8.1.1 Constructor Summary

Description

```
SwitchBoardEntry(String sysObjIdentifierStr, String title
, String description, String portId, String authenticatio
nMethodType, String serializationType, String protocolTyp
e, String queryLanguageType, String clientType, String cl
ientTypeInd, String clientIdInitSeqNum)
```

Constructs a SwitchBoardEntry object with the provided parameters.

4.8.1.2 Field Details

4.8.1.2.1 identifier

public Identifier identifier

4.8.1.2.2 sysObjIdentifierStr

public String sysObjIdentifierStr

The system object identifier string of the switchboard entry.

4.8.1.2.3 title

public String title

The title of the switchboard entry.

4.8.1.2.4 description

public String description

The description of the switchboard entry.

4.8.1.2.5 portId

public String portId

The port ID of the switchboard entry.

4.8.1.2.6 authenticationMethodType

public String authenticationMethodType

The authentication method type of the switchboard entry.

4.8.1.2.7 serializationType

public String serializationType

The serialization type of the switchboard entry.

4.8.1.2.8 protocolType

public String protocolType

The protocol type of the switchboard entry.

4.8.1.2.9 queryLanguageType

public String queryLanguageType

The query language type of the switchboard entry.

4.8.1.2.10 clientType

public String clientType

The client type of the switchboard entry.

4.8.1.2.11 clientTypeInd

public String clientTypeInd

The client type indicator of the switchboard entry.

4.8.1.2.12 clientIdInitSeqNum

public String clientIdInitSeqNum

The initial sequence number of the client ID for the switchboard entry.

4.8.1.3 Method Details

4.8.1.3.1 getClientIdentifier

public Identifier getClientIdentifier()

Retrieves the identifier of the switchboard entry.

Returns: The identifier of the switchboard entry.

4.8.1.3.2 getSysObjIdentifierStr

public String getSysObjIdentifierStr()

Retrieves the system object identifier string of the switchboard entry.

Returns: The system object identifier string.

4.8.1.3.3 getClientTitle

public String getClientTitle()

Retrieves the title of the switchboard entry.

Returns: The title of the switchboard entry.

4.8.1.3.4 getClientDescription

public String getClientDescription()

Retrieves the description of the switchboard entry.

Returns: The description of the switchboard entry.

4.8.1.3.5 getClientIdInitSeqNum

public String getClientIdInitSeqNum()

Retrieves the initial sequence number of the client ID for the switchboard entry.

Returns: The initial sequence number of the client ID.

4.8.1.3.6 getClientPortId

public String getClientPortId()

Retrieves the port ID of the switchboard entry.

Returns: The port ID of the switchboard entry.

4.8.1.3.7 getClientAuthenticationMethodType

public String getClientAuthenticationMethodType()

Retrieves the authentication method type of the switchboard entry.

Returns: The authentication method type of the switchboard entry.

4.8.1.3.8 getClientSerializationType

public String getClientSerializationType()

Retrieves the serialization type of the switchboard entry.

Returns: The serialization type of the switchboard entry.

4.8.1.3.9 getClientProtocolType

public String getClientProtocolType()

Retrieves the protocol type of the switchboard entry.

Returns: The protocol type of the switchboard entry.

4.8.1.3.10 getClientQueryLanguageType

public String getClientQueryLanguageType()

Retrieves the query language type of the switchboard entry.

Returns: The query language type of the switchboard entry.

4.8.1.3.11 getClientType

public String getClientType()

Retrieves the client type of the switchboard entry.

Returns: The client type of the switchboard entry.

4.8.1.3.12 getClientTypeInd

public String getClientTypeInd()

Retrieves the client type indicator of the switchboard entry.

Returns: The client type indicator of the switchboard entry.

4.8.1.3.13 getSwitchBoardEntryAsStr

public String getSwitchBoardEntryAsStr(String delimiter)

Retrieves the switchboard entry properties as a concatenated string using the specified delimiter.

Parameters: `delimiter` - The delimiter to use for concatenating the properties.

Returns: The switchboard entry properties as a concatenated string.

Package

4.8.2 CLASS SWITCH BOARD STATE

SwitchBoardState

All Implemented Interfaces:

`SwitchBoardInterface`

---

public class SwitchBoardState extends Object implements SwitchBoardInterface

The SwitchBoardState class represents the state of the Switch Board.

4.8.2.1 Method Details

4.8.2.1.1 getSwitchBoardEntry

public SwitchBoardEntry getSwitchBoardEntry(String identifier)

Retrieves a SwitchBoardEntry from the Switch Board based on the given identifier.

Specified by: `getSwitchBoardEntry` in interface `SwitchBoardInterface`

Parameters: `identifier` - The identifier of the Switch Board entry.

Returns: The SwitchBoardEntry associated with the identifier.

4.8.2.1.2 getActiveClientById

public OAISIFClient getActiveClientById(String identifier)

Retrieves an active client from the Switch Board based on the given identifier.

Specified by: `getActiveClientById` in interface `SwitchBoardInterface`

Parameters: `identifier` - The identifier of the active client.

Returns: The active OAISIFClient associated with the identifier.

4.8.2.1.3 getActiveClientIdArr

public ArrayList<String> getActiveClientIdArr()

Retrieves an ArrayList of identifiers for active clients in the Switch Board.

Specified by: `getActiveClientIdArr` in interface `SwitchBoardInterface`

Returns: An ArrayList containing the identifiers of active clients.

4.8.2.1.4 getCandidateClientIdArr

public ArrayList<String> getCandidateClientIdArr()

Retrieves an ArrayList of identifiers for candidate clients in the Switch Board.

Specified by: `getCandidateClientIdArr` in interface `SwitchBoardInterface`

Returns: An ArrayList containing the identifiers of candidate clients.

4.8.2.1.5 getSwitchBoardStateAsString

public String getSwitchBoardStateAsString()

Retrieves the state of the Switch Board as a string representation.

Specified by: `getSwitchBoardStateAsString` in interface `SwitchBoardInterface`

Returns: The state of the Switch Board as a string.

4.8.2.1.6 setSwitchBoardState

public void setSwitchBoardState(OAISIFClient lOAISIFClient)

Sets the state of the Switch Board using the given OAISIFClient.

Specified by: `setSwitchBoardState` in interface `SwitchBoardInterface`

Parameters: `lOAISIFClient` - The OAISIFClient to set as the Switch Board state.

4.9 CLASS SPECIFICATIONS – OAIS IF GENERIC CLIENT

Class specifications are blueprints or templates that define the structure and behavior of objects. They serve as blueprints for creating instances of objects, which are individual occurrences based on the class specification. These class specifications have been abbreviated to include only the class name, definition, and methods/constructors.

The following class specifications are for the OAIS Interoperability Framework's generic client.

Package

4.9.1 CLASS OAISIF CLIENT

OAISIFClient

All Implemented Interfaces:

`AdapterInterface`

---

public class OAISIFClient extends SpecificAdapter

4.9.1.1 Methods inherited from class SpecificAdapter

`getIdentifierSpecific`, `getObjectStore`, `putInfoObjectGeneric`

4.9.1.2 Methods inherited from class GenericAdapter

`asyncGetPackage`, `asyncSendPackage`, `getArchivalStorage`, `getClientPortId`, `getClientType`, `getClientTypeInd`, `getEndPoi`

nts, getIdentifier, getSwitchBoardEntry, getTitle, syncSendPackage

4.9.1.3 Method Details

4.9.1.3.1 getInfoObjectStoreArr

public ArrayList<InfoObject> getInfoObjectStoreArr()

4.9.1.3.2 getSysObjIdentifier

public Identifier getSysObjIdentifier()

4.9.1.3.3 getNextIdentifierRawStr

public String getNextIdentifierRawStr()

Description copied from class: SpecificAdapter

Retrieves the next raw identifier string.

Overrides: getNextIdentifierRawStr in class SpecificAdapter

Returns: The next identifier string.

4.9.1.3.4 requestInformationObjectGeneric

public void requestInformationObjectGeneric(boolean isInfoObjectRequest)

4.9.1.3.5 requestInformationObjectSwitchBoardState

public void requestInformationObjectSwitchBoardState()

4.9.1.3.6 sendInfoObjectGeneric

public void sendInfoObjectGeneric()

4.9.1.3.7 returnRequestGeneric

public void returnRequestGeneric()

4.9.1.3.8 queryRepositoryRaw

public String queryRepositoryRaw(ArrayList<String> queryStringClauseArr)

4.10 CLASS SPECIFICATIONS – RDF MANAGER

Class specifications are blueprints or templates that define the structure and behavior of objects. They serve as blueprints for creating instances of objects, which are individual occurrences based on the class specification. These class specifications have been abbreviated to include only the class name, definition, and methods/constructors.

The following class specifications are for the RDF manager. For this implementation of the OAISIF, RDF/OWL is used to capture the OAIS RM Information Model so that SPARQL queries can be used for semantic search and discovery using the entity name defined in the OAIS RM.

Package

4.10.1 CLASS RDF4J MGR

RDF4JMgr

---

public class RDF4JMgr extends Object

RDF4JMgr class provides methods for managing RDF models and executing SPARQL queries.

4.10.1.1 Method Details

4.10.1.1.1 loadModel

public void loadModel()

Opens an input stream, parses the file, and loads the model.

4.10.1.1.2 getResultSet

public ArrayList<ArrayList<String>> getResultSet(String queryString)

Executes a SPARQL query and returns the result set.

Parameters: `queryString` - The SPARQL query string.

Returns: The result set as an array of arrays of strings.

4.10.1.1.3 getSparQLQuery

public String getSparQLQuery(String id)

Retrieves a specific SPARQL query string from the registry.

Parameters: `id` - The ID of the SPARQL query.

Returns: The SPARQL query string.

4.10.1.1.4 getSparQLQueryPrefix

public String getSparQLQueryPrefix()

Retrieves the SPARQL query prefix.

Returns: The SPARQL query prefix string.

4.10.1.1.5 getSparqlQueryTitleArr

public static ArrayList<String> getSparqlQueryTitleArr()

Retrieves the SPARQL query title array.

Returns: The SPARQL query title array.

4.10.1.1.6 getBoundVarbArr

public ArrayList<String> getBoundVarbArr(String queryString)

Retrieves the array of bound variables from a SPARQL query string.

Parameters: `queryString` - The SPARQL query string.

Returns: The array of bound variables.

4.11 CLASS SPECIFICATIONS – REST HTTP STANDARD METHODS

Class specifications are blueprints or templates that define the structure and behavior of objects. They serve as blueprints for creating instances of objects, which are individual occurrences based on the class specification. These class specifications have been abbreviated to include only the class name, definition, and methods/constructors.

The following class specifications are for REST, an architectural style used in designing networked applications. Interactions between clients and servers are performed through standard HTTP methods. RESTful systems use standard URIs (Uniform Resource Identifiers) to uniquely identify resources in various formats, such as JSON or XML.

Package

4.11.1 CLASS ECHO GET HANDLER

EchoGetHandler

All Implemented Interfaces:

`com.sun.net.httpserver.HttpHandler`

---

public class EchoGetHandler extends Object implements com.sun.net.httpserver.HttpHandler

An HTTP handler that handles GET requests and provides an echo response based on the query parameters.

4.11.1.1 Method Details

4.11.1.1.1 handle

public void handle(com.sun.net.httpserver.HttpExchange he) throws IOException

Specified by: `handle` in interface `com.sun.net.httpserver.HttpHandler`

Throws: `IOException`

Package

4.11.2 CLASS ECHO HEADER HANDLER

EchoHeaderHandler

All Implemented Interfaces: `com.sun.net.httpserver.HttpHandler`

---

public class EchoHeaderHandler extends Object implements com.sun.net.httpserver.HttpHandler

An HTTP handler that handles requests and echoes back the headers of the request in the response.

4.11.2.1 Method Details

4.11.2.1.1 handle

public void handle(com.sun.net.httpserver.HttpExchange he) throws IOException

Specified by: `handle` in interface `com.sun.net.httpserver.HttpHandler`

Throws: `IOException`

Package

4.11.3 CLASS ECHO POST HANDLER

EchoPostHandler

All Implemented Interfaces:

`com.sun.net.httpserver.HttpHandler`

---

public class EchoPostHandler extends Object implements com.sun.net.httpserver.HttpHandler

An HTTP handler that handles POST requests and echoes back the request parameters in the response.

4.11.3.1 Method Details

4.11.3.1.1 handle

public void handle(com.sun.net.httpserver.HttpExchange he) throws IOException

Specified by: `handle` in interface `com.sun.net.httpserver.HttpHandler`

Throws: `IOException`

Package

4.11.4 CLASS HTTP REQUEST

HTTPRequest

---

public class HTTPRequest extends Object

The HTTPRequest class represents an HTTP request to a specified URL.

4.11.4.1 Method Details

4.11.4.1.1 getHTTPRequestResultStr

public String getHTTPRequestResultStr()

Retrieves the result of the HTTP request as a string.

Returns: the result of the HTTP request

Package

4.11.5 CLASS HTTP SERVER MAIN

HTTPServerMain

---

public class HTTPServerMain extends Object

The HTTPServerMain class represents the main entry point for the HTTP server.

4.11.5.1 Method Details

4.11.5.1.1 parseQuery

public
static void parseQuery(String query, Map<String,Object> parameters) throws UnsupportedEncodingException

Parses an HTTP query string and populates the provided map with key-value pairs.

Parameters: `query` - the HTTP query string

`parameters` - the map to store the parsed key-value pairs

Throws: `UnsupportedEncodingException` - if the encoding is not supported

Package

4.11.6 CLASS ROOT HANDLER

RootHandler

All Implemented Interfaces:

`com.sun.net.httpserver.HttpHandler`

---

public class RootHandler extends Object implements com.sun.net.httpserver.HttpHandler

The root handler for HTTP requests.

4.11.6.1 Method Details

4.11.6.1.1 handle

public void handle(com.sun.net.httpserver.HttpExchange he) throws IOException

Handles the HTTP request and sends a response to the client.

Specified by: `handle` in interface `com.sun.net.httpserver.HttpHandler`

Parameters: `he` - The HttpExchange object representing the HTTP request and response.

Throws: `IOException` - If an I/O error occurs while handling the request.

Package

4.11.7 CLASS END POINT CONNECTION

Identifier

EndPointConnection

All Implemented Interfaces:

`IdentifierInterface`

---

public class EndPointConnection extends Identifier implements IdentifierInterface

An EndPointConnection is an extension of Identifier and names an instance.

Package

4.12 CLASS SPECIFICATIONS – MAIN APPLICATION

Class specifications are blueprints or templates that define the structure and behavior of objects. They serve as blueprints for creating instances of objects, which are individual occurrences based on the class specification. These class specifications have been abbreviated to include only the class name, definition, and methods/constructors.

The following class specifications are for the main application.

4.12.1 CLASS ACME MAIN

ACMEMain

---

public class ACMEMain extends Object

The ACMEMain class is the entry point of the OAIS IF application. It initializes the OAIS driver and starts the application.

4.12.1.1 Method Details

4.12.1.1.1 main

public static void main(String[] args)

The main method is the entry point of the OAIS IF application. It initializes the OAIS driver and starts the application.

Parameters: `args` - command line arguments

Package

4.12.2 CLASS ACME GUI

ACMEGUI

All Implemented Interfaces:

`ImageObserver`, `MenuContainer`, `Serializable`, `Accessible`, `RootPaneContainer`, `WindowConstants`

---

public class ACMEGUI extends JFrame

4.12.2.1 Method Details

4.12.2.1.1 reloadClientTables

public void reloadClientTables()

4.12.2.1.2 setJTableColumnWidthAndFont

protected void setJTableColumnWidthAndFont(JTable lJTable)

Package

4.12.3 CLASS ACME DRIVER

ACMEDriver

All Implemented Interfaces:

ImageObserver, MenuContainer, Serializable, Accessible, RootPaneContainer, WindowConstants

---

public class ACMEDriver extends JFrame

4.12.3.1 Method Details

4.12.3.1.1 isClientRunning

public static boolean isClientRunning(String title)

4.12.3.1.2 setClientRunning

public static boolean setClientRunning(String title)

4.12.3.1.3 getRequest

public Request getRequest()

4.12.3.1.4 getNextSysObjIdentifier

public static Identifier getNextSysObjIdentifier(String type)

4.12.3.1.5 getNextSysObjIdentifierStr

public static String getNextSysObjIdentifierStr(String type)

4.12.3.1.6 getNextGlobalIdentifierRawStr

public static String getNextGlobalIdentifierRawStr()

4.12.3.1.7 getClientNameList

public static ArrayList<String> getClientNameList()

4.12.3.1.8 getClientNameListUpper

public static ArrayList<String> getClientNameListUpper()

4.12.3.1.9 getAvailableQueries

public static Object[] getAvailableQueries()

4.12.3.1.10 getSelectedQueryString

public static String getSelectedQueryString(String queryKey)

4.12.3.1.11 OKDialogNotImplemented

public static void OKDialogNotImplemented(JFrame lJFrame)

4.12.3.1.12 OKDialogDisabled

public static void OKDialogDisabled(JFrame lJFrame)

4.12.3.1.13 MessageInfoOK

public
static void MessageInfoOK(JFrame lJFrame, String lDialogTitle, String lMessageText)

4.12.3.1.14 MessageWarningOK

public
static void MessageWarningOK(JFrame lJFrame, String lDialogTitle, String lMessageText)

4.12.3.1.15 MessageWarningYesNo

public
static boolean MessageWarningYesNo(JFrame lJFrame, String lDialogTitle, String lMessageText)

4.12.3.1.16 MessageErrorOK

public
static void MessageErrorOK(JFrame lJFrame, String lDialogTitle, String lMessageText)

4.12.3.1.17 getUserInputSingleFrames

public
static String getUserInputSingleFrames(JFrame lJFrame, String lDialogLabel, String lInputLabel)

4.13 CLASS SPECIFICATIONS – MISCELLANEOUS

Class specifications are blueprints or templates that define the structure and behavior of objects. They serve as blueprints for creating instances of objects, which are individual occurrences based on the class specification. These class specifications have been abbreviated to include only the class name, definition, and methods/constructors.

The following class specifications are for miscellaneous code used for initiating registries with test data, displaying results, etc.

Package

4.13.1 CLASS ARCHIVAL STORAGE CLIENT INITIATE

ArchivalStorageClientInitiate

---

public class ArchivalStorageClientInitiate extends Object

Package

4.13.2 CLASS OAISIF DISPLAY INFO OBJECT

OAISIFDisplayInfoObject

All Implemented Interfaces:

ImageObserver, MenuContainer, Serializable, Accessible, RootPaneContainer, WindowConstants

---

public class OAISIFDisplayInfoObject extends JFrame

A class representing the display of an InfoObject in the OAIS interface.

4.13.2.1 Method Details

4.13.2.1.1 setJTableColumnWidthAndFont

protected void setJTableColumnWidthAndFont(JTable lJTable)

ANNEX A

IMPLEMENTATION CONFORMANCE STATEMENT (ICS) PROFORMA

(NORMATIVE)

A1 INTRODUCTION

OVERVIEW

This annex provides the Implementation Conformance Statement (ICS) Requirements List (RL) for an implementation of [Specification]. The ICS for an implementation is generated by completing the RL in accordance with the instructions below. An implementation claiming conformance must satisfy the mandatory requirements referenced in the RL.

ABBREVIATIONS AND CONVENTIONS

The RL consists of information in tabular form. The status of features is indicated using the abbreviations and conventions described below.

Item Column

The item column contains sequential numbers for items in the table.

Feature Column

The feature column contains a brief descriptive name for a feature. It implicitly means "Is this feature supported by the implementation?"

Status Column

The status column uses the following notations:

- M mandatory;
- O optional;
- C conditional;
- X prohibited;
- I out of scope;
- N/A not applicable.

Support Column Symbols

The support column is to be used by the implementer to state whether a feature is supported by entering Y, N, or N/A, indicating:

Y Yes, supported by the implementation.

N No, not supported by the implementation.

N/A Not applicable.

The support column should also be used, when appropriate, to enter values supported for a given capability.

INSTRUCTIONS FOR COMPLETING THE RL

An implementer shows the extent of compliance to the Recommended Standard by completing the RL; that is, the state of compliance with all mandatory requirements and the options supported are shown. The resulting completed RL is called an ICS. The implementer shall complete the RL by entering appropriate responses in the support or values supported column, using the notation described in 0. If a conditional requirement is inapplicable, N/A should be used. If a mandatory requirement is not satisfied, exception information must be supplied by entering a reference X*i*, where *i* is a unique identifier, to an accompanying rationale for the noncompliance.

A2 ICS PROFORMA FOR [SPECIFICATION]

GENERAL INFORMATION

Identification of ICS

A2.1.1.1 Test

Date of Statement (DD/MM/YYYY)	
ICS serial number	
System Conformance statement cross-reference	

Identification of Implementation Under Test

Implementation Name	
Implementation Version	
Special Configuration	

Other Information	

Identification of Supplier

Supplier	
Contact Point for Queries	
Implementation Name(s) and Versions	
Other information necessary for full identification, e.g., name(s) and version(s) for machines and/or operating systems; System Name(s)	

Identification of Specification

[CCSDS Document Number]	
Have any exceptions been required? NOTE – A YES answer means that the implementation does not conform to the Recommended Standard. Non-supported mandatory capabilities are to be identified in the ICS, with an explanation of why the implementation is non-conforming.	Yes [ ] No [ ]

REQUIREMENTS LIST

[See CCSDS A20.1-Y-1, *CCSDS Implementation Conformance Statements* (Yellow Book, Issue 1, April 2014).]

ANNEX B

SECURITY, SANA, AND PATENT CONSIDERATIONS

(INFORMATIVE)

B1 SECURITY CONSIDERATIONS

SECURITY CONCERNS WITH RESPECT TO THE CCSDS DOCUMENT

CCSDS requires there to be an informative annex which points out security considerations for implementations of its standards, including those implementing archival systems based on this OAIS Reference Model. It must be borne in mind that the Reference Model itself is not a design and does not specify any particular implementation techniques.

General guidance on security issues may be found in the CCSDS Informational Report, The Application of CCSDS Protocols to Secure Systems (reference [D9]) and references therein.

To be conformant to the OAIS Reference Model an implementation should use the Information Model and follow the mandatory requirements in 3.2. Appendix F of the document provides annotations on those mandatory requirements and provides some guidance on security concerns.

The OAIS Interoperability Framework (OAIS-IF) defines a set of interfaces for an abstract Generic Adapter, a class designed for requesting and sending Information Objects. The implementation of the interfaces are not provided. The implementation of a Generic Adapter should follow the mandatory requirements in 3.2 of the OAIS Reference Model. Similarly, the implementation of a Specific Adapter, an extension of the Generic Adapter designed to interface with specific client and archive interfaces should also following the mandatory requirements in 3.2. Additional information on possible implementations are provided in the Green Book. (reference needed)

POTENTIAL THREATS AND ATTACK SCENARIOS

The abstract Generic Adapter is designed to use a communications interaction protocol to send and receive Information Objects. The standard does not provide an implementation. The implementation of a Generic Adapter must address the concomitant security issues.

CONSEQUENCES OF NOT APPLYING SECURITY TO THE TECHNOLOGY

The abstract Generic Adapter is designed to use a communications interaction protocol to send and receive Information Objects. Not addressing security issues in the implementation of a Generic Adapter would results in serious security violations.

B2 SANA CONSIDERATIONS

[See CCSDS 313.0-Y-1, *Space Assigned Numbers Authority (SANA)—Role, Responsibilities, Policies, and Procedures* (Yellow Book, Issue 1, July 2011).]

B3 PATENT CONSIDERATIONS

[See CCSDS A20.0-Y-4, *CCSDS Publications Manual* (Yellow Book, Issue 4, April 2014).]

ANNEX C

INFORMATIVE REFERENCES

(INFORMATIVE)

[C1] *Reference Model For An Open Archival Information System (OAIS)*. Issue 2.1. CCSDS Record (Pink Book), CCSDS 650.0-P-2.1. Washington, D.C.: CCSDS, October 2020.