INTERNATIONAL STANDARD

ISO/IEC 4396-3

First edition

Telecommunications and information exchange between systems — Recursive inter-network architecture —

Part 3: Common distributed application protocol

Télécommunications et échange d'information entre systèmes — Architecture récursive inter-réseaux —

Partie 3: Protocole pour les applications distribuées CDAP

PROOF/ÉPREUVE

Reference number
ISO/IEC 4396-3:2023(E)

ISO copyright office
CP 401 • Ch. de Blandonnet 8
CH-1214 Vernier, Geneva
Phone: +41 22 749 01 11
Email: copyright@iso.org
Website: www.iso.org

Published in Switzerland

Contents

Page

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work.

The procedures used to develop this document and those intended for its further maintenance are described in the ISO/IEC Directives, Part 1. In particular, the different approval criteria needed for the different types of document should be noted. This document was drafted in accordance with the editorial rules of the ISO/IEC Directives, Part 2 (see www.iso.org/directives or www.iec.ch/members_experts/refdocs).

ISO and IEC draw attention to the possibility that the implementation of this document may involve the use of (a) patent(s). ISO and IEC take no position concerning the evidence, validity or applicability of any claimed patent rights in respect thereof. As of the date of publication of this document, ISO and IEC had received notice of (a) patent(s) which may be required to implement this document. However, implementers are cautioned that this may not represent the latest information, which may be obtained from the patent database available at www.iso.org/patents and https://patents.iec.ch. ISO and IEC shall not be held responsible for identifying any or all such patent rights.

Any trade name used in this document is information given for the convenience of users and does not constitute an endorsement.

For an explanation of the voluntary nature of standards, the meaning of ISO specific terms and expressions related to conformity assessment, as well as information about ISO's adherence to the World Trade Organization (WTO) principles in the Technical Barriers to Trade (TBT) see www.iso.org/iso/foreword.html. In the IEC, see www.iec.ch/understanding-standards.

This document was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 6 *Telecommunications and information exchange between systems*.

A list of all parts in the ISO/IEC 4396 series can be found on the ISO and IEC websites.

Any feedback or questions on this document should be directed to the user's national standards body. A complete listing of these bodies can be found at www.iso.org/members.html and www.iec.ch/national-committees.

Introduction

The common distributed application protocol (CDAP) is used by communicating recursive inter-network architecture (RINA) applications to exchange application-specific data and effect remote actions. CDAP implements the protocol messages and state machines to allow the two endpoints of a communication flow to exchange read/write, start/stop, and create/delete method invocations on remote "objects". The semantics of those objects and operations are opaque to the CDAP protocol itself. Because CDAP is not specific to RINA, it can be used by any distributed application that needs to share information or initiate state changes with another application over a network. CDAP is used in the RINA architecture by applications, and specifically by inter-process communication (IPC) Processes, which are specialized applications that cooperate to create a Distributed IPC Facility (DIF) that provides network transport to other applications.

Telecommunications and information exchange between systems — Recursive inter-network architecture —

Part 3: Common distributed application protocol

1 Scope

This document provides the common distributed application protocol (CDAP) specification. CDAP enables distributed applications to deal with communications at an object level, rather than forcing applications to explicitly deal with serialization and input/output operations. CDAP provides the application protocol component of a distributed application facility (DAF). CDAP provides a straightforward and unifying approach to sharing data over a network without having to create specialized protocols.

This document provides:

— an overview of CDAP;

— the specification of CDAP;

— a description of policies, in the specific sense introduced in the text;

— notes on the context of CDAP.

2 Normative references

The following documents are referred to in the text in such a way that some or all of their content constitutes requirements of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

ISO/IEC 4396-1, *Telecommunications and information exchange between systems – Recursive Inter-Network Architecture — Part 1: RINA Reference Model*

3 Terms and definitions

For the purposes of this document, the terms and definitions given in ISO/IEC 4396-1, and the following apply.

ISO and IEC maintain terminology databases for use in standardization at the following addresses:

— ISO Online browsing platform: available at https://www.iso.org/obp

— IEC Electropedia: available at https://www.electropedia.org/

3.1
application object model
common distributed application protocol (CDAP) object model used in the application connection (AC) data transfer phase.

Note 1 to entry: If an AE is capable of supporting multiple object models, this selects the one to be used for this AC. If not, the requested object model value should match that of the sole implemented model.

3.2
CDAP message encoding rules
encoding of bits sent for all CDAP messages in an application connection

Note 1 to entry: The concrete syntax to be used for the AC is determined by the Common Application Connection Establishment Phase (CACEP) and provided to the CDAP data phase in the application connection state vector (ACSV).

3.3
CDAP policy
CDAP implementation option where something can be done in multiple ways, and no single choice is mandated

Note 1 to entry: CDAP implementers should make a choice of how to implement every CDAP policy; recommended policies are provided in this document.

3.4
CDAP profile
specific and complete definition of all the policies in CDAP and the object model that should be implemented as described in order to create a conforming, compatible implementation

3.5
CDAP syntax version
abstract syntax
integer that indicates version of abstract syntax used

3.6
method call
request for the recipient application-entity (AE) to make a call to a method (function) specific to an object

3.7
object
named conceptual data entity residing within the resource information base (RIB) view made available to an application connection (AC) by the application

Note 1 to entry: Objects have a class (type), a name and possibly an interchangeable ObjectID (OID), and a value. The object [in an unspecified way, as object/class models and inheritance are unspecified by common distributed application protocol (CDAP)] provides a set of methods (functions) that correspond to the CDAP message types, which can become the targets of method calls on that object during the data transfer phase of the AC.

3.8
object method
method (function) associated with an object that will be invoked by receipt of a corresponding CDAP message

3.9
object model
entire set of objects available to an application connection (AC) and their collective relationships and behavioursNote 1 to entry: The object model to be used for an AC is selected by the Application Object Model variable set during the Common Application Connection Establishment Phase (CACEP).

3.10
object model version
specific named object model used for an application connection

Note 1 to entry: Common distributed application protocol (CDAP) does not dictate the form of Object Model Version names/values; values are mutually agreed-upon by a priori application agreement, determined by Common Application Connection Establishment Phase (CACEP), and possibly made available in the application connection state vector (ACSV) for use by objects.

3.11
policy point
place to optimize or configure a common distributed application protocol (CDAP) implementation to perform some operation

Note 1 to entry: Policy points have the form POL-CDAP-x.

4 Description of CDAP

4.1 CDAP –RINA application protocol

The word “application” is used in this document in a general sense. Independent application programs, individual instances of a distributed program, or even independent threads of a single process execution can be considered to be “applications” if they communicate via CDAP. In typical RINA usage, the “applications” are Application Entities (AEs) residing within two instances of a Distributed Application Process connected by a RINA flow.

The objects that CDAP operates upon are mutually-agreed-upon representations of the exposed data and entities between two communicating applications. CDAP does not assume that the applications share any directly-accessible memory, so CDAP is an example of a Remote Procedure Call protocol. RINA refers to the set of these known objects as a Resource Information Base (RIB), a virtual database of objects that the two applications are mutually aware of and able to address via CDAP.

CDAP does not assume that the RIB exposes the entire state of either application, or that the objects manipulated by CDAP map 1:1 onto actual state variables or entities; CDAP simply communicates requested operations on objects that the communicating applications choose to expose. The objects can actually be virtual objects synthesized for external presentation or to generate side effects; they can be considered to be “projections” or “views” of implementation objects, created for convenience or abstraction purposes. While a hierarchical tree-based organization is often used to unambiguously name and locate objects in a RIB, CDAP itself imposes no particular structure.

CDAP borrows from Common Management Information Protocol (CMIP) a straightforward standards-defined protocol and shares many properties with other remote-method-invocation based protocols such as the Bell Labs Plan 9 “file protocol”. While known concepts are employed to benefit from the experience of prior implementations, the design of CDAP explicitly takes into account that implementations will be done in multiple languages and on various platforms of widely different scales, so some lowest-common-denominator choices have been adopted, and many details have been left undefined in the base protocol to allow tailoring CDAP implementations for their end use.

CDAP does not define a single implementation specification, but is rather a framework in which specific application, communication, and scale requirements can be met while still providing higher-level application code with a consistent set of operations and behaviors. CDAP does this by providing a core set of mechanisms with a variable set of policies that allow but encapsulate and limit the desired variability. Specific instances of the set of variable aspects of the protocol, e.g. message encoding syntax, that are characterized in this specification as policies, can be standardized to ensure interoperability of different implementations. This document provides default policies as examples; in the absence of problems with these defaults for a specific usage, it is recommended that CDAP implementations adopt them to promote reuse and prevent unnecessary divergence.

4.2 Application entities (AEs) within applications

RINA defines an AE as a portion of an application that performs some distinct function. An AE communicates with one or more compatible AEs in other applications to accomplish a specific function of the application. For example, one AE in an application can deal with database access, another with authentication, and another with reporting and management, each of which operates over and needs access to a specific, and potentially distinct, set of remote data that is relevant to its function.

Within each communicating application, there will be one or more AE that performs a particular purpose of communication, with a corresponding set of objects that have state, a usage protocol (i.e. legal sequences of operations on the objects), and specific semantics. In practice, AEs may be implemented via sub-routine libraries, threads, objects, or some other processing model, but the AE concept is present whether explicit or implicit.

Application connections are created between an AE instance of one application and a corresponding, compatible AE instance of another application. Applications may implement any number of AEs, and those may conduct any number of application connections, with any number of other applications.

From a security standpoint, it is important to restrict the access permissions of an AE to only those RIB objects relevant to its operation. While CDAP does not itself implement that separation, it provides applications with the information needed to do so.

4.3 Objects

All modern programming languages can implement the concept of an object that encapsulates data and a set of operations upon it. Properties, e.g. fundamental types, inheritance, extensibility, safety properties, aggregation capabilities, are richer in some languages than others. CDAP implements a bare-minimum model that can interface with virtually any existing object-oriented implementation approach. In the CDAP model, the AEs that are communicating with one another via an application connection share an object model that has a defined set of objects, object classes, and naming structure for the objects. Each object implements a specific set of method (function) calls that operate on the object.

4.4 Method calls on objects

CDAP messages each encode a method call, with arguments, that represents a request for a remote AE to perform that method call on one of its objects. A CDAP protocol implementation provides the mechanisms for a sequence of steps that include encoding the desired method call and its arguments into a message, sending the message over the RINA flow, decoding it, presenting the method call to the addressed object, possibly returning a result message, and completing the original method call. A goal of CDAP is to allow doing this sequence of steps in a common way, enabling a standardized programming API, in the face of different policies for, e.g. encoding syntax, implementation language, object models.

4.5 Object model

CDAP encodes read/write/start/stop/create/delete method calls on a set of objects that is specific to an application connection. A meaningful CDAP exchange requires that the applications agree fully upon the set of objects that can be addressed and their exposed properties, operations, and semantics. In this document, the "object model" is used to refer to the entire definition of the set of objects, including their class, their data description, their names and naming conventions, their interrelationships, their methods, their usage pattern, and their behaviors such as side-effects. An application connection is associated with a specific object model, determined when the application connection is established. An application can implement and expose, through its AEs, as many object models as appropriate to its purpose.

The underlying application data exposed in an object model as an “object” can actually be organized differently than it is presented to the other AE in the object model. These “virtual” objects may represent application data that needs to be aggregated, selected, computed, or iterated over in order to satisfy the semantics of a requested CDAP operation on the target virtual object. Any such operation is performed by the methods of the (virtual) objects that are exposed, not by CDAP.

Since objects and their semantics generally evolve with time, the Common Application Connection Establishment Phase (CACEP) protocol that is used to establish an application connection allows specifying at connection establishment time a “version” of the object model that will be used for the duration of that application connection. This makes it possible for an application to discover a potential incompatibility between the object model being used by the other application, and/or to have a vocabulary of multiple object model versions to use, and to choose the correct one to use for a particular

application connection depending on its partner application. This is especially useful during transition periods when an object model is being revised, and updated versions of applications that understand the new model are not yet fully propagated.

There is no CDAP-defined object model or set of objects that all applications using CDAP should implement. Object models are not part of CDAP itself.

4.6 Application connection

Application cooperation requires the applications to agree upon the syntax of the CDAP messages (CDAP can be encoded in multiple ways) and the object model, and further, to agree upon the privileges that each application grants to the other with respect to its own objects, based on identity, capabilities, or other security mechanism. An application connection operates within an establishment process, the CACEP, that authenticates the applications, determines the message syntax and object model to use, and provides these parameters for use in an application connection, prior to the data transfer phase where CDAP messages are exchanged, and operations are performed on objects.

4.7 Application connection state vector (ACSV)

The CACEP, either by message exchange, defaults, or other mutually-agreed-upon method (the specifics are policies of the applications and of CACEP, and are completely opaque to CDAP), determines the values of key application connection parameters before handing off operation on the flow to the data transfer phase. This information is provided to the CDAP implementation via an Application Connection State Vector (ACSV), which CDAP will consult when needed as it interprets messages. While the ACSV is not exchanged via CDAP messages and is therefore not specifically defined by the CDAP protocol, the contents are relevant to some policies, especially CACEP, which should provide some of the contents of the ACSV, and to some objects, e.g. those that should perform access permission or object model checks. Therefore, the ACSV is defined as a visible CDAP concept to ensure that the requirements that it places on CACEP and CDAP policies have a place to be made explicit.

4.8 Requestor and responder roles

Many interactions between two applications are asymmetric, such as a classic client/server exchange, but CDAP builds in no such constraint. The CDAP protocol is symmetric, in the sense that either side may make method calls via CDAP messages on objects of the other side, without regard to which side initiated the application connection. In this document, when the initiator of a message if referred to as the “requestor”, and the recipient of the message that (optionally) generates a reply as the “responder”, it is in the context of a specific message exchange, not the original flow or application connection establishment. The application AEs can have a specific role in the application connection and behave asymmetrically, but it is not relevant to CDAP operations.

4.9 Validation of values/operations by CDAP

The CDAP protocol provides the mechanism for communicating method calls with values, but has no inherent understanding of the semantics of objects or the meaning of the object fields and values. Those remain the responsibility of the applications and their object models and the policies they use. However, CDAP messages themselves have encoding and consistency requirements. Messages are parsed, generated, and have common operations performed on them (such as tracking replies) using mechanisms common to any object model, and many errors in forming or parsing CDAP messages are detected by those mechanisms of CDAP itself. Other semantic correctness requirements on the operations carried by CDAP messages, e.g. the range of a field value in a READ or WRITE message, are enforced by object methods; in such cases, CDAP mechanisms (such as sending a log message or even abandoning the application connection) may be invoked by objects to request CDAP mechanism assistance to respond to these errors.

To enhance the ability of objects to respond to and validate application connection specific parameters such as authentication, CDAP passes application connection parameters provided in the ACSV to object method operations for their examination. It is the responsibility of the method implementations

to validate the parameters extracted from a message, to restrict access to objects as necessary, and to generate and present appropriate views of objects, in accordance to the application connection parameters in the ACSV. For example, in many applications CACEP will have established credentials that may then be used to validate access permissions (CDAP's only role in this is to provide the credentials, via the ACSV, to objects for use in that access validation). In practice, such common actions should be factored out and performed by mechanisms outside the actual object method implementations. This is especially important for security-related operations such as access permission validation, to avoid introducing a defect opportunity into every object that is expected to perform a security check.

4.10 CDAP application programming interface (API)

This document does not specify an API for generating requests or receiving replies during the CDAP data phase. It only defines the messages that are created and consumed by such interfaces. The CDAP programming API is an application implementation choice. In fact, CDAP and object operations can be implicit in data operations on a RIB, with no explicit CDAP API provided. However, since all CDAP implementations have a common set of mechanisms, a common API that operates across multiple CDAP variations is straightforward to create.

4.11 Standardization and policies

The purpose of standardization of a protocol is to enable interoperability. This is achieved by intentionally limiting flexibility, typically by choosing a specific, usually single, way that implementations should encode communications, define the semantics of communication exchanges, and define all aspects and behaviors of the objects being communicated about. When a protocol is intended for a specific application, freezing options and making optimized trade-off choices is both possible and necessary.

However, the CDAP protocol is intended to apply to a broad range of applications, so rather than narrowing it to a specific range by setting all details in concrete a priori, this specification provides a framework that allows defining a variety of optimized fit-for-purpose CDAP protocol implementations.

In many parts of this document an implementation choice is designated as being a policy and a name for the policy is provided to aid in defining and documenting specific CDAP implementations. While there is flexibility provided in message encoding and other aspects of CDAP itself, the vast majority of variability and policy decisions involve the object model (the specific objects being manipulated via CDAP and their behaviors), which by their nature are application-specific. When a complete set of policies has been specified for a specific CDAP usage, including the object model, that specification combined with this specification defines a CDAP Profile. Two AEs that implement the same CDAP Profile can communicate meaningfully. Two AEs that implement different CDAP Profiles, in general, cannot.

In order to reduce unnecessary divergence among implementations, many suggested default policy choices are provided. In the absence of a compelling reason (e.g. highly-constrained resources or a pre-existing standard that constrains some aspect of the application), these default policy choices are recommended to be adopted. This topic is discussed further in 5.1.

5 Specification

5.1 CDAP profile — Policies and standardization

CDAP provides applications with a fixed set of methods, corresponding 1:1 to CDAP message types, to effect operations on remote objects via exchange of messages. In this document common properties of all CDAP implementations such as these are referred to as the mechanisms of the protocol. The mechanisms for exchange of CDAP operations are defined in this document, but the bit-level encoding of messages and the naming, values, types, and semantics and behaviors of the objects do not have a mandated single definition. These designed-in variability points are identified in this subclause as named policies that may vary, and are specified in detail in that context, when CDAP is used for a specific application. Policies can be selected to optimize a CDAP implementation for (e.g. size, speed, readability, flexibility, reuse, compatibility), to best serve the application needs.

When defining an interoperability standard for applications that use CDAP, a CDAP Profile is defined and mandated as part of that standard. For every named CDAP policy point described in this document, a CDAP Profile specifies the policy that compliant applications will use. To encourage commonality across implementations, a suggested default policy is provided for each policy point; object model definitions should adopt the default policy unless there is a compelling reason not to do so.

5.2 Application connection establishment

Application connection establishment in the RINA architecture is performed by the CACEP as described in which is loosely patterned after the Association Control Service Element (ACSE) protocol. The details of how the CACEP phase performs its function are not relevant to CDAP itself. To operate properly, CDAP depends upon establishment of the necessary set of initial conditions and parameter values prior to commencing CDAP operations. Conceptually, and commonly in an implementation, those values are communicated using the ACSV described below.

The data transfer phase is entered after the CACE phase completes successfully. The application connection ends when either application closes the flow, or detects that the flow has closed, or the application informs CACEP that the application connection is complete. All application connection information is discarded at that time, and a CACEP phase should be performed again if it is desired to resume operation.

The following is a summary of the CACEP process to show how CDAP fits within the enclosing establishment framework.

Steps for creation, parameter setting, and destruction of an application connection:

a) Establishment of a data flow (communication channel, or colloquially “connection”) between the applications.

   For full functionality, CDAP requires that the flow delivers Service Data Units (SDUs) transparently, in-order, loss-free, and with negligible error rate. Partial functionality can still be provided if the in-order and loss-free requirements are not met. The RINA flow allocation process with appropriate Quality of Service (QoS) parameters can be used to establish such a flow.

b) Establishment of parameters for an application connection.

   This phase first identifies the CACEP syntax and then carries out the CACEP protocol exchange, which is implemented via customizable CACEP policy, that authenticates the applications to each other to their mutual satisfaction (the policy can range from “no authentication required” to a bi-directional mandatory authentication). It also determines and initializes, again via CACEP policy, key parameters for the CDAP data phase, saving all required information in an ACSV for use by the AC and by RIB object methods. Fundamentally, CACEP establishes and then hands off a fully initialized application connection description to CDAP.

c) The application connection data transfer phase

   If the CACEP process proceeded to successful completion, the preconditions for CDAP execution will have been established, and AC parameters will have been stored in the ACSV. The applications then enter the data transfer phase of the AC in which CDAP messages are exchanged. The ACSV provides all necessary information for further CDAP and object operations.

d) If the CDAP AC is no longer needed, either application typically ends the AC by requesting CACEP to terminate it. CACEP would typically destroy the flow, but if the flow is reused, care should be exercised to ensure that no residual information from a previous AC persists except the flow and its properties.

5.3 Application connection state vector (ACSV)

The ACSV is provided to CDAP by the CACEP at the completion of the CACE phase, which begins the data transfer phase by invoking CDAP initialization logic with the ACSV. The CDAP initialization logic

will examine the ACSV and may choose to reject the AC and close it immediately if it is incomplete or inconsistent. It is recommended that the CDAP implementation provide CACEP with enough information to preclude initiation of an AC that cannot be supported by CDAP, or to participate with CACEP in negotiating the parameters. The precise contents of the ACSV are a policy of a specific CDAP Profile [POL-CDAP-ACSVcontents]. Contents of the ACSV may include implementation-specific information, but shall include at least the following fields that are referenced elsewhere in this document:

a) CDAP message encoding concrete syntax specification [type integer, value (POL-CDAP-CSYNTAX)].

   CDAP messages can be encoded using any desired syntax, provided that the implementations can appropriately encode and decode it. The only requirement is that both applications can encode and decode CDAP messages using the selected syntax correctly and that the syntax can encode the full range of values needed for CDAP message fields and for each of the addressable objects.

b) CDAP syntax version [type integer, value (POL-CDAP-OBJ-VERSION)].

   This is a value specifying which version of the CDAP specification is in use. It is solely intended to allow the CDAP message encoding (fields and their meanings) to evolve if necessary, and is intended to change only if incompatible changes should be made to the fundamental CDAP message types or fields. An application shall reject an AC if the Syntax Version does not correspond to a CDAP message syntax that it can accept and generate.

c) The flow (type flow_handle).

   This is used to perform I/O operations to/from the flow. Implementations may allow the handle to be queried to determine, for example, the QoS parameters or other properties of the flow. For example, if the QoS of the underlying flow indicates that it is unreliable, the application cannot assume that messages will always be received, or if received that the sender will always receive a requested reply, so the application should limit its CDAP operations to a subset that can withstand such uncertainty or should terminate the AC. The specific operations available on the flow_handle are implementation-dependent and not specified by CDAP, but a specific capability may be implied by specific policies.

d) The AE name requesting the AC (type string).

   This may be used by the application to further restrict or select the set of objects in the CDAP object model that can be operated upon by this AC [POL-CDAP-OBJ-VISIBILITY]. The meaning of this parameter is opaque to CDAP (CDAP's only role is to make it available to CDAP and to method calls invoked via CDAP messages).

e) The Object Model to be used for the AC (type integer, value described by [POL-CDAP-OBJ-VERSION]).

   In the event that multiple versions of the object model used by the AEs of the communicating applications exist, e.g. due to evolution of the object model, incomplete rollout of updated implementations, or unintentional implementation discrepancies, this value specifies which one the AC will use. If an AE implements a single object model, this value should match the value representing that model or the AC shall be terminated. This value is opaque to CDAP, it is made available to the method calls invoked by CDAP messages, which may adapt their behaviour to correspond to the version in use for the AC, but it is not examined by CDAP mechanisms.

f) Security information.

   If CACEP establishes an identity or other credentials to be used in validating permission [POL-CDAP-AUTH], this field provides privileges/identity/credentials of the other application for purposes of limiting the AC's access to objects in the view. The field includes the mandatory Verbose. The remainder of the field, if any, is specified in [POL-CDAP-AUTH]. The Boolean value Verbose in this field indicates to CDAP mechanisms whether CDAP encode/decode and other internal CDAP operations are to return detailed information on success or errors (if the Boolean is TRUE), or simple SUCCESS/FAILURE return values (if the Boolean is FALSE), allowing the policy to control potential leakage of implementation information to less-trusted applications by CDAP mechanisms. The Boolean may also be referenced for the same purpose by object methods

