INTERNATIONAL STANDARD

ISO/IEC 4396-2

First edition
2023-12

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

Part 2: Common application connection establishment procedure

Reference number
ISO/IEC 4396-2:2023(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 functions of creating an application connection between instances of application processes are to:

— exchange application naming information;

—) optionally, authenticate each;

—) establish the set of objects to which remote operations on the flow have access.·

This document defines the Common Application Connection Establishment Procedure (CACEP), patterned after the Association Control Service Element (ACSE) protocol. ACSE was chosen for three reasons:

— it already exists;

— it provides all of the necessary functions and no more;

— it was designed to be used recursively.

ACSE provides the basic requirements for exchanging application naming and context information and provides for an authentication module to be included.

Although the primary use of CACEP is to combine it with common distributed application protocol (CDAP) for the application information data transfer phase, there are situations when it is desirable to use CACEP with a different protocol in the data transfer phase, e.g. HTTP, in effect, wrapping CACEP establishment around a legacy protocol. CACEP exchanges naming information and provides for an authentication policy. CACEP is a protocol exchange over a flow that serves to authenticate flow participants to their mutual satisfaction and to initialize the application naming information and information related to the application protocol that will be used by applications to exchange information, e.g. abstract and encoding rules, object model versions. In the case of recursive inter-network architecture (RINA), the application protocol is CDAP

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

Part 2:
Common application connection establishment procedure

1 Scope

This document provides the common application connection establishment procedure (CACEP) specification. It includes an overview of CACEP, its specification, the syntax of the protocol data units (PDUs), and the policies available.

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 4396-1, *Telecommunications and information exchange between systems — Recursive inter-network architecture — Part 1: Reference model*

3 Terms and definitions

For the purposes of this document, the terms and definitions given in ISO 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 Naming Information
names used to reference an application that includes the required name Application Process Name and can include distributed application facility (DAF) name or distributed IPC (inter-process communication) facility (DIF) name, application process name, application process instance id, application entity id, and application entity instance id

4 Overview

The Initiating Process first allocates an (N-1)-flow with a destination application. When this is complete, it sends an A_CONNECT Request with the appropriate parameters and initiates the authentication policy. When the Authentication policy completes, a positive or negative A_CONNECT Response is returned by the destination application process and the connection is established.

5 Detailed specification

5.1 General

This specification describes both sides of the procedure for establishing communication between the Initiating and Responding Processes.

5.2 Definitions of the process states

5.2.1 General

Creating an applications connection will transition among the following states:

5.2.2 Initiating states

— Authenticating – a connection is in the process of being established and the initiator is authenticating within policy the responder and vice versa.

— ConnectPending – a Connect Request has been sent, awaiting a response.

— Established – the connection is established and PDUs can be exchanged.

— FlowPending – determining if the flow is well-formed.

— Null – the state machine is awaiting input.

— Releasing – the state machine is releasing all resources associated with this connection.

5.2.3 Responding states

— Authenticating – a connection is in process of being established and the responder is authenticating within policy the initiator and vice versa.

— ConnectPending – a Connect Request has been sent, awaiting a response.

— Established – the connection is established and PDUs can be exchanged.

— Null – the state machine is awaiting input.

— Releasing – the state machine is releasing all resources associated with this connection.

5.3 Allocate_Request.submit

5.3.1 When invoked

This primitive is invoked when a process has been instructed to create an application connection.

5.3.2 Action upon receipt

The Initiating-Process is provided with the Application-Naming-Information and a supporting DIF that can be used to contact the Destination Application Process. The Initiating-Process is in the NULL state and invokes an Allocate_Request to the supporting DIF:

— Allocate_Request(<DIF-Name>.any.DIF or URL, <Initiating-Process-name>, QoS parameters, access control parameters).

The Initiating-Process transitions to the FlowPending state.

5.4 Allocate_Response.deliver

5.4.1 When invoked

When the Initiating Application Process is in the *FlowPending* state and the supporting DIF receives a Create Flow Response.

5.4.2 Action upon receipt

If the Initiating Application Process is not in the FlowPending state, this is an error, the Initiating Process should invoke a Deallocate primitive and report the error appropriately. If the Allocate was successful, then there is an allocated flow with the requested QoS and the Initiating Process can transition from the FlowPending state to the ConnectPending state, sending a A_CONNECT PDU to the Responding Process. If the Response was negative, an error is returned, and the state returns to NULL.

5.5 Allocate_Indication

5.5.1 When invoked

When the named Destination Application in the NULL state and receives an Allocate_Indication from the supporting DIF.

5.5.2 Action upon receipt

The Responding Process is in the NULL state. If the Allocate_Indication specified an Application Process-Instance and Application-Entity-Instance and the instance was not in the NULL state, then this is an error; otherwise, a new instance is created and it is not an error.

If the Responding Process is not willing to accept the request, it returns a negative Allocate_Confirm to the supporting DIF. The state remains NULL. If the Responding Process is willing to accept the connection, it notifies the supporting DIF with a positive Allocate_Confirm primitive. The Responding Process transitions to ConnectPending state.

5.6 A_CONNECT Request

5.6.1 When invoked

When a supporting flow has been allocated, the Responding Process is in the *ConnectPending* State (waiting for an A_CONNECT Request).

5.6.2 Action upon receipt

If the Responding Entity-instance is not in the ConnectPending State, the Responding Entity-instance should send an A_RELEASE and invoke a Deallocate and abruptly terminate the connection.

Otherwise, If the PotentialConnection Timer is set then (this is a retry), the timer is cancelled. Otherwise, The Responding Entity-instance will evaluate the parameters of the A_CONNECT Request and if they are valid will invoke the Authentication module and transition to the Authenticating state.

If the Authentication module completes successfully, the Responding Entity instance sends a positive A_CONNECT Response and transitions to the Established state.

If Authentication fails, the Responding Entity-instance sends a negative *A_CONNECT* Response. The Responding Entity instance increments the count of number of retries. If the count is less than or equal MaxConnectRetries, then it sets a PotentialConnection Timer greater than 2MPL (Maximum Packet Lifetime) and remains in the ConnectPending State.

If the count is greater than MaxConnectRetries, the Responding Entity-instance sends an A_RELEASE (with no response required) and invokes a Deallocate primitive.

5.7 A_CONNECT Response

5.7.1 When invoked

When the Initiating Process has sent an A_CONNECT Request and transitioned to the Authenticating state, awaiting a positive or negative A_CONNECT Response.

5.7.2 Action upon receipt

If the Initiating Process is not in the ConnectPending State, this is an error. Send an A_RELEASE (with no response required) and invoke Deallocate.

If the A_CONNECT Response PDU indicated a failure, the Initiating Process may remain in the Connect Pending state and attempt to modify the parameters and send a new A_CONNECT Request.

Otherwise, it sends an A_RELEASE (with no response required) and invokes a Deallocate(port-id) on the flow and flags an error and transitions to the NULL state. The Initiating Process should also have a count of retries and give up after the maximum. However, this will be enforced by the responding process regardless.

If the A_CONNECT Response PDU indicated success, the Initiating Process transitions to the Established state.

5.8 A_DATA (Optional)

5.8.1 When invoked

When the Initiating or Responding Entity Instance is in the Established state. This PDU may be used with Applications that are not using CDAP but have multiple AEs to which data is being sent. This PDU type is not used with CDAP.

5.8.2 Action upon receipt

When an Application Process receives an A_DATA PDU, it inspects the AE-name or AE-instance-identifier to deliver the PDU to the appropriate instance.

5.9 A_RELEASE Request

5.9.1 When invoked

An A_RELEASE can be sent by either the Initiating or the Responding Process in any state. If the sender indicates a response is requested, it transitions to the Releasing state.

5.9.2 Action upon receipt

When the Initiating or the Responding Process, receives an A_RELEASE, if an A_RELEASE Response is not requested, it immediately invokes an Deallocate primitive and transitions to the NULL state.

If a A_RELEASE Response was requested, it sends an A_RELEASE Response and after 2MPL+A invokes Deallocate. All state associated with this connection is deallocated.

5.10 A_RELEASE Response

5.10.1 When invoked

When an Initiating or Responding Process has received an A_RELEASE indicating a response is desired.

5.10.2 Action upon receipt

The Process receiving an A_RELEASE with a response requested, sends an A_RELEASE Response, invokes a Deallocate primitive and transitions to the NULL state. All state associated with this connection is deallocated.

5.11 Deallocate indication

5.11.1 When invoked

This primitive maybe invoked at any time and in any state. These transitions are not included in the state diagrams to facilitate clarity.

5.11.2 Action upon receipt

The receiving process immediately deallocates any state associated with this application connection and transitions to the NULL state.

5.12 A_UNIT_DATA (optional)

5.12.1 When invoked

This PDU type allows an Application Process to send a single PDU to another Application Process without creating an application connection. The advantage of this PDU type is that a 2-way exchange is avoided. The disadvantage is that the PDU would have to carry all of the application naming and authentication in each PDU. The "user-data" field can be a CDAP command.

To use an *A_UNIT_DATA* PDU, the Application Process shall allocate a flow with a supporting DIF.

5.12.2 Action upon receipt

When an A_UNIT_DATA arrives, the application process inspects the Application Naming Information, processes the Authentication information, if present and delivers the user-data to the AE-instance indicated.

5.13 Potential enrollment timer

5.13.1 When invoked

This event is invoked upon expiration of the PotentialEnrollment Timers.

5.13.2 Action upon receipt

When this event occurs, the Responding Process invokes a Dealloate primitive for the flow and all resources associated with the flow and the application-entity instances are deallocated.

6 Syntax of the PDUs

6.1 General

The preferred encoding of the PDUs is contained in this clause. In Annex A, provided as an informative Annex, there are legacy encoding rules. It should be noted that the legacy rules can be used as encoding rules for the ASN.1 syntax.

6.2 ASN.1 Definition

```
CACEP DEFINITIONS AUTOMATIC TAGS ::= BEGIN
CACEP-Message ::= SEQUENCE {
   operation CACEP-OPERATION.&opCode ({AllOperations}),
   opData CACEP-OPERATION.&MessageType ({AllOperations} {@operation})
}

OpCode ::= ENUMERATED { -- List of possible operation codes
        connect,
        connectResponse,
        release,
        releaseResponse,
        data,
        request,
        requestResponse
      }

CACEP-OPERATION ::= CLASS {
   &opCode OpCode UNIQUE,
   &MessageType
}
WITH SYNTAX {
   OPCODE &opCode MESSAGE &MessageType
}

AllOperations CACEP-OPERATION ::= {
   InvokeOperations |
   ResponseOperations |
   OtherOperations
}

InvokeOperations CACEP-OPERATION ::= {
   connectOperation |
   releaseOperation |
   requestOperation
}

ResponseOperations CACEP-OPERATION ::= {
   connectOperationResponse |
   releaseResponseOperation |
   requestResponseOperation
}

OtherOperations CACEP-OPERATION ::= {
   dataOperation
}

connectOperation CACEP-OPERATION ::= {OPCODE connect MESSAGE Connect}

connectOperationResponse   CACEP-OPERATION   ::=   {OPCODE connectResponse   MESSAGE
ConnectResponse}

releaseOperation CACEP-OPERATION ::= {OPCODE release MESSAGE Release}

releaseResponseOperation   CACEP-OPERATION   ::=   {OPCODE releaseResponse   MESSAGE
ReleaseResponse}
```

```
requestOperation CACEP-OPERATION ::= {OPCODE request MESSAGE Request}

requestResponseOperation   CACEP-OPERATION   ::=   {OPCODE requestResponse   MESSAGE
RequestResponse}

dataOperation CACEP-OPERATION ::= {OPCODE data MESSAGE Data}

Connect ::= SEQUENCE {
   absSyntax   AbstractSyntaxID,
   authMech   AuthenticationMechanismName OPTIONAL,
   authValue   AuthenticationValue OPTIONAL,
   destAEInst   DestinationApplicationEntityInstanceId OPTIONAL,
   destAEName   DestinationApplicationEntityName,
   destAPInst   DestinationApplicationProcessInstanceId OPTIONAL,
   destAPName   DestinationApplicationProcessName,
   flags   Flags DEFAULT {},
   invokeID   InvokeID,
   srcAEInst   SourceApplicationEntityInstanceId OPTIONAL,
   srcAEName   SourceApplicationEntityName,
   srcAPInst   SourceApplicationProcessInstanceId OPTIONAL,
   srcAPName   SourceApplicationProcessName,
   version   Version
}

ConnectResponse ::= SEQUENCE {
   absSyntax   AbstractSyntaxID,
   authMech   AuthenticationMechanismName OPTIONAL,
   authValue   AuthenticationValue OPTIONAL,
   destAEInst   DestinationApplicationEntityInstanceId OPTIONAL,
   destAEName   DestinationApplicationEntityName OPTIONAL,
   destAPInst   DestinationApplicationProcessInstanceId OPTIONAL,
   destAPName   DestinationApplicationProcessName OPTIONAL,
   flags   Flags DEFAULT {},
   invokeID   InvokeID,
   resultReason   ResultReason OPTIONAL,
   result   CACEP-Result,
   srcAEInst   SourceApplicationEntityInstanceId OPTIONAL,
   srcAEName   SourceApplicationEntityName OPTIONAL,
   srcAPInst   SourceApplicationProcessInstanceId OPTIONAL,
   srcAPName   SourceApplicationProcessName OPTIONAL,
   version   Version
}

Release ::= SEQUENCE {
   flags   Flags DEFAULT {},
   invokeID   InvokeID OPTIONAL,
   version   Version OPTIONAL
}

ReleaseResponse ::= SEQUENCE {
   flags   Flags DEFAULT {},
   invokeID   InvokeID OPTIONAL,
   resultReason   ResultReason OPTIONAL,
   result   CACEP-Result,
   version   Version OPTIONAL
}

Data ::= SEQUENCE {
   destAEInst   DestinationApplicationEntityInstanceId OPTIONAL,
   destAEName   DestinationApplicationEntityName OPTIONAL,
   flags   Flags DEFAULT {},
   srcAEInst   SourceApplicationEntityInstanceId OPTIONAL,
   srcAEName   SourceApplicationEntityName OPTIONAL,
   version   Version OPTIONAL,
   userData   UserData
}

Request ::= SEQUENCE {
   absSyntax   AbstractSyntaxID OPTIONAL,
   destAEInst   DestinationApplicationEntityInstanceId OPTIONAL,
   destAEName   DestinationApplicationEntityName OPTIONAL,
```

```
   destAPInst   DestinationApplicationProcessInstanceId OPTIONAL,
   destAPName   DestinationApplicationProcessName OPTIONAL,
   flags   Flags DEFAULT {},
   invokeID   InvokeID OPTIONAL,
   srcAEInst   SourceApplicationEntityInstanceId OPTIONAL,
   srcAEName   SourceApplicationEntityName OPTIONAL,
   srcAPInst   SourceApplicationProcessInstanceId OPTIONAL,
   srcAPName   SourceApplicationProcessName OPTIONAL,
   userData   UserData
}

RequestResponse ::= SEQUENCE {
   absSyntax   AbstractSyntaxID OPTIONAL,
   destAEInst   DestinationApplicationEntityInstanceId OPTIONAL,
   destAEName   DestinationApplicationEntityName OPTIONAL,
   destAPInst   DestinationApplicationProcessInstanceId OPTIONAL,
   destAPName   DestinationApplicationProcessName OPTIONAL,
   flags   Flags DEFAULT {},
   invokeID   InvokeID OPTIONAL,
   resultReason   ResultReason OPTIONAL,
   result   CACEP-Result,
   srcAEInst   SourceApplicationEntityInstanceId OPTIONAL,
   srcAEName   SourceApplicationEntityName OPTIONAL,
   srcAPInst   SourceApplicationProcessInstanceId OPTIONAL,
   srcAPName   SourceApplicationProcessName OPTIONAL,
   userData   UserData
}

AbstractSyntaxID ::= INTEGER

AuthenticationMechanismName ::= UTF8String

AuthenticationValue ::= OCTET STRING

DestinationApplicationEntityInstanceId ::= CACEP-String

DestinationApplicationEntityName ::= CACEP-String

DestinationApplicationProcessInstanceId ::= CACEP-String

DestinationApplicationProcessName ::= CACEP-String

Flags ::= BIT STRING {
           reserved1 (0),
           reserved2 (1),
           -- omitted for brevity
           reserved32 (31)
         } -- These flag bits will be defined by implementations as policy.

InvokeID ::= INTEGER

ResultReason ::= CACEP-String

SourceApplicationEntityInstanceId ::= CACEP-String

SourceApplicationEntityName ::= CACEP-String

SourceApplicationProcessInstanceId ::= CACEP-String

SourceApplicationProcessName ::= CACEP-String

Version ::= INTEGER

UserData ::= OCTET STRING

CACEP-Result ::= INTEGER {
        success (0),
        fail (-1),
        osError (-2)
      } -- Note that these are not the only possible error values. --
```

```
CACEP-String ::= UTF8String

END
```

7 Policies

Authentication policies are the subject of other specifications. A NULL authentication policy can be used if no authentication is required, with the following values for *authMech* and *authValue*:

— authMech: PSOC_Authentication_NULL

— authValue: not present

With this policy the authentication step does not require any processing and is always positive.

Annex A
(informative)

Legacy encoding rules

The encoding of the PDU type (opCode) and other values in the PDU into a form used for communication on a “wire” is referred to as the concrete syntax of the message. Agreement between communicating applications on the concrete syntax to use is a fundamental requirement for communication. Once this is established, it becomes possible to discuss other aspects of the communication, such as the version of the message declarations (the abstract syntax) and object definitions and values at the application level (usually summarized as a “version”) in use.

For a CACEP connection, after the data transfer flow is established the first PDU sent from one application (the requestor) to another (the responder) is a request for connection. The first PDUs exchanged, the A_CONNECT and A_CONNECT_R messages, shall include the concrete syntax value that corresponds to the CACEP message encoding method being used. This is done by encoding the concrete syntax in the first byte(s) of the message so that it can be examined before making the decision to use Google Protocol Buffer (GPB) syntax, vs. a different one, to interpret the remainder of the message. In the A_CONNECT PDU, the value in the first byte(s) of the PDU designates the concrete syntax used to encode the entire PDU as described in the next paragraph. After connection establishment, the applications are free to use a different concrete syntax, if their agreed-to protocol version so indicates. In the A_CONNECT_R response, the first byte of the PDU has the same value, representing acceptance, or a different one, representing a counter-proposal. If either party fails to recognize the syntax value it receives (either the concrete or abstract syntax), or understands it but refuses to use that syntax, it discards the PDU and the connection establishment fails.

In the defined GPB encoding of CACEP PDUs, encoding the value of the abstract syntax identifier as the very first field of the PDU will produce the constant value of 0x08 (a 32-bit Varint with field number = 1) in the first byte of the message. Encoding the GPB message fields in canonical order, the usual behaviour will achieve this automatically. The CACE protocol engine can check for this value to recognize that the syntax is GPB and can then safely parse the remainder of the PDU. If other concrete syntaxes are defined for CDAP in the future, they will use a different value for the first byte of the PDU, making it possible for multiple concrete syntaxes to exist contemporaneously.

Besides the PDU type (as indicated by the Opcode), there are a number of defined fields that may be present in a PDU. This Annex provides an overview of these fields, their descriptions, and some of the constraints on their values. Each of these has a value whose meaning is consistent across PDU types, but the value may or may not be relevant or present in a specific PDU. The declaration of these fields and their types, to be shared among compatible applications, is the abstract syntax definition which in GPB is coded into a .proto file. Since it is possible that CACEP can evolve, there is an allowance for the possibility by defining an integer corresponding to a version of the abstract syntax.

Table A.1 shows the mapping from CACEP abstract syntax formal names to Google Protocol Buffer PDU field names and types, with a very brief description. Additional details are provided in a later section. Notational convention: field names are identifiers defined in a GPB .proto type declaration; field names begin with a lower-case letter, and use capitalization to indicate embedded word-concatenation or abbreviations.

There are very few representations for typed data in GPB. The ones used in Table A.1 are listed in the table key.

In the GBP implementation, optional fields can have a default value that is assumed by the receiver if the field is not present in a message. This property can be exploited in common cases to permit applications to omit predictable fields, shortening the PDUs. Where applicable, the default values for the fields below are identified in the prototype definition of the GPB PDU format.