

IEEE Standard for Information Technology— ACSE and Presentation Layer Services— Application Program Interface (API) [C Language Binding]

Circuits and Devices

Communications Technology

IEEE Computer Society

Sponsored by the
Portable Applications Standards Committee

*Electromagnetics and
Radiation*

Energy and Power

Industrial Applications

*Signals and
Applications*

*Standards
Coordinating
Committees*

IEEE Std 1353-1994



Published by the Institute of Electrical and Electronics Engineers, Inc., 345 East 47th Street, New York, NY 10017, USA.

December 22, 1994



Documents

Document Sharing Hub

IEEE Standard for Information Technology— ACSE and Presentation Layer Services— Application Program Interface (API) [C Language Binding]

Sponsor
Portable Applications Standards Committee
of the
IEEE Computer Society

Approved September 22, 1994
IEEE Standards Board

Abstract: A C language binding to IEEE Std 1351-1994, a language-independent standard defining an Application Program Interface (API) to the ACSE application service element from the OSI application layer and the connection-oriented services of the presentation layer is defined. The purpose of this API is to support portable implementations of application-specific OSI service elements and non-OSI applications.

Keywords: ACSE, API, application layer, application portability, C binding, language binding, open systems, open systems interconnection, OSI, presentation layer

The Institute of Electrical and Electronics Engineers, Inc.
345 East 47th Street, New York, NY 10017-2394, USA

Copyright © 1994 by the
Institute of Electrical and Electronics Engineers, Inc.
All rights reserved. Published 1994.
Printed in the United States of America.

ISBN 1-55937-466-7

*No part of this publication may be reproduced in any form,
in an electronic retrieval system or otherwise,
without the prior written permission of the publisher.*

IEEE Standards documents are developed within the Technical Committees of the IEEE Societies and the Standards Coordinating Committees of the IEEE Standards Board. Members of the committees serve voluntarily and without compensation. They are not necessarily members of the Institute. The standards developed within IEEE represent a consensus of the broad expertise on the subject within the Institute as well as those activities outside of IEEE that have expressed an interest in participating in the development of the standard.

Use of an IEEE Standard is wholly voluntary. The existence of an IEEE Standard does not imply that there are no other ways to produce, test, measure, purchase, market, or provide other goods and services related to the scope of the IEEE Standard. Furthermore, the viewpoint expressed at the time a standard is approved and issued is subject to change brought about through developments in the state of the art and comments received from users of the standard. Every IEEE Standard is subjected to review at least every five years for revision or reaffirmation. When a document is more than five years old and has not been reaffirmed, it is reasonable to conclude that its contents, although still of some value, do not wholly reflect the present state of the art. Users are cautioned to check to determine that they have the latest edition of any IEEE Standard.

Comments for revision of IEEE Standards are welcome from any interested party, regardless of membership affiliation with IEEE. Suggestions for changes in documents should be in the form of a proposed change of text, together with appropriate supporting comments.

Interpretations: Occasionally questions may arise regarding the meaning of portions of standards as they relate to specific applications. When the need for interpretations is brought to the attention of IEEE, the Institute will initiate action to prepare appropriate responses. Since IEEE Standards represent a consensus of all concerned interests, it is important to ensure that any interpretation has also received the concurrence of a balance of interests. For this reason IEEE and the members of its technical committees are not able to provide an instant response to interpretation requests except in those cases where the matter has previously received formal consideration.

Comments on standards and requests for interpretations should be addressed to:

Secretary, IEEE Standards Board
445 Hoes Lane
P.O. Box 1331
Piscataway, NJ 08855-1331
USA

IEEE standards documents may involve the use of patented technology. Their approval by the Institute of Electrical and Electronics Engineers, Inc. does not mean that using such technology for the purpose of conforming to such standards is authorized by the patent owner. It is the obligation of the user of such technology to obtain all necessary permissions.

Contents

	PAGE
Introduction	v
Section 1: General	1
1.1 Scope	1
1.2 Normative References	1
1.3 Conformance	2
1.4 Test Methods	3
Section 2: Terminology and General Requirements	5
2.1 Conventions	5
2.2 Definitions	5
Section 3: Overview (Informative)	9
3.1 Functions and Mechanisms	9
3.1.1 User Data Buffering	9
3.1.2 Error Reporting	11
Section 4: APS Environment	13
4.1 Environment Attribute Identifiers/Types	13
Section 5: Service Interface	17
5.1 Introduction	17
5.2 C Type Definitions	17
5.2.1 ap_aei_api_id_t	17
5.2.2 ap_aeq_t	17
5.2.3 ap_apr_t	18
5.2.4 ap_a_assoc_env_t	18
5.2.5 ap_cdata_t	20
5.2.6 ap_cdl_t	20
5.2.7 ap_cdr_t	21
5.2.8 ap_conn_id_t	21
5.2.9 ap_dcn_t	21
5.2.10 ap_dcs_t	22
5.2.11 ap_diag_t	22
5.2.12 ap_objid_t	22
5.2.13 ap_octet_string_t	23
5.2.14 ap_old_conn_id_t	23
5.2.15 ap_osi_vbuf_t/ap_osi_dbuf_t	23
5.2.16 ap_paddr_t	24
5.2.17 ap_pollfd	25
5.2.18 ap_qos_t	26

5.2.19 Other LI Datatypes	27
5.3 Functions	30
5.3.1 Bind Address	30
5.3.2 Close an APS Instance	31
5.3.3 Error Text	32
5.3.4 Free Memory	32
5.3.5 Get Environment Attribute Value	34
5.3.6 Initialize Environment	35
5.3.7 Control Software Interrupts	36
5.3.8 Look at Received Primitive	37
5.3.9 Open Instance	38
5.3.10 Poll for Events	40
5.3.11 Receive Primitive	41
5.3.12 Restore APS Instance	43
5.3.13 Save an APS Instance	44
5.3.14 Set Environment Attribute Value	45
5.3.15 Send Primitive	46
5.4 Client-Supplied Memory Functions	47
5.4.1 Client-Supplied Memory Allocation Function	47
5.4.2 Client-Supplied Memory Deallocation Function	48
5.5 Error Codes	49
Section 6: Primitives	53
6.1 Primitive Identifiers	53
Annex A (informative) Example <xap.h> Header	55
Annex B (informative) Bibliography	65
Alphabetic Topical Index	67
FIGURES	
Figure 3-1 – Basic Buffer Structures	10
Figure 3-2 – Multiple Reference Buffer Example	10
TABLES	
Table 4-1 – Environment Attribute Identifiers/Types	14
Table 5-1 – Identifiers for the ap_a_assoc_env_t.mask element	19
Table 5-2 – Identifiers for the ap_cdata_t.sync_type element	20
Table 5-3 – Identifiers for the ap_nsap_t typedef	25
Table 5-4 – Identifiers for the ap_pollfd typedef	25
Table 5-5 – Identifiers for the ap_qos_t YES/NO fields	27
Table 5-6 – Identifiers for the ap_qos_t.priority field	27
Table 5-7 – LI Boolean Array Datatypes—Mapped to Unsigned Long	27

Table 5-8 – LI State Datatypes—Mapped to Unsigned Long	29
Table 5-9 – LI Miscellaneous Datatypes—Mapped to Unsigned Long	29
Table 5-10 – LI Operation/C Function Name Mapping	30
Table 5-11 – Event Mask Identifiers	37
Table 5-12 – LI—C Binding Error Code Mapping	50

Introduction

(This introduction is not a normative part of IEEE Std 1353-1994, IEEE Standard for Information Technology—ACSE and Presentation Layer Services—Application Program Interface (API) [C Language Binding], but is included for information only.)

Purpose

This standard defines a C language binding to the language-independent ACSE Presentation Services API defined in IEEE Std 1351-1994 {2}.

Related Standards Activities

There are no related standards activities at this time.¹⁾

Extensions are approved as “ammendments” or “revisions” to this document, following the IEEE Standards procedures.

Approved supplements are published separately and distributed with orders from the IEEE for this document until the full document is reprinted and such supplements are incorporated in their proper positions.

If you have any question about the completeness of your version, you may contact the IEEE Standards Department [+1 (908) 562-3800] to determine what supplements have been published.

If you have an interest in participating in the working groups addressing these issues, please send your name, address, and phone number to the Secretary, IEEE Standards Board, Institute of Electrical and Electronics Engineers, Inc., P.O. Box 1331, 445 Hoes Lane, Piscataway, NJ 08855-1331, and ask to have this forwarded to the chairperson of the appropriate PASC working group.

1) A *Standards Status Report* that lists all current IEEE Computer Society standards projects is available from the IEEE Computer Society, 1730 Massachusetts Avenue NW, Washington, DC 20036-1903, USA; Telephone: +1 202 371-0101; FAX: +1 202 728-9614.

This standard was prepared by the P1238 Working Group, sponsored by the Portable Applications Standards Committee of the IEEE Computer Society. At the time this standard was approved, the membership of the P1238 Working Group was as follows:

Portable Applications Standards Committee (PASC)

Chair: Lowell Johnson
Vice Chairs: Barry Needham
Lorraine Kevra
Andrew Josey
Treasurer: Pete Meier
Secretary: Charles Severance

P1238 Working Group Officials

Chair: Tim Boland
Editor/Secretary: Graham Jack

Working Group

Reed Adams
Dave Cannon

Steve Greismer
Petr Janeček
Ed Kelly

Curtis Royster
Larry Spieler

The following persons were members of the P1238 Balloting Group that approved the standard for submission to the IEEE Standards Board:

Reed Adams
Andy Bihain
Tim Boland
Loren E. Buhle
Malcolm Butler
Steve Case
Philip H. Enslow, Jr.
Steve Griesmer

Chris Harding
Jeffrey S. Haemer
Petr Janecek
Judy S. Kerner
Martin J. Kirk
Thomas M. Kurihara
Roland McGrath
Curtis Royster

Jon F. Spencer
Dennis C. Stewart
Mark Sylvester
Mark-Rene Uchida
Angel Vina
Andrew Walker
Keith S. Weir
Oren Yuen

The IEEE Standards Board approved this standard on September 22, 1994 with the following membership:

Wallace S. Read, *Chair*

Donald C. Loughry, *Vice Chair*

Andrew G. Salem, *Secretary*

Gilles A. Baril
Bruce B. Barrow
Jose A. Berrios de la Paz
Clyde R. Camp
James Costantino
Stephen L. Diamond
Donald C. Fleckenstein
Jay Forster*
Ramiro Garcia

Donald N. Heirman
Richard J. Holleman
Jim Isaak
Ben C. Johnson
Sonny Kasturi
Lorraine C. Kevra
E. G. "Al" Kiener
Ivor N. Knight

Joseph L. Koepfinger*
D. N. "Jim" Logothetis
L. Bruce McClung
Marco W. Migliaro
Mary Lou Padgett
Arthur K. Reilly
Ronald H. Reimer
Gary S. Robinson
Leonard L. Tripp

*Member Emeritus

Also included are the following nonvoting IEEE Standards Board liaisons:

Satish K. Aggarwal
James Beall
Richard B. Engelman
Robert E. Hebner
David E. Soffrin

Rachel A. Meisel
IEEE Standards Project Editor

IEEE Standard for Information Technology— ACSE and Presentation Layer Services— Application Program Interface (API) [C Language Binding]

Section 1: General

1.1 Scope

This standard defines an ISO/IEC 9899 : 1990 {1} C language binding to the language-independent API defined in IEEE Std 1351-1994 {2}. It is intended to support application portability at the source-code level and to be used by application developers, system implementors, test method writers, and users.

1.2 Normative References

The following standards contain provisions that, through references in this text, constitute provisions of this standard. At the time of publication, the editions indicated were valid. All standards are subject to revision, and parties to agreements based on this standard are encouraged to investigate the possibility of applying the most recent editions of the standards listed below.

{1} ISO/IEC 9899 : 1990,¹⁾ *Programming Languages — C*.

¹⁾ ISO/IEC documents can be obtained from the ISO office, 1 Rue de Varembé, Case Postale 56, CH-1211, Genève 20, Switzerland/Suisse.

- 1 {2} IEEE Std 1351-1994,²⁾ IEEE Standard for Information Technology—ACSE
2 and Presentation Layer Services—Application Program Interface (API)
3 [Language Independent].

4 **1.3 Conformance**

5 **1.3.1 Implementation Conformance**

6 **1.3.1.1 Requirements**

7 A conforming implementation for this standard shall be a conforming implementa-
8 tion, as defined in IEEE Std 1351-1994 {2}, for the programming language binding
9 specification that is contained in this standard.

10 **1.3.1.2 Documentation**

11 A conformance document shall be available for an implementation claiming confor-
12 mance to this standard. It shall satisfy the requirements that are defined in IEEE
13 Std 1351-1994 {2} for a conformance document for a programming language bind-
14 ing specification.

15 The phrases “shall document” or “shall be documented” in this standard or in a
16 programming language binding specification for this standard mean that docu-
17 mentation of the feature shall appear in the conformance document, as described
18 previously, unless the system documentation is explicitly mentioned.

19 The system documentation should also contain the information found in the con-
20 formance document.

21 **1.3.1.3 Conforming Implementation Options**

22 There are no implementation options specific to the C Binding. See the language-
23 independent specification in IEEE Std 1351-1994 {2} for a list of the implementa-
24 tion options defined there.

25 **1.3.2 Application Conformance**

26 All applications claiming conformance to a programming language binding
27 specification for this standard shall fall within one of the following categories.

28
29 ²⁾ IEEE documents can be obtained from the Institute of Electrical and Electronics Engineers, 445 Hoes Lane, P.O.
30 Box 1331, Piscataway, NJ, 08855-1331, USA.

1.3.2.1 Strictly Conforming Application

A Strictly Conforming Application is an application that requires only the facilities described in this standard, in the programming language binding specification, and in the applicable language standards. Such an application shall accept any behavior described in this standard or in the programming language binding specification as unspecified or implementation-defined, and for symbolic constants shall accept any value in the ranges permitted by this standard and the programming language binding specification.

1.3.2.2 ISO/IEC Conforming Application

An ISO/IEC Conforming Application of a programming language binding specification for this standard is an application that uses only the facilities described in this standard, in the programming language binding specification, in the applicable language standards, and in other language binding standards approved by ISO or IEC. Such an application shall include a statement of conformance that documents all options and limit dependencies, and all other ISO or IEC standards used.

1.3.2.3 Conforming Application Using Extensions

A Conforming Application Using Extensions of a programming language binding specification for this standard is an application that differs from a Conforming Application only in that it uses nonstandard facilities that are consistent with this standard and with the programming language binding specification. Such an application shall fully document its requirements for these extended facilities, in addition to the documentation required of a Conforming Application.

1.4 Test Methods

No test methods are available for this standard.

Section 2: Terminology and General Requirements

2.1 Conventions

2.1.1 General and Typographic Conventions

When presented in **Synopsis** sections or in extended code fragments, C language source code is printed using fixed width font. Where C language names or code fragments appear in the text of the standard, the following conventions apply:

- A C language function name, function argument name, data reference, or code fragment appears in *italic* font, *ap_cdata_t.sync_p_sn*. A function name is indicated by following parentheses, *ap_rcv()*.
- A C language typedef name appears in **bold** font, **ap_cdata_t**.
- A symbolic constant, other than an error name, appears unadorned in Roman (normal) font capitals, AP_NDELAY.
- A symbolic constant for an error name is surrounded by brackets, [AP_NOTSUPPORTED].

2.1.2 Language-Independent Conventions

When making reference to language-independent objects defined in IEEE Std 1351-1994 {2}, the conventions defined in that standard shall apply.

2.2 Definitions

2.2.1 Terminology

For the purposes of this standard, the following definitions apply:

2.2.1.1 conformance document: A document provided by an implementor that contains implementation details as described in 1.3.1.2.

2.2.1.2 implementation defined: An indication that the implementation shall define and document the requirements for correct program constructs and correct data of a value or behavior [ISO/IEC 9945-1 : 1990 {B1}].

2.2.1.3 may: An indication of an optional feature [ISO/IEC 9945-1 : 1990 {B1}].

With respect to implementations, the word *may* is to be interpreted as an optional feature that is not required in this standard, but can be provided. With respect to Strictly Conforming Applications, the word *may* means that the optional feature shall not be used.

2.2.1.4 shall: An indication of a requirement on the implementation or on Strictly Conforming Applications, where appropriate [ISO/IEC 9945-1 : 1990 {B1}].

2.2.1.5 should:

- (1) With respect to implementations, an indication of an implementation recommendation, but not a requirement [ISO/IEC 9945-1 : 1990 {B1}].
- (2) With respect to applications, an indication of a recommended programming practice for applications and a requirement for Strictly Conforming Applications [ISO/IEC 9945-1 : 1990 {B1}].

2.2.1.6 supported: A condition regarding optional functionality.

Certain functionality in this standard is optional, but the interfaces to that functionality are always required. If the functionality is *supported*, the interfaces work as specified by this standard (except that they do not return the error condition indicated for the unsupported case). If the functionality is not *supported*, the interface shall always return the indication specified for this situation [ISO/IEC 9945-1 : 1990 {B1}].

2.2.1.7 system documentation: All documentation provided with an implementation, except the conformance document [ISO/IEC 9945-1 : 1990 {B1}].

Electronically distributed documents for an implementation are considered part of the system documentation.

2.2.1.8 undefined: An indication that this standard imposes no portability requirements on the use by an application of an indeterminate value or its behavior with erroneous program constructs or erroneous data [ISO/IEC 9945-1 : 1990 {B1}].

Implementations (or other standards) may specify the result of using that value or causing that behavior. An application using such behaviors is using extensions, as defined in 1.3.2.3.

2.2.1.9 unspecified: An indication that this standard imposes no portability requirements on applications for correct program constructs or correct data regarding a value or behavior [ISO/IEC 9945-1 : 1990 {B1}].

Implementations (or other standards) may specify the result of using that value or causing that behavior. An application requiring a specific behavior, rather than tolerating any behavior when using that functionality, is using extensions, as defined in 1.3.2.3.

2.2.2 General Terms

For the purposes of this standard, the terms defined in IEEE Std 1351-1994 {2} shall apply.

2.2.3 Abbreviations

For the purposes of this standard, the abbreviations defined in IEEE Std 1351-1994 {2} shall apply. The following additional abbreviations are used in this standard.

2.2.3.1 C Standard: ISO/IEC 9899 : 1990 {1}.

Section 3: Overview (Informative)

NOTE: This section provides additional overview material for the C Binding to the APS API and is not Normative. It does not contain requirements for conforming implementations.

3.1 Functions and Mechanisms

3.1.1 User Data Buffering

Most of the primitives supported by the service use the *user data* parameter of the *ap_rcv()* or *ap_snd()* functions to send or receive user data. The amount of data associated with a primitive varies from a few octets for a P_XDATA primitive, to an unlimited amount for a P_DATA primitive. In order to provide a consistent interface, a single mechanism has been designed to cater for both extremes. The mechanism separates the functions of buffer allocation and use by defining two structures: **ap_osi_dbuf_t** is a structure that controls an allocation of buffer space, containing a pointer to the start and end of the buffer (*db_base* and *db_limit*) and a reference count (*db_ref*); **ap_osi_vbuf_t** is a structure that controls the use of space, containing a pointer to the **ap_osi_dbuf_t** structure that controls a buffer (*b_datap*), and read and write pointers (*b_rptr* and *b_wptr*), which indicate the portion of that buffer reserved by this structure. It also contains a *b_cont* element, which can be used to chain a series of **ap_osi_vbuf_t** structures together.

This mechanism allows data to be passed between the service and the client in one or more discrete buffers, using start and end pointers to identify the portion(s) of each buffer that contains valid data. The client may elect to provide allocation functions for the buffers that the service uses to receive data.

Figure 3-1 shows the **ap_osi_vbuf_t** and **ap_osi_dbuf_t** structures. Figure 3-2 shows a more sophisticated example of the use of the buffering structures, with two separate segments of data passed in a single buffer.

For data being sent, the service user is responsible for freeing buffers once the *ap_snd()* function has completed. For data received, the service allocates the necessary buffers, possibly using the user allocation functions. Again the client shall free these buffers once the data contained in them has been processed.

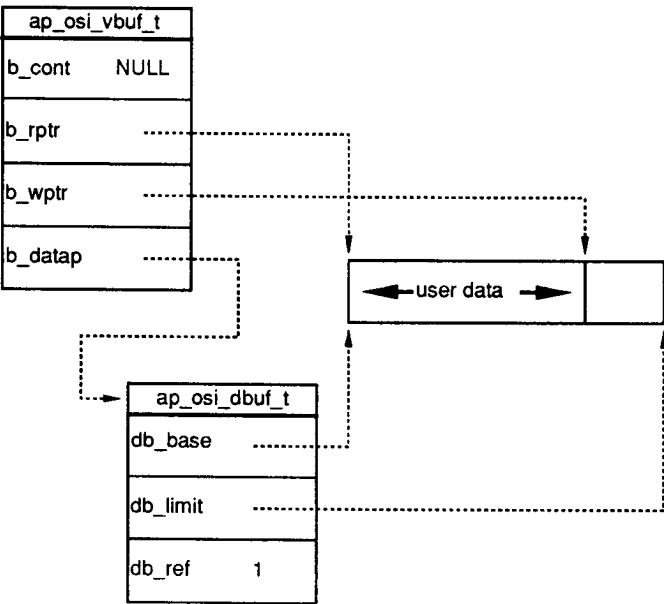


Figure 3-1 – Basic Buffer Structures

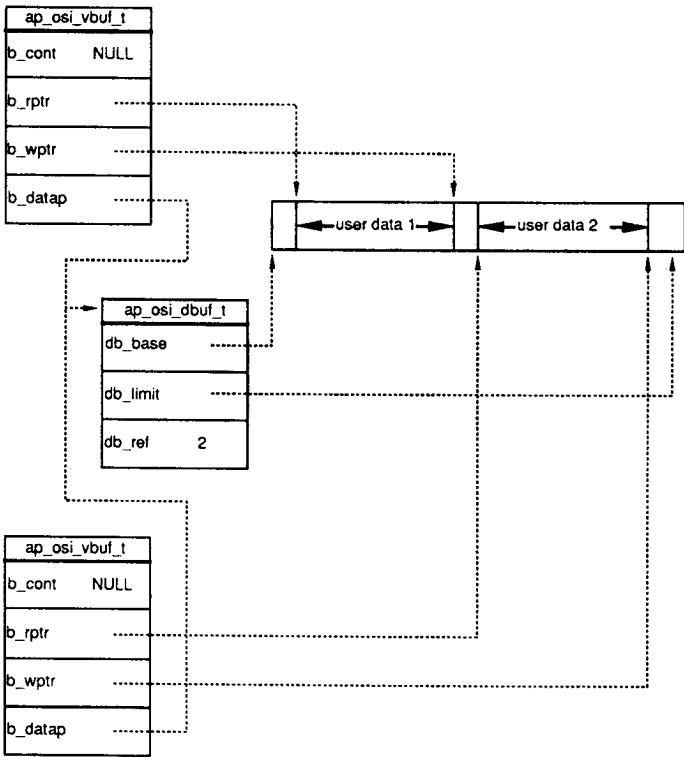


Figure 3-2 – Multiple Reference Buffer Example

3.1.2 Error Reporting

Most operations return a result code as the operation value. This code is zero if the operation was completely successful, and -1 if any sort of error or warning condition occurred. In addition, each function includes an error code return location, *aperrno_p*, as its final argument. The client shall pass a pointer to a location into which an error code is returned if the result of the operation is -1 . The value in this location is unchanged if the operation result is zero (that is, "SUCCESS").

Section 4: APS Environment

The C Binding for the ACSE/Presentation API supports the environment defined in IEEE Std 1351-1994 {2}. The C Binding for the datatypes that represent the environment attributes are defined in section 5. This section defines the C Binding-specific aspects of the APS environment.

4.1 Environment Attribute Identifiers/Types

Table 4.1 lists the *#defines* and C language types used to identify attributes of the APS Environment and to get or set their values. The identifiers are derived from the environment attribute names defined in IEEE Std 1351-1994 {2} by adding the prefix *AP_*.

An implementation conforming to this standard shall define each of the identifiers in this list as a unique bit mask.

NOTE: The values are defined as bit masks for use in the C Binding-specific *mask* field of the **ap_a_assoc_env_t** typedef. In addition, they may be passed as the *kind* argument to the *ap_free()* function to free the storage associated with substructures.

Table 4-1 – Environment Attribute Identifiers/Types

Identifier	C Type
AP_ACSE_AVAIL	unsigned long
AP_ACSE_SEL	unsigned long
AP_AFU_AVAIL	unsigned long
AP_AFU_SEL	unsigned long
AP_BIND_PADDR	ap_paddr_t
AP_CLD_AEID	ap_aei_api_id_t
AP_CLD_AEQ	ap_aeq_t
AP_CLD_APIID	ap_aei_api_id_t
AP_CLD_APT	ap_apr_t
AP_CLD_CONN_ID	ap_conn_id_t
AP_CLG_AEID	ap_aei_api_id_t
AP_CLG_AEQ	ap_aeq_t
AP_CLG_APIID	ap_aei_api_id_t
AP_CLG_APT	ap_apr_t
AP_CLG_CONN_ID	ap_conn_id_t
AP_CNTX_NAME	ap_objid_t
AP_COPYENV	unsigned long
AP_DCS	ap_dcs_t
AP_DIAGNOSTIC	ap_diag_t
AP_DPCN	ap_dcn_t
AP_DPCR	unsigned long
AP_FLAGS	unsigned long
AP_INIT_SYNC_PT	unsigned long
AP_INIT_TOKENS	unsigned long
AP_LCL_PADDR	ap_paddr_t
AP_LIB_AVAIL	unsigned long
AP_LIB_SEL	unsigned long
AP_MODE_AVAIL	unsigned long
AP_MODE_SEL	unsigned long
AP_MSTATE	unsigned long

Table 4.1 – Environment Attribute Identifiers/Types (*concluded*)

Identifier	C Type
AP_OLD_CONN_ID	ap_old_conn_id_t
AP_OPT_AVAIL	unsigned long
AP_PCDL	ap_cdl_t
AP_PCDRL	ap_cdrl_t
AP_PFU_AVAIL	unsigned long
AP_PFU_SEL	unsigned long
AP_PRES_AVAIL	unsigned long
AP_PRES_SEL	unsigned long
AP_QLEN	unsigned long
AP_QOS	ap_qos_t
AP_REM_PADDR	ap_paddr_t
AP_ROLE_ALLOWED	unsigned long
AP_ROLE_CURRENT	unsigned long
AP_RSP_AEID	ap_aei_api_id_t
AP_RSP_AEQ	ap_aeq_t
AP_RSP_APIID	ap_aei_api_id_t
AP_RSP_APT	ap_apt_t
AP_SESS_AVAIL	unsigned long
AP_SESS_SEL	unsigned long
AP_SESS_OPT_AVAIL	unsigned long
AP_SFU_AVAIL	unsigned long
AP_SFU_SEL	unsigned long
AP_STATE	unsigned long
AP_TOKENS_AVAIL	unsigned long
AP_TOKENS_OWNED	unsigned long

Section 5: Service Interface

5.1 Introduction

This section defines the C Binding to the service interface.

5.2 C Type Definitions

This clause provides the C Bindings for the language-independent datatypes defined in section 5.2 of IEEE Std 1351-1994 (2).

In general, the name of the C datatype is derived from the associated LI datatype name by adding the prefix “ap_” and converting the “_type” suffix to “_t”. Thus the LI datatype `aeq_type` becomes the C typedef `ap_aeq_t`. Where this conversion does not apply, the definition indicates the associated LI datatype(s).

There are some LI datatypes for which there are not equivalent C language typedefs. These datatypes are listed in 5.2.19, where their C Binding representation is defined.

5.2.1 ap_aei_api_id_t

This structure implements the LI datatype `aei_api_id_type`.

5.2.1.1 C Declaration

```
typedef struct {  
    int size;                /* buffer size in bytes */  
    unsigned char *udata;    /* buffer with user encoded AE-identifier */  
                             /* or AP-identifier */  
} ap_aei_api_id_t;
```

5.2.2 ap_aeq_t

This structure implements the LI datatype `aeq_type`.