Committee Draft ISO/IEC CD	
Date: **2004-08-06**	Reference number: ISO/JTC 1/SC 32**N1165**
Supersedes document SC 32Nxxx	

THIS DOCUMENT IS STILL UNDER STUDY AND SUBJECT TO CHANGE. IT SHOULD NOT BE USED FOR REFERENCE PURPOSES.

ISO/IEC JTC 1/SC 32 Data Management and Interchange Secretariat: USA (ANSI)	Circulated to P- and O-members, and to technical committees and organizations in liaison for voting (P-members only) by: **2004-11-07** Please return all votes and comments in electronic form directly to the SC 32 Secretariat by the due date indicated.

ISO/IEC CD 20944-40:200x(E)

Title: Information technology — Metadata Interoperability & Bindings (MDIB) Part 40: Common Provisions for API binding

Project: 1.32.17.01.40.00

Introductory note: The attached document is hereby submitted for a three-month letter ballot to the National Bodies of ISO/IEC JTC 1/SC 32. The ballot starts 2004-08-06.

Medium: E

No. of pages: 29

Address Reply to: Douglas Mann, Secretariat, ISO/IEC JTC 1/SC 32, Pacific Northwest National Laboratory, 13667 Legacy Circle Apt H, Herndon, VA, 20171, United States of America

Telephone: +1 202-566-2126; Facsimile; +1 202-566-1639; E-mail: MannD@battelle.org

© ISO 2004 – All rights reserved

Reference number of working document: **ISO/IEC JTC1 SC32 N1165**

Date: 2004-08-04

Reference number of document: **ISO/IEC CD1 20944-40**
[Release Sequence #7]

Committee identification: ISO/IEC JTC1 SC32 WG2

SC32 Secretariat: US

Information technology — Metadata Interoperability and Bindings (MDIB) — Part 40: Common provisions for API bindings

Warning

This document is not an ISO International Standard. It is distributed for review and comment. It is subject to change without notice and may not be referred to as an International Standard.

Recipients of this draft are invited to submit, with their comments, notification of any relevant patent rights of which they are aware and to provide supporting documentation.

Document type: International standard
Document subtype: if applicable
Document stage: (30) Committee
Document language: E

Copyright notice

This ISO document is a working draft or committee draft and is copyright-protected by ISO. While the reproduction of working drafts or committee drafts in any form for use by participants in the ISO standards development process is permitted without prior permission from ISO, neither this document nor any extract from it may be reproduced, stored or transmitted in any form for any other purpose without prior written permission from ISO.

Requests for permission to reproduce this document for the purpose of selling it should be addressed as shown below or to ISO's member body in the country of the requester:

ISO copyright office
Case postale 56
CH-1211 Geneva 20
Tel. +41 22 749 01 11
Fax +41 22 749 09 47
E-mail copyright@iso.org
Web www.iso.org

Reproduction for sales purposes may be subject to royalty payments or a licensing agreement.

Violators may be prosecuted.

Contents

Foreword

ISO (the International Organization for Standardization) is a worldwide federation of national standards bodies (ISO member bodies). The work of preparing International Standards is normally carried out through ISO technical committees. Each member body interested in a subject for which a technical committee has been established has the right to be represented on that committee. International organizations, governmental and non-governmental, in liaison with ISO, also take part in the work. ISO collaborates closely with the International Electrotechnical Commission (IEC) on all matters of electrotechnical standardization.

International Standards are drafted in accordance with the rules given in the ISO/IEC Directives, Part 2.

The main task of technical committees is to prepare International Standards. Draft International Standards adopted by the technical committees are circulated to the member bodies for voting. Publication as an International Standard requires approval by at least 75 % of the member bodies casting a vote.

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO shall not be held responsible for identifying any or all such patent rights.

ISO/IEC 20944-40 was prepared by Technical Committee ISO/IEC JTC1, *Information Technology*, Subcommittee SC32, *Data Management and Interchange*.

ISO/IEC 20944 consists of the following parts, under the general title *Information technology — Metadata Interoperability and Bindings (MDIB)*:

— *Part 01: Framework*

— *Part 02: Common vocabulary*

— *Part 03: Common provisions for conformance*

— *Part 04: Generic usage*

— *Part 05: Common data structures and services*

— *Part 06: Semi-structured aggregation*

— *Part 20: Common provisions for coding bindings*

— *Part 21: XML coding binding*

— *Part 22: DIVP coding binding*

— *Part 23: ASN.1 coding binding*

— *Part 40: Common provisions for application programming interface (API) bindings*

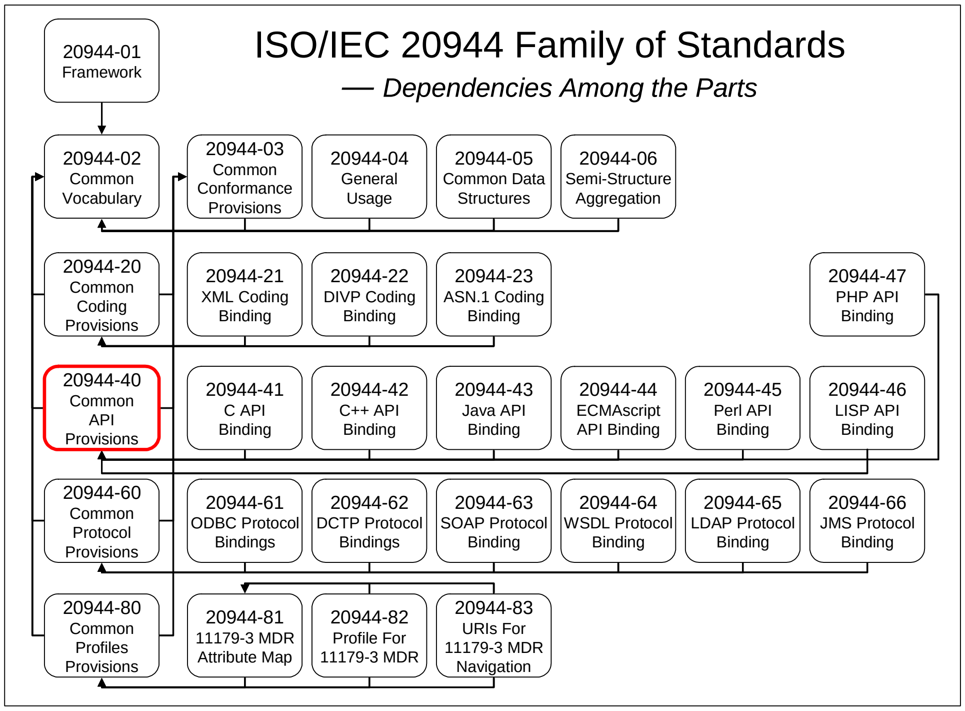
— *Part 41: C API binding*

© ISO 2004 – All rights reserved

- *Part 42: C++ API binding*
- *Part 43: Java API binding*
- *Part 44: ECMAscript API binding*
- *Part 45: Perl binding*
- *Part 46: LISP binding*
- *Part 47: PHP binding*
- *Part 60: Common provisions for protocol bindings*
- *Part 61: ODBC protocol binding*
- *Part 62: DCTP protocol binding*
- *Part 63: SOAP protocol binding*
- *Part 64: WSDL protocol binding*
- *Part 65: LDAP protocol binding*
- *Part 66: JMS protocol binding*
- *Part 80: Common provisions for profiles*
- *Part 81: Attribute mapping for 11179-3 metadata registry metamodel*
- *Part 82: Profile for 11179-3 metadata registry metamodel*
- *Part 83: Uniform Resource Identifier (URI) suffixes for 11179-3 metadata registry metamodel navigation*

Introduction

The following diagram shows the organization of the ISO/IEC 20944 family of standards with this Part highlighted.


ISO/IEC 20944 Family of Standards
— Dependencies Among the Parts
20944-01 Framework
20944-02 Common Vocabulary
20944-03 Common Conformance Provisions
20944-04 General Usage
20944-05 Common Data Structures
20944-06 Semi-Structure Aggregation
20944-20 Common Coding Provisions
20944-21 XML Coding Binding
20944-22 DIVP Coding Binding
20944-23 ASN.1 Coding Binding
20944-47 PHP API Binding
20944-40 Common API Provisions
20944-41 C API Binding
20944-42 C++ API Binding
20944-43 Java API Binding
20944-44 ECMAscript API Binding
20944-45 Perl API Binding
20944-46 LISP API Binding
20944-60 Common Protocol Provisions
20944-61 ODBC Protocol Bindings
20944-62 DCTP Protocol Bindings
20944-63 SOAP Protocol Binding
20944-64 WSDL Protocol Binding
20944-65 LDAP Protocol Binding
20944-66 JMS Protocol Binding
20944-80 Common Profiles Provisions
20944-81 11179-3 MDR Attribute Map
20944-82 Profile For 11179-3 MDR
20944-83 URIs For 11179-3 MDR Navigation

Organization of ISO/IEC 20944 family of standards.

This Part of ISO/IEC 20944 concerns provisions that are common to the API bindings, i.e., Parts 40 to 59. For example, common features include:

— using a session paradigm to access data
— using a parameterized security framework to support a variety of security techniques
— using a hierarchical navigation for data access

This Part is intended to be normatively referenced by

— the individual API bindings (i.e., Parts 41 to 59)
— application-specific data models, e.g., using 20944 for their coding, API, and protocol bindings
— coding, API, and protocol bindings for data (metadata) interchange for ISO/IEC 11179-3

© ISO 2004 – All rights reserved

Information technology — Metadata Interoperability and Bindings (MDIB) — Part 40: Common provisions for API bindings

Editor's Note: Each part of 20944 is marked with a common sequence number ("**[Release Sequence #N]**") to indicate they are synchronized and harmonized among themselves. The mark "**[Release Sequence #N]**" does *not* imply that there are a complete set of N-1 prior drafts for any particular Part.

1 Scope

This Part of this International Standard specifies provisions that are common across coding bindings for the 20944 family of standards. This Part is incorporated, via normative reference, into the individual API bindings.

2 Normative references

The following referenced documents are indispensable for the application 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 11404:—[1], *Information technology — General Purpose Datatypes (GPD)*

ISO/IEC 13886:1995[2], *Information technology — Language-Independent Procedure Calling (LIPC)*

ISO/IEC 20944-02:—[3], *Information technology — Metadata Interoperability and Bindings (MDIB) — Common vocabulary*[4]

3 Terms and definitions

For the purposes of this document, the following terms and definitions given in Part 02 and the following apply[5].

[1] To be published, under revision (prior edition: 1996). For current status, see "http://ra.dkuug.dk/jtc1/sc22/wg11".

[2] ISO/IEC 13886 is freely available, see "http://www.jtc1.org" under .

[3] To be published.

[4] The international standards, technical reports, and drafts of the 11179, 20943, and 20944 series are available at

http://metadata-standards.org/11179
http://metadata-standards.org/20943
http://metadata-standards.org/20944

3.1
referenced data interchange specification
data model that is being used for a defined interoperability binding

NOTE The term *referenced data interchange specification*, defined in 20944-02, is used throughout the 20944 family of standards to reference the data model that is being used for the bindings. The *referenced data interchange specification* is tied to the bindings via normative reference, e.g., some other standard defines a data model and uses 20944, via normative reference, to provide some coding, API, or protocol bindings. For Part 82, the *referenced data interchange specification* refers to the 11179-3 metamodel. Part 04 of this International Standard, explains how other standards and specifications may use or re-use portions of the 20944 family of standards.

4 Functional capabilities

The purpose of a common API provisions is to provide a common and consistent description of services across all API bindings.

5 Conceptual model

The API bindings have commonality in their conceptualization of data access services. For example, common features include:

— using a session paradigm to access data
— using a parameterized security framework to support a variety of security techniques
— using a hierarchical navigation for data access

The individual API bindings (Parts 041 to 059) each incorporate a mapping of common data access services to their individual binding requirements.

6 Services

6.1 Session establishment services

The following messages start up and shut down sessions with the metadata registries.

6.1.1 Connect

Synopsis

```
mdib_connect:
procedure
(
    in target: string, // repository to connect to
    in options: string, // connect options
)
returns mdib_handle,
```

5 Users and implementers of this International Standard may find it useful to reference additional terms and definitions from 20944-02.

© ISO 2004 – All rights reserved

Description

Creates a new session to a metadata repository as named by target. The options parameter is a whitespace-separated list of name-value pairs that describe an implementation-defined set of connection options. If successful, returns a session handle to the repository, but does not request access (see mdib_open). If not successful, returns a null session handle.

Example

```
// C/C++ illustration
mdib_handle sh; // session handle
mdib_handle ch; // child session handle
sh = mdib_connect("dctp://xyz.com/repository_2",
    "option_x=p option_y=q option_z");
if ( sh != NULL )
{
    ch = mdib_open(sh,"postal_address","read-only");
    // ...
    mdib_close(ch);
    // ...
    mdib_disconnect(sh);
}
```

6.1.2 Disconnect

Synopsis

```
mdib_disconnect:
procedure
(
    in session: mdib_handle // session handle
)
returns (state(success,failure)),
```

Description

Closes and disconnects a session to a metadata repository associated with the handle session and all of its child sessions. Returns success if successful, failure if unsuccessful.

Example

```
// C/C++ illustration
mdib_handle sh; // session handle
sh = mdib_connect("dctp://xyz.com/repository_2",
    "option_x=p option_y=q option_z");
// ... open session threads
// ... access metadata
// ... close session threads
mdib_disconnect(sh);
```

6.1.3 Open

Synopsis

```
mdib_open:
procedure
(
    in session: mdib_handle, // session handle
    in node: characterstring, // portion of repository to open
    in options: characterstring, // connect options
)
returns (mdib_handle),
```

Description

Opens a child session within a session to metadata repository, as pointed to by the handle session.

The node parameter is the name of a portion of the repository — a view. If node is the empty string (""), then the child session is a duplicate session of the parent session. The partitioning and naming of contents of repositories is outside the scope of this Standard.

The options parameter is a whitespace-separated list of name-value pairs that describe a set of options from the following list:

— "read-only" (or "ro" or "r"): The child session is opened with read-only access.
— "read-write" (or "rw"): The child session is opened with read-write access.
— "read-seq" (or "rs"): The child session is opened with read sequential access. This option may be followed by the sub-option "ordering=xxx" where "xxx" may be "breadth", "depth", "alphasort", or "datesort".
— "write-seq" (or "ws"): The child session is opened with write sequential access.
— "append" (or "a"): The child session is opened for write-append access.
— "inherit" (or ""): The child session inherits the characteristics of the parent session.

If mdib_open is successful, it returns a handle to the child session. If not successful, it returns a null handle.

Example

```
// C/C++ illustration
mdib_handle sh; // session handle
mdib_handle ch; // child session handle
sh = mdib_connect("dctp://xyz.com/repository_2",
    "option_x=p option_y=q option_z");
if ( sh != NULL )
{
    ch = mdib_open(sh,"postal_address","read-only");
    // ...
    mdib_close(ch);
    // ...
    mdib_disconnect(sh);
}
```

6.1.4 Close

Synopsis

© ISO 2004 – All rights reserved

```
mdib_close:
procedure
(
    in mdib_handle: session, // session handle
)
returns (state(success,failure)),
```

Description

Closes a session associated with the handle session and all of its child sessions. Returns success if successful, failure if unsuccessful.

Example

```
// C/C++ illustration
mdib_handle sh; // session handle
mdib_handle ch; // child session handle
sh = mdib_connect("dctp://xyz.com/repository_2",
    "option_x=p option_y=q option_z");
if ( sh != NULL )
{
    ch = mdib_open(sh,"postal_address","read-only");
    // ...
    mdib_close(ch);
    // ...
    mdib_disconnect(sh);
}
```

6.2 Session parameter services

The following services may be used to modify and retrieve the parameters of the session.

6.2.1 Get path

Synopsis

```
mdib_get_path:
procedure
(
    in session: mdib_handle, // session handle
)
returns (characterstring),
```

Description

Retrieves the current default node path.

If mdib_get_path is successful, it returns a string containing the default node path; otherwise, error return is indicated by a return of a null pointer (not an empty string).

Example

```
// C/C++ illustration
mdib_handle sh; // session handle
mdib_handle ch; // child session handle
sh = mdib_connect("dctp://xyz.com/repository_2",
    "option_x=p option_y=q option_z");
if ( sh != NULL )
{
    // open "postal_address"
    ch = mdib_open(sh,"postal_address","read-only");
    // change path to "city"
    mdib_put_path(ch,"city");
    // retrieve path, should be "city"
    new_path = mdib_get_path(ch)
    if ( new_path != "city" )
    {
        // error
    }
}
```

6.2.2 Put path

Synopsis

```
mdib_put_path:
procedure
(
    in session: mdib_handle, // session handle
    in node: characterstring, // portion of repository
)
returns (state(success,failure)),
```

Description

Changes the current default node path to the path specified by node. If node is a relative path, then the new path is relative to the previous default node path.

If mdib_put_path is successful, it returns success, otherwise error return is indicated by a return of failure.

Example

```
// C/C++ illustration
mdib_handle sh; // session handle
mdib_handle ch; // child session handle
sh = mdib_connect("dctp://xyz.com/repository_2",
    "option_x=p option_y=q option_z");
if ( sh != NULL )
{
    // open "postal_address"
    ch = mdib_open(sh,"postal_address","read-only");
    // change path to "city"
    if ( mdib_put_path(ch,"city") == -1 )
    {
        // error
    }
    // success
}
```

© ISO 2004 – All rights reserved

6.3 Security services

The following services are supported.

6.3.1 Request Authorization/Authentication

Synopsis

```
mdib_request_auth:
procedure
(
    in session: mdib_handle, // session handle
    in auth_type: characterstring, // auth type
    in auth_options: characterstring, // auth options
)
returns (state(success,failure)),
```

Description

Requests the repository to supply authorization and/or authentication credentials, as pointed to by the handle session.

The auth_type parameter is a whitespace-separated list of name-value pairs that describe a set of credentials that are requested from the repository:

- "symmetric": The authorization and/or authentication type is symmetric, i.e., both sides agree on the same "word", such as a password. The auth_options parameter specifies a list of words to request as credentials.
- "asymmetric": The authorization and/or authentication type is asymmetric, i.e., both sides agree on a separate set of "words", such a public key techniques.
- "challenge": Requests a response to the security challenge..
- "identifier": Requests the repository supply a list of identifiers for authentication.
- "operation": Requests the repository supply a list of operations to authorize.
- "nomad": Requests agreement and authorization of a nomadic connection.

The auth_options parameter is a whitespace-separated list of name-value pairs that describe a set of authorization and/or authentication options from the following list:

- "password": The password is requested from the repository.
- "key": The key is requested from the repository for asymmetric access.
- "identifier": The identifier is requested from the repository.
- "operation": The operation is requested from the repository.
- "nomad_request_id": The identifier associated with the requestor of the nomadic connection.
- "nomad_response_id": The identifier associated with the respondent of the nomadic connection.
- "nomad_timeout": The timeout associated with the nomadic connection.

If mdib_request_auth is successful, it returns success, otherwise error return is indicated by a return of failure.

Example

```
// C/C++ illustration
mdib_handle sh; // session handle
mdib_handle ch; // child session handle
sh = mdib_connect("dctp://xyz.com/repository_2",
    "option_x=p option_y=q option_z");
if ( mdib_request_auth(sh, "symmetric", "password") !=
    "swordfish" )
{
    // ... error, abort because of password mismatch
    return;
}
if ( sh != NULL )
{
    ch = mdib_open(sh,"postal_address","read-only");
    // ...
    mdib_close(ch);
    // ...
    mdib_disconnect(sh);
}
```

6.3.2 Response Authorization/Authentication

Synopsis

```
mdib_response_auth:
procedure
(
    in session: mdib_handle, // session handle
    in auth_type: string, // auth type
    auth_handler(): procedure, // auth handler function
)
returns state(success,failure),
```

Description

Registers a handler for authorization and/or authentication responses, as requested by the repository pointed to by the handle session.

The auth_type parameter is a whitespace-separated list of name-value pairs that describe a set of credentials that are requested from the repository. See 8.3.1 above for a list of auth_type parameters.

The auth_handler parameter a pointer to a handler service. The handler service is called for each authorization and/or authentication request that matches the type auth_type. The auth_handler service is called with at least three parameters: the session handle, the actual auth_type, the auth_option, and zero or more parameters.

If mdib_response_auth is successful, it returns success, otherwise error return is indicated by a return of failure.

Example

```
// C/C++ illustration
mdib_handle sh; // session handle
mdib_handle ch; // child session handle
sh = mdib_connect("dctp://xyz.com/repository_2",
    "option_x=p option_y=q option_z");
```

© ISO 2004 – All rights reserved

```
mdib_response_auth(sh,"symmetric",handler);
if ( sh != NULL )
{
    ch = mdib_open(sh,"postal_address","read-only");
    // ...
    mdib_close(ch);
    // ...
    mdib_disconnect(sh);
}

// auth handler: returns a password when requested
string handler
(
    mdib_handle sh,
    string auth_type,
    string auth_options,
    va_list other_parameters
)
{
    if ( auth_type == "symmetric" &&
        auth_options == "password" )
    {
        return "swordfish";
    }
}
```

6.4 Data transfer services

The following data transfer are supported.

6.4.1 Get value

Synopsis

```
mdib_get_value:
procedure
(
    in session: mdib_handle, // session handle
    in src_identifier: characterstring, // src object name
    in dst_label_type: characterstring, // saved label: typeof
    out dst_label_ptr: pointer, // saved label: ptr to
    in dst_type_type: characterstring, // saved type: typeof
    out dst_type_ptr: pointer, // saved type: ptr to
    in dst_object_type: characterstring, // saved value: typeof
    out dst_object_ptr: pointer, // saved value: ptr to
    in dst_proplist_type: characterstring, // saved proplist: typeof
    out dst_proplist_ptr: pointer, // saved proplist: ptr to
)
returns (state(success,failure)),
```

Description

Gets a value converted to a particular type, as identified by src_identifier.

Example

```
// C/C++ illustration
mdib_handle sh; // session handle
mdib_handle ch; // child session handle
mdib_type_type value_type; // type information
string city_name;
value_type = mdib_make_type(city_name); // target type
sh = mdib_connect("dctp://xyz.com/repository_2",
    "option_x=p option_y=q option_z");
if ( sh != NULL )
{
    // open "postal_address"
    ch = mdib_open(sh,"postal_address","read-only");
    city_name = mdib_get_value_as_str8(ch,"city",
        NULL, NULL, // ignore label info
        NULL, NULL, // ignore type info
        value_type, city_name, // save value info
        NULL, NULL // ignore property info
        );
}
```

6.4.2 Typed get value

Synopsis

```
mdib_get_value_as_str8:
procedure
(
    session: mdib_handle, // session handle
    src_identifier characterstring, // src object name
)
returns (str8)
raises (bad_conversion),

mdib_get_value_as_str16:
procedure
(
    session: mdib_handle, // session handle
    src_identifier characterstring, // src object name
)
returns (str16)
raises (bad_conversion),

mdib_get_value_as_str32:
procedure
(
    session: mdib_handle, // session handle
    src_identifier characterstring, // src object name
)
returns (str32)
raises (bad_conversion),

mdib_get_value_as_int8:
```

© ISO 2004 – All rights reserved

```
procedure
(
    session: mdib_handle, // session handle
    src_identifier characterstring, // src object name
)
returns (int8)
raises (bad_conversion),

mdib_get_value_as_uint8:
procedure
(
    session: mdib_handle, // session handle
    src_identifier characterstring, // src object name
)
returns (unit8)
raises (bad_conversion),

mdib_get_value_as_int16:
procedure
(
    session: mdib_handle, // session handle
    src_identifier characterstring, // src object name
)
returns (int16)
raises (bad_conversion),

mdib_get_value_as_uint16:
procedure
(
    session: mdib_handle, // session handle
    src_identifier characterstring, // src object name
)
returns (uint16)
raises (bad_conversion),

mdib_get_value_as_int32:
procedure
(
    session: mdib_handle, // session handle
    src_identifier characterstring, // src object name
)
returns (int32)
raises (bad_conversion),

mdib_get_value_as_uint32:
procedure
(
    session: mdib_handle, // session handle
    src_identifier characterstring, // src object name
)
returns (uint32)
raises (bad_conversion),

mdib_get_value_as_int64:
procedure
```

```
(
    session: mdib_handle, // session handle
    src_identifier characterstring, // src object name
)
returns (int64)
raises (bad_conversion),

mdib_get_value_as_uint64:
procedure
(
    session: mdib_handle, // session handle
    src_identifier characterstring, // src object name
)
returns (uint64)
raises (bad_conversion),

mdib_get_value_as_int128:
procedure
(
    session: mdib_handle, // session handle
    src_identifier characterstring, // src object name
)
returns (int128)
raises (bad_conversion),

mdib_get_value_as_uint128:
procedure
(
    session: mdib_handle, // session handle
    src_identifier characterstring, // src object name
)
returns (uint128)
raises (bad_conversion),

mdib_get_value_as_real32:
procedure
(
    session: mdib_handle, // session handle
    src_identifier characterstring, // src object name
)
returns (real32)
raises (bad_conversion),

mdib_get_value_as_real64:
procedure
(
    session: mdib_handle, // session handle
    src_identifier characterstring, // src object name
)
returns (real64)
raises (bad_conversion),

mdib_get_value_as_real80:
procedure
(
```

© ISO 2004 – All rights reserved

```
        session: mdib_handle, // session handle
        src_identifier characterstring, // src object name
    )
    returns (real80)
    raises (bad_conversion),
```

Description

Gets a value converted to a particular type, as identified by src_identifier. The following types are supported:

- str8: A string of "8-bit* characters, i.e., characters with a repertoire of ISO/IEC 8859-1.
- str16: A string of "16-bit* characters, i.e., characters with a repertoire of ISO/IEC 10646-1 Basic Multilingual Plane (BMP).
- str32: A string of "32-bit* characters, i.e., characters with a repertoire of ISO/IEC 10646-1.
- int8: A signed, 2's complement integer of 8 bits, as described by IEEE 1596.5.
- uint8: An unsigned integer of 8 bits, as described by IEEE 1596.5.
- int16: A signed, 2's complement integer of 16 bits, as described by IEEE 1596.5.
- uint16: An unsigned integer of 16 bits, as described by IEEE 1596.5.
- int32: A signed, 2's complement integer of 32 bits, as described by IEEE 1596.5.
- uint32: An unsigned integer of 32 bits, as described by IEEE 1596.5.
- int64: A signed, 2's complement integer of 64 bits, as described by IEEE 1596.5.
- uint64: An unsigned integer of 64 bits, as described by IEEE 1596.5.
- int128: A signed, 2's complement integer of 128 bits, as described by IEEE 1596.5.
- uint128: An unsigned integer of 128 bits, as described by IEEE 1596.5.
- real32: An IEC 60599, 32-bit floating point number.
- real64: An IEC 60599, 64-bit floating point number.
- real80: An IEC 60599, 80-bit floating point number.

If mdib_get_value_* is successful, it returns the value associated with src_identifier; otherwise, error returns are indicated by null pointers for strings, -1 for signed integers, 0 for unsigned integers, and NaN (not a number) for reals.

Example

```
    // C/C++ illustration
    mdib_handle sh; // session handle
    mdib_handle ch; // child session handle
    string city_name; // city name in postal address
    sh = mdib_connect("dctp://xyz.com/repository_2",
        "option_x=p option_y=q option_z");
    if ( sh != NULL )
    {
        // open "postal_address"
        ch = mdib_open(sh,"postal_address","read-only");
        city_name = mdib_get_value_as_str8(ch,"city")
    }
```

6.4.3 Put value

Synopsis

```
    integer mdib_put_value
    (
        in session: mdib_handle, // session handle
        in src_identifier: characterstring, // src object name
        in src_label_type: characterstring, // assigned label: typeof
        in object src_label_ptr, // assigned label: ptr to
        in src_type_type: characterstring, // assigned type: typeof
        in object src_type_ptr, // assigned type: ptr to
        in src_object_type: characterstring, // assigned value: typeof
        in object src_object_ptr, // assigned value: ptr to
        in src_proplist_type: characterstring, // assigned proplist: typeof
        in object src_proplist_ptr, // assigned proplist: ptr to
    )
```

Description

Puts a value in an object named src_identifier. The label is specified by the parameter type src_label_type and by a pointer to the parameter value src_label_ptr. The object type is specified by the parameter type src_type_type and by a pointer to the parameter value src_type_ptr. The object value is specified by the parameter type src_object_type and by a pointer to the parameter value src_object_ptr. The property list is specified by the parameter type src_proplist_type and by a pointer to the parameter value src_proplist_ptr. The parameter value may by MDIB_PARAMETER_STRING that specifies the next parameter is a string type.

Example

```
    // C/C++ illustration
    mdib_handle sh; // session handle
    mdib_handle ch; // child session handle
    mdib_type_type value_type; // type information
    string city_name;
    value_type = mdib_make_type(city_name); // target type
    sh = mdib_connect("dctp://xyz.com/repository_2",
        "option_x=p option_y=q option_z");
    if ( sh != NULL )
    {
        // open "postal_address"
        ch = mdib_open(sh,"postal_address","read-only");
        city_name = mdib_get_value_as_str8(ch,"city",
            NULL, NULL, // ignore label info
            NULL, NULL, // ignore type info
            value_type, city_name, // save value info
            NULL, NULL // ignore property info
            );
    }
```

6.4.4 Typed put value

Synopsis

```
    mdib_put_value_as_str8:
    procedure
    (
        session: mdib_handle, // session handle
        dst_identifier: characterstring, // dst object name
        src_value: str8, // src value
    )
```

© ISO 2004 – All rights reserved

```
returns (str8)
raises (bad_conversion),

mdib_put_value_as_str16:
procedure
(
    session: mdib_handle, // session handle
    dst_identifier: characterstring, // dst object name
    src_value: str16, // src value
)
returns (str16)
raises (bad_conversion),

mdib_put_value_as_str32:
procedure
(
    session: mdib_handle, // session handle
    dst_identifier: characterstring, // dst object name
    src_value: str32, // src value
)
returns (str32)
raises (bad_conversion),

mdib_put_value_as_int8:
procedure
(
    session: mdib_handle, // session handle
    dst_identifier: characterstring, // dst object name
    src_value: int8, // src value
)
returns (int8)
raises (bad_conversion),

mdib_put_value_as_uint8:
procedure
(
    session: mdib_handle, // session handle
    dst_identifier: characterstring, // dst object name
    src_value: uint8, // src value
)
returns (uint8)
raises (bad_conversion),

mdib_put_value_as_int16:
procedure
(
    session: mdib_handle, // session handle
    dst_identifier: characterstring, // dst object name
    src_value: int16, // src value
)
returns (int16)
raises (bad_conversion),

mdib_put_value_as_uint16:
procedure
```

```
(
    session: mdib_handle, // session handle
    dst_identifier: characterstring, // dst object name
    src_value: uint16, // src value
)
returns (uint16)
raises (bad_conversion),

mdib_put_value_as_int32:
procedure
(
    session: mdib_handle, // session handle
    dst_identifier: characterstring, // dst object name
    src_value: int32, // src value
)
returns (int32)
raises (bad_conversion),

mdib_put_value_as_uint32:
procedure
(
    session: mdib_handle, // session handle
    dst_identifier: characterstring, // dst object name
    src_value: uint32, // src value
)
returns (uint32)
raises (bad_conversion),

mdib_put_value_as_int64:
procedure
(
    session: mdib_handle, // session handle
    dst_identifier: characterstring, // dst object name
    src_value: int64, // src value
)
returns (int64)
raises (bad_conversion),

mdib_put_value_as_uint64:
procedure
(
    session: mdib_handle, // session handle
    dst_identifier: characterstring, // dst object name
    src_value: uint64 // src value
)
returns (unit64)
raises (bad_conversion),

mdib_put_value_as_int128:
procedure
(
    session: mdib_handle, // session handle
    dst_identifier: characterstring, // dst object name
    src_value: int128, // src value
)
```

© ISO 2004 – All rights reserved

```
returns (int128)
raises (bad_conversion),

mdib_put_value_as_uint128:
procedure
(
    session: mdib_handle, // session handle
    dst_identifier: characterstring, // dst object name
    src_value: uint128, // src value
)
returns (unit128)
raises (bad_conversion),

mdib_put_value_as_real32:
procedure
(
    session: mdib_handle, // session handle
    dst_identifier: characterstring, // dst object name
    src_value: real32, // src value
)
returns (real32)
raises (bad_conversion),

mdib_put_value_as_real64:
procedure
(
    session: mdib_handle, // session handle
    dst_identifier: characterstring, // dst object name
    src_value: real64, // src value
)
returns (real64)
raises (bad_conversion),

mdib_put_value_as_real80:
procedure
(
    session: mdib_handle, // session handle
    dst_identifier: characterstring, // dst object name
    src_value: real80, // src value
)
returns (real80)
raises (bad_conversion),
```

Description

Gets a value converted to a particular type, as identified by src_identifier. The following types are supported:

— str8: A string of "8-bit* characters, i.e., characters with a repertoire of ISO/IEC 8859-1.
— str16: A string of "16-bit* characters, i.e., characters with a repertoire of ISO/IEC 10646-1 Basic Multilingual Plane (BMP).
— str32: A string of "32-bit* characters, i.e., characters with a repertoire of ISO/IEC 10646-1.
— int8: A signed, 2's complement integer of 8 bits, as described by IEEE 1596.5.
— uint8: An unsigned integer of 8 bits, as described by IEEE 1596.5.
— int16: A signed, 2's complement integer of 16 bits, as described by IEEE 1596.5.

- uint16: An unsigned integer of 16 bits, as described by IEEE 1596.5.
- int32: A signed, 2's complement integer of 32 bits, as described by IEEE 1596.5.
- uint32: An unsigned integer of 32 bits, as described by IEEE 1596.5.
- int64: A signed, 2's complement integer of 64 bits, as described by IEEE 1596.5.
- uint64: An unsigned integer of 64 bits, as described by IEEE 1596.5.
- int128: A signed, 2's complement integer of 128 bits, as described by IEEE 1596.5.
- uint128: An unsigned integer of 128 bits, as described by IEEE 1596.5.
- real32: An IEC 60599, 32-bit floating point number.
- real64: An IEC 60599, 64-bit floating point number.
- real80: An IEC 60599, 80-bit floating point number.

If mdib_put_value_* is successful, it returns the new value associated with dst_identifier; otherwise, error returns are indicated by null pointers for strings, -1 for signed integers, 0 for unsigned integers, and NaN (not a number) for reals.

Example

```
// C/C++ illustration
mdib_handle sh; // session handle
mdib_handle ch; // child session handle
string city_name; // city name in postal address
sh = mdib_connect("dctp://xyz.com/repository_2",
    "option_x=p option_y=q option_z");
if ( sh != NULL )
{
    // open "postal_address"
    ch = mdib_open(sh,"postal_address","read-only");
    city_name = mdib_put_value_as_str8(ch,
        "city","New York")
}
```

6.5 Miscellaneous

6.5.1 Make Object message

Synopsis

```
mdib_make_object:
procedure
(
    in session: mdib_handle, // session handle
    in src_identifier: characterstring, // src object name
    in src_label_type: characterstring, // assigned label: typeof
    in src_label_ptr: pointer, // assigned label: ptr to
    in src_type_type: characterstring, // assigned type: typeof
    in src_type_ptr: pointer, // assigned type: ptr to
    in src_object_type: characterstring, // assigned value: typeof
    in src_object_ptr: pointer, // assigned value: ptr to
    in src_proplist_type: characterstring, // assigned proplist: typeof
    in src_proplist_ptr: pointer, // assigned proplist: ptr to
)
returns (state(success,failure)),
```

© ISO 2004 – All rights reserved

Description

Creates a new object. The parameters are the same as the mdib_put_object.

If mdib_make_object is successful, it returns success, otherwise error return is indicated by a return of failure.

Example

```
// C/C++ illustration
mdib_handle sh; // session handle
mdib_handle ch; // child session handle
sh = mdib_connect("dctp://xyz.com/repository_2",
    "option_x=p option_y=q option_z");
if ( sh != NULL )
{
    // open "postal_address"
    ch = mdib_open(sh,"postal_address","read-only");
    // make object "city"
    if ( mdib_make_object(ch, "city", NULL, NULL,
        STRING_PARAMETER, "string",
        STRING_PARAMETER, "New York",
        NULL, NULL) == -1 )
    {
        // error
    }
    // success
}
```

6.5.2 Remove Object message

Synopsis

```
mdib_remove_object:
procedure
(
    in session: mdib_handle, // session handle
    in src_identifier: characterstring, // src object name
)
returns (state(success,failure)),
```

Description

Removes an object. The src_identifier identifies the object to delete.

If mdib_remove_object is successful, it returns success, otherwise error return is indicated by a return of failure.

Example

```
// C/C++ illustration
mdib_handle sh; // session handle
mdib_handle ch; // child session handle
sh = mdib_connect("dctp://xyz.com/repository_2",
    "option_x=p option_y=q option_z");
if ( sh != NULL )
{
    // open "postal_address"
    ch = mdib_open(sh,"postal_address","read-only");
    // make object "city"
    if ( mdib_remove_object(ch, "city") == -1 )
    {
        // error
    }
    // success
}
```

6.5.3 Link Object message

Synopsis

```
    mdib_link_object:
    procedure
    (
        session: mdib_handle, // session handle
        in src_identifier: characterstring, // src object name
        in dst_identifier: characterstring, // dst object name
        in link_type: characterstring, // link type: soft, hard
    )
    returns (state(success,failure)),
```

Description

Links an existing object to a new name. The src_identifier is the name of the existing object. The dst_identifier is the name of the new object. The link_type is the link type, either "hard" (equal references), or "soft" (automatically dereferences the link).

If mdib_link_object is successful, it returns success, otherwise error return is indicated by a return of failure.

Example

```
    // C/C++ illustration
    mdib_handle sh; // session handle
    mdib_handle ch; // child session handle
    sh = mdib_connect("dctp://xyz.com/repository_2",
        "option_x=p option_y=q option_z");
    if ( sh != NULL )
    {
        // open "postal_address"
        ch = mdib_open(sh,"postal_address","read-only");
        // link object "state" to "province"
        if ( mdib_link_object(ch, "state", "province") == -1 )
        {
            // error
        }
        // success
    }
```

6.5.4 List Object message

Synopsis

```
mdib_list_object
(
    in session: mdib_handle, // session handle
    in src_identifier: characterstring, // src object name (wildcard)
)
returns (characterstring),
```

Description

Lists all objects that match src_identifier.

If mdib_list_object is successful, it returns a string; otherwise, error return is indicated by the NULL pointer.

Example

```
// C/C++ illustration
mdib_handle sh; // session handle
mdib_handle ch; // child session handle
string match_list; // resulting list
sh = mdib_connect("dctp://xyz.com/repository_2",
    "option_x=p option_y=q option_z");
if ( sh != NULL )
{
    // open "postal_address"
    ch = mdib_open(sh,"postal_address","read-only");
    // list objects
    match_list = mdib_list_object(ch, "*");
    if ( match_list == NULL )
    {
        // error
    }
    // success
}
```

7 Bindings

This Part describes the common conceptual model and common data services across all coding bindings. A conforming API binding shall conform to the requirements described in Clauses 4, 5, 6, 8, and 9 of this Part.

8 Administration

There are no administrative requirements.

9 Conformance

9.1 API conformance paradigm

The conformance paradigm for 20944 consists of the following conformance roles: API application, API environment.

Definitions: support, use, test, access, probe

The following terms are defined in the context of API conformance for data interchange participants:

- A "supported" feature is one that is implemented by the API environment and may be used by any API application.
- A feature is "used" if it is read, written, or operated upon by an API application.
- A feature is "tested" if an API application inquires about the existence of that feature in the API environment.
- A feature is "accessed" if an API application attempts to read or write data associated with that feature.
- A feature is "probed" if an API application implicitly tests the existence of that feature by attempting to use the feature (see "use" above) within an API environment that permits error recovery.

NOTE API conformance makes requirements upon all data interchange participants: the API environment (implementations of the interface, services, resources, etc., of the API binding); and the API application (applications that use the API binding).

9.2 API application

An strictly conforming API application shall use the features of the API according to the requirements of the API binding. A strictly conforming API application shall not perform implementation-defined behavior.

An conforming API application shall use the features of the API according to the requirements of the API binding.

9.3 API environment

An strictly conforming API application shall implement all the features of the API according to the requirements of the API binding. A strictly conforming API environment shall not implement extensions to the API binding.

An conforming API application shall implement all the features of the API according to the requirements of the API binding. A conforming API environment may implement extensions to the API binding. A conforming API may provide services to test, access, and/or probe for additional capabilities.

© ISO 2004 – All rights reserved