INTERNATIONAL STANDARD

ISO/IEC 20944-3

First edition
2013-01-15

Information technology — Metadata Registries Interoperability and Bindings (MDR-IB) —

Part 3: API bindings

Technologies de l'information — Interopérabilité et liaisons des registres de métadonnées (MDR-IB) —

Partie 3: Liaisons API

Reference number
ISO/IEC 20944-3:2013(E)

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

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. In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1.

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

The main task of the joint technical committee is to prepare International Standards. Draft International Standards adopted by the joint technical committee are circulated to national bodies for voting. Publication as an International Standard requires approval by at least 75 % of the national 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 and IEC shall not be held responsible for identifying any or all such patent rights.

ISO/IEC 20944-3 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 32, *Data management and interchange*.

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

— *Part 1: Framework, common vocabulary, and common provisions for conformance*

— *Part 2: Coding bindings*

— *Part 3: API bindings*

— *Part 4: Protocol bindings*

— *Part 5: Profiles*

Introduction

The ISO/IEC 20944 series of International Standards provides the bindings and their interoperability for metadata registries, such as those specified in the ISO/IEC 11179 series of International Standards.

This part of ISO/IEC 20944 contains provisions that are common to API bindings (Clauses 4-10) and the API bindings themselves (Clause 11 onward). The API bindings have commonality in their conceptualization of the services provided. 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.

Clause 11 and onward are the actual API bindings themselves. The clauses of this part of ISO/IEC 20944 are organized such that future amendments are possible, which would include additional API bindings.

Information technology — Metadata Registries Interoperability and Bindings (MDR-IB) —

Part 3:
API bindings

1 Scope

The ISO/IEC 20944 series of International Standards describes codings, application programming interfaces (APIs), and protocols for interacting with an ISO/IEC 11179 metadata registry (MDR).

This part of ISO/IEC 20944 specifies provisions that are common across API bindings for the ISO/IEC 20944 series of International Standards, and provides the individual API bindings themselves.

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 9899:1999, *Programming languages — C*

ISO/IEC 11404:2007, *Information technology — General-Purpose Datatypes (GPD)*

ISO/IEC 13886:1996, *Information technology — Language-Independent Procedure Calling (LIPC)*

ISO/IEC 16262, *Information technology — Programming languages, their environments and system software interfaces — ECMAScript language specification*

ISO/IEC 20944-1:2013, *Information technology — Metadata Registries Interoperability and Bindings (MDR-IB) — Part 1: Framework, common vocabulary, and common provisions for conformance*

IETF RFC 3986, *Uniform Resource Identifier (URI): Generic Syntax*, January 2005

3 Terms and definitions

For the purposes of this document, the terms and definitions given in ISO/IEC 20944-1 apply.

4 Intended use of this part of ISO/IEC 20944

The purpose of this part of ISO/IEC 20944 is to provide a common set of services (common API provisions) and standardized API bindings such that portable programs can be written to access the MDR repositories. These programs are portable in the sense that the same program should behave similarly across all operating environments and the same program should be able to access MDR repositories that conform to this International Standard.

5 Abstract model

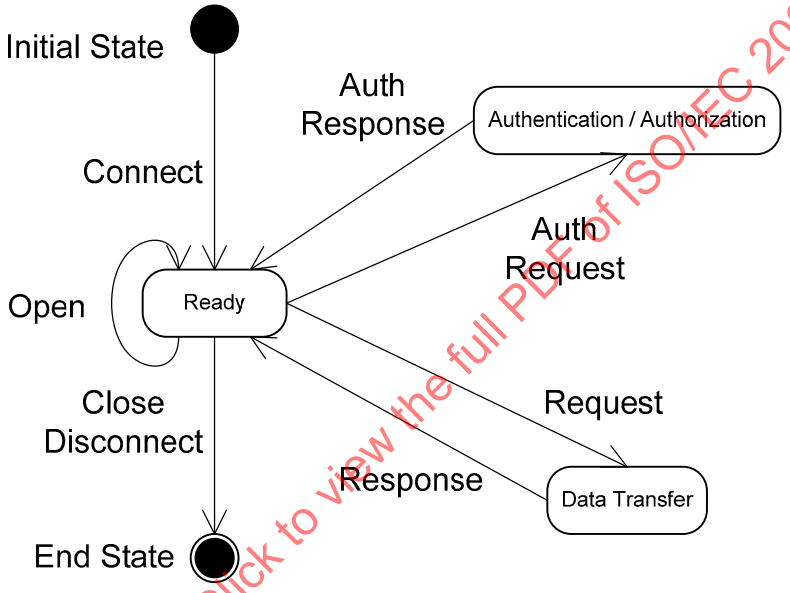
5.1 General

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

5.2 Session paradigm

The following state diagram describes the different states of the services:



The states are:

— **initial state:** The initial state of an instance of the API.

— **end state:** The final state of an instance of the API.

— **ready:** The API is ready to for service requests.

— **authentication-authorization:** The API is processing an authentication-authorization request.

— **data transfer:** The API is processing a data transfer request.

The events are:

— **connect:** Setting up the initial connection.

— **open:** Adding more parameters to create a new handle and the context of a current node path for that handle.

— **close:** Destroying a handle created by open.

— **disconnect:** Knocking down a connection.

— **request:** A data transfer request.

— **response:** A data transfer response.

— **auth request:** An authentication-authorization request.

— **auth response:** An authentication-authorization response.

5.3 Security framework

The security framework provides a common technique for accessing security services and supports a common technique for implementing security services.

The common accessing technique uses the Request-Response Authorization-Authentication (RRAA) services. In the RRAA services, a Request is places for authorization, authentication, or some other security service. The Request provides the data, as required for the particular RRAA service. The RRAA service then provides a Response to the request. The services are symmetric in that both the API application (e.g., the program using the API) and the API environment (e.g., the underlying services and infrastructure) may each make requests of the other party, i.e., the API application can make requests to the API environment, and the API environment can make requests to the API application.

The supporting infrastructure should provide a run-time, dynamically configurable selection of RRAA services such that programs do not need to be recompiled or re-linked to take advantage of a new RRAA services or new security technologies[1].

This International Standard makes no requirements for a particular set of security services. The available services are implementation-defined.

5.4 Hierarchical navigation paths

The API services may access data via hierarchical navigation paths[2]. The navigation paths are based upon Uniform Resource Identifiers, as defined in RFC 3986. Absolute paths, which start at the top of a repository, are specified in subclause 3.3 of RFC 3986 under "absolute paths". Relative paths, which are relative to the current path[3], are specified in subclause 5 of RFC 3986 under "relative URIs".

The common accessing technique uses the Request-Response Authorization-Authentication (RRAA) services. In the RRAA services, a Request is places for authorization, authentication, or some other security service. The Request provides the data, as required for the particular RRAA service. The RRAA service then provides a Response to the request. The services are symmetric in that both the API application (e.g., the program using the API) and the API environment (e.g., the underlying services and infrastructure) may each make requests of the other party, i.e., the API application can make requests to the API environment, and the API environment can make requests to the API application.

1 See the white paper "Making Login Services Independent of Authentication Technologies", by Vipin Samar and Charlie Lai, that was presented at the Third ACM Conference on Computer Communications and Security (1996-03), which is available at

http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.117.8767&rep=rep1&type=pdf

and the supporting API framework and services of Pluggable Authentication Modules (PAM) at

http://en.wikipedia.org/wiki/Pluggable_Authentication_Modules

2 A hierarchical navigation path does not imply that the internal structure of data is hierarchical itself.

3 See Get Path and Put Path services in subclause 6.3.

6 Services

6.1 Use of ISO/IEC 13886

The notation of ISO/IEC 13886 Language-Independent Procedure Calls is used to describe service interfaces.

6.2 Session establishment services

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

6.2.1 System information

Synopsis

```
mdrib_system_info:
procedure
(
    in session: mdrib_handle // session handle
)
returns characterstring,
```

Description

The **mdrib_system_info** service retrieves implementation-defined newline terminated, name-value pairs regarding the current configuration, limitations, and parameters of the session in **mdrib_handle**. If **mdrib_handle** is NULL, the information is for all sessions.

The following parameters are available on all implementations:

— "apistyle=xxx", where "xxx" is one of "synchronous" (API always blocks waiting for return), "nonblock" (API always returns immediately), "interrupt" (API interrupts when complete), "callback" (API performs callback when complete)

Example

```
// C/C++ illustration
printf(NULL, "mdrib configuration:\n%s", mdrib_system_info());

// sample output
maximum_input_string=4095
supported_types=str8,int,char,long
version_info=3.17
version_description="C implementation with LDAP backend"
```

6.2.2 Configuration

Synopsis

```
// values accessible via "._system_info" identifier
data_object_identifier_length // Maximum supported data object identifier length
datatype_family_support // List of families supported
identifier_character_family_support // Allowed data object identifier characters
navigation_identifier_max // Maximum hierarchical navigation identifier length
session_max: Maximum number of simultaneous opened sessions
octet_transfer_max: // Maximum octets of data for a data object

// error codes accessible via "._error_family" and "._error_list"
error_family // Name of error code system
error_list // List of error codes and meanings
```

Description

The configuration features are accessible via Get and Put services via the identifiers "._system_info", "._error_family", and "._error_list".

Example

```
// C/C++ illustration
if ( strcmp(mdrib_get_value_as_str8(NULL,"._error_family",0,0,NULL),"POSIX") == 0 )
{
    // Error returns/handling are based upon POSIX standard
    // ...
}
```

6.2.3 Connect

Synopsis

```
mdrib_connect:
procedure
(
    in target: characterstring, // repository to connect to
    in options: characterstring, // connect options
)
returns mdrib_handle,
```

Description

Creates a new session to a data repository as named by target, an implementation-defined characterstring. The options parameter is a whitespace-separated list of name-value pairs that describe an implementation-defined set of connection options. Whitespace inside double-quoted characterstrings is escaped, i.e., this whitespace does not function as a name-value separator. If successful, returns a session handle to the repository, but does not request access (see mdrib_open).[4] If not successful, returns a null session handle.

Example

```
// C/C++ illustration
mdrib_handle sh; // session handle
mdrib_handle ch; // child session handle
sh = mdrib_connect("http://xyz.com/repository_2",
    "option_x=\"j k\" option_y=q option_z"); // note: option_x has the value "j k"
if ( sh != NULL )
{
    ch = mdrib_open(sh,"postal_address","read-only");
    // ...
    mdrib_close(ch);
    // ...
    mdrib_disconnect(sh);
}
```


[4] It is possible to combine connect, open, get, and close services in a single line of programming to give a state-less style of programming:

```
value = mdrib_get_as_str8( handle = mdrib_open( mdrib_connect("http://xyz.com/repository_2",
        "option_x=p option_y=q option_z"), "open options"), "city_name"), mdrib_close(handle);
```

6.2.4 Disconnect

Synopsis

```
mdrib_disconnect:
procedure
(
    in session: mdrib_handle // session handle
)
returns (state(success,failure)),
```

Description

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

Example

```
// C/C++ illustration
mdrib_handle sh; // session handle
sh = mdrib_connect("http://xyz.com/repository_2",
    "option_x=p option_y=q option_z");
// ... open session threads
// ... access metadata
// ... close session threads
mdrib_disconnect(sh);
```

6.2.5 Open

Synopsis

```
mdrib_open:
procedure
(
    in session: mdrib_handle, // session handle
    in node: characterstring, // portion of repository to open
    in options: characterstring, // open options
)
returns (mdrib_handle)
```

Description

Opens a child session within a session to data 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 implementation-defined.

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

Example

```
// C/C++ illustration
mdrib_handle sh; // session handle
mdrib_handle ch; // child session handle
sh = mdrib_connect("http://xyz.com/repository_2",
    "option_x=p option_y=q option_z");
if ( sh != NULL )
{
    ch = mdrib_open(sh,"postal_address","read-only");
    // ...
    mdrib_close(ch);
    // ...
    mdrib_disconnect(sh);
}
```

6.2.6 Close

Synopsis

```
mdrib_close:
procedure
(
    in mdrib_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
mdrib_handle sh; // session handle
mdrib_handle ch; // child session handle
sh = mdrib_connect("http://xyz.com/repository_2",
    "option_x=p option_y=q option_z");
if ( sh != NULL )
{
    ch = mdrib_open(sh,"postal_address","read-only");
    // ...
    mdrib_close(ch);
    // ...
    mdrib_disconnect(sh);
}
```

6.3 Session parameter services

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

6.3.1 Get path

Synopsis

```
mdrib_get_path:
procedure
(
    in session: mdrib_handle, // session handle
)
returns (characterstring),
```

Description

Retrieves the current node path for the session mdrib_handle.

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

Example

```
// C/C++ illustration
mdrib_handle sh; // session handle
mdrib_handle ch; // child session handle
sh = mdrib_connect("http://xyz.com/repository_2",
    "option_x=p option_y=q option_z");
if ( sh != NULL )
{
    // open "postal_address"
    ch = mdrib_open(sh,"postal_address","read-only");
    // change path to "city"
    mdrib_put_path(ch,"city");
    // retrieve path, should be "city"
    new_path = mdrib_get_path(ch);
    if ( new_path != "city" )
    {
        // error
    }
}
```

6.3.2 Put path

Synopsis

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

Description

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

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

Example

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

6.4 Security services

The following services are supported.

6.4.1 Request Authorization/Authentication

Synopsis

```
mdrib_request_auth:
procedure
(
    in session: mdrib_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 mdrib_request_auth is successful, it returns success, otherwise error return is indicated by a return of failure.

Example

```
// C/C++ illustration
mdrib_handle sh; // session handle
mdrib_handle ch; // child session handle
sh = mdrib_connect("http://xyz.com/repository_2",
    "option_x=p option_y=q option_z");
if ( mdrib_request_auth(sh, "symmetric", "password") !=
    "swordfish" )
{
    // ... error, abort because of password mismatch
    return;
}
if ( sh != NULL )
{
    ch = mdrib_open(sh,"postal_address","read-only");
    // ...
    mdrib_close(ch);
    // ...
    mdrib_disconnect(sh);
}
```

6.4.2 Response Authorization/Authentication

Synopsis

```
mdrib_response_auth:
procedure
(
    in session: mdrib_handle, // session handle
    in auth_type: characterstring, // 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 6.4.1 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 mdrib_response_auth is successful, it returns success, otherwise error return is indicated by a return of failure.

Example

```
// C/C++ illustration
mdrib_handle sh; // session handle
mdrib_handle ch; // child session handle
sh = mdrib_connect("http://xyz.com/repository_2",
    "option_x=p option_y=q option_z");
mdrib_response_auth(sh,"symmetric",handler);
if ( sh != NULL )
{
    ch = mdrib_open(sh,"postal_address","read-only");
    // ...
    mdrib_close(ch);
    // ...
    mdrib_disconnect(sh);
}

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

6.5 Data transfer services

The following data transfer are supported.

6.5.1 Get value

Synopsis

```
mdrib_get_value:
procedure
(
    in session: mdrib_handle, // session handle
    in octet_offset: integer, // starting offset for transfer
    in octet_length: integer, // maximum number of octets to transfer
    out octet_transferred: integer, // maximum number of octets transferred
    in src_identifier: characterstring, // src object name
    in dst_object_type: characterstring, // saved value: typeof
    out dst_object_ptr: pointer, // saved value: ptr to
)
returns (state(success,failure)),
```

Description

Gets a value converted to a particular datatype, as identified by src_identifier. If octet_offset is not -1, the first octet_offset octets are skipped in the data transfer. If octet_length is not -1, a maximum of octet_length octets are transferred. The total number of octets transferred is returned in octets_transferred. The dst_object_type is the datatype of the data object where the data is transferred, and dst_object_ptr points to the destination data.

Example

```
// C/C++ illustration
mdrib_handle sh; // session handle
mdrib_handle ch; // child session handle
mdrib_type_type value_type; // datatype information
characterstring city_name;
int octet_transferred;
value_type = "characterstring"; // target datatype
sh = mdrib_connect("http://xyz.com/repository_2",
    "option_x=p option_y=q option_z");
if ( sh != NULL )
{
    // open "postal_address"
    ch = mdrib_open(sh,"postal_address","read-only");
    success = mdrib_get_value(ch, -1, -1, octet_transferred,
        "city", value_type, city_name
        );
}
```

6.5.2 Typed get value

Synopsis

```
mdrib_get_value_as_str8:
procedure
(
    session: mdrib_handle, // session handle
    src_identifier characterstring, // src object name
    in octet_offset: integer, // starting offset for transfer
    in octet_length: integer, // maximum number of octets to transfer
    out octet_transferred: integer, // maximum number of octets transferred
)
returns (str8)
raises (bad_conversion),

mdrib_get_value_as_str16:
procedure
(
    session: mdrib_handle, // session handle
    src_identifier characterstring, // src object name
    in octet_offset: integer, // starting offset for transfer
    in octet_length: integer, // maximum number of octets to transfer
    out octet_transferred: integer, // maximum number of octets transferred
)
returns (str16)
raises (bad_conversion),

mdrib_get_value_as_str32:
procedure
(
```

```
    session: mdrib_handle, // session handle
    src_identifier characterstring, // src object name
    in octet_offset: integer, // starting offset for transfer
    in octet_length: integer, // maximum number of octets to transfer
    out octet_transferred: integer, // maximum number of octets transferred
)
returns (str32)
raises (bad_conversion),

mdrib_get_value_as_int8:
procedure
(
    session: mdrib_handle, // session handle
    src_identifier characterstring, // src object name
)
returns (int8)
raises (bad_conversion),

mdrib_get_value_as_uint8:
procedure
(
    session: mdrib_handle, // session handle
    src_identifier characterstring, // src object name
)
returns (unit8)
raises (bad_conversion),

mdrib_get_value_as_int16:
procedure
(
    session: mdrib_handle, // session handle
    src_identifier characterstring, // src object name
)
returns (int16)
raises (bad_conversion),

mdrib_get_value_as_uint16:
procedure
(
    session: mdrib_handle, // session handle
    src_identifier characterstring, // src object name
)
returns (uint16)
raises (bad_conversion),

mdrib_get_value_as_int32:
procedure
(
    session: mdrib_handle, // session handle
    src_identifier characterstring, // src object name
)
returns (int32)
raises (bad_conversion),

mdrib_get_value_as_uint32:
procedure
(
    session: mdrib_handle, // session handle
    src_identifier characterstring, // src object name
)
```

```
returns (uint32)
raises (bad_conversion),

mdrib_get_value_as_int64:
procedure
(
    session: mdrib_handle, // session handle
    src_identifier characterstring, // src object name
)
returns (int64)
raises (bad_conversion),

mdrib_get_value_as_uint64:
procedure
(
    session: mdrib_handle, // session handle
    src_identifier characterstring, // src object name
)
returns (uint64)
raises (bad_conversion),

mdrib_get_value_as_int128:
procedure
(
    session: mdrib_handle, // session handle
    src_identifier characterstring, // src object name
)
returns (int128)
raises (bad_conversion),

mdrib_get_value_as_uint128:
procedure
(
    session: mdrib_handle, // session handle
    src_identifier characterstring, // src object name
)
returns (uint128)
raises (bad_conversion),

mdrib_get_value_as_real32:
procedure
(
    session: mdrib_handle, // session handle
    src_identifier characterstring, // src object name
)
returns (real32)
raises (bad_conversion),

mdrib_get_value_as_real64:
procedure
(
    session: mdrib_handle, // session handle
    src_identifier characterstring, // src object name
)
returns (real64)
raises (bad_conversion),

mdrib_get_value_as_real80:
procedure
```

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

mdrib_get_value_as_datetime:
procedure
(
    session: mdrib_handle, // session handle
    src_identifier characterstring, // src object name
)
returns (datetime)
raises (bad_conversion),

mdrib_get_value_as_duration:
procedure
(
    session: mdrib_handle, // session handle
    src_identifier characterstring, // src object name
)
returns (duration)
raises (bad_conversion),

mdrib_get_value_as_uri:
procedure
(
    session: mdrib_handle, // session handle
    src_identifier characterstring, // src object name
)
returns (uri)
raises (bad_conversion),
```

Description

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

- str8: A characterstring of 8-bit characters, i.e., characters with a repertoire of ISO/IEC 8859-1.
- str16: A characterstring of 16-bit characters, i.e., characters with a repertoire of ISO/IEC 10646 Basic Multilingual Plane (BMP).[5]
- str32: A characterstring of 32-bit characters, i.e., characters with a repertoire of ISO/IEC 10646.[6]
- 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.

5 The encoding of 16-bit characters is implementation-defined.

6 The encoding of 32-bit characters is implementation-defined.

— real32: An IEC 60559, 32-bit floating point number.
— real64: An IEC 60559, 64-bit floating point number.
— real80: An IEC 60559, 80-bit floating point number.
— datetime: A date-time value.
— duration: A time duration value.
— uri: A uniform resource identifier.

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

Example

```
// C/C++ illustration
mdrib_handle sh; // session handle
mdrib_handle ch; // child session handle
characterstring city_name; // city name in postal address
sh = mdrib_connect("http://xyz.com/repository_2",
    "option_x=p option_y=q option_z");
if ( sh != NULL )
{
    // open "postal_address"
    ch = mdrib_open(sh,"postal_address","read-only");
    city_name = mdrib_get_value_as_str8(ch,"city")
}
```

6.5.3 Put value

Synopsis

```
integer mdrib_put_value
(
    in session: mdrib_handle, // session handle
    in src_identifier: characterstring, // src object name
     in octet_offset: integer, // starting offset for transfer at destination
    in octet_length: integer, // maximum number of octets to transfer
    out octet_transferred: integer, // maximum number of octets transferred
    in src_object_type: characterstring, // source value: typeof
    in src_object_ptr: pointer, // source value: ptr to
)
```

Description

Puts a value in an object named src_identifier. If octet_offset is not -1, the first octet_offset octets are skipped in the data transfer at the target. If octet_length is not -1, a maximum of octet_length octets are transferred. The total number of octets transferred is returned in octets_transferred. The src_object_type is the datatype of the data object where the data is transferred from, and src_object_ptr points to the source data.

Example

```
// C/C++ illustration
mdrib_handle sh; // session handle
mdrib_handle ch; // child session handle
mdrib_type_type value_type; // datatype information
characterstring city_name = "New York";
value_type = "characterstring"; // target datatype
sh = mdrib_connect("http://xyz.com/repository_2",
    "option_x=p option_y=q option_z");
if ( sh != NULL )
```

```
{
    // open "postal_address"
    ch = mdrib_open(sh,"postal_address","read-write");
    status = mdrib_put_value(ch,"city",-1,-1,octet_transferred,
        "string", city_name
        );
}
```

6.5.4 Typed put value

Synopsis

```
mdrib_put_value_as_str8:
procedure
(
    session: mdrib_handle, // session handle
    in octet_offset: integer, // starting offset for transfer at destination
    in octet_length: integer, // maximum number of octets to transfer
    out octet_transferred: integer, // maximum number of octets transferred
    dst_identifier: characterstring, // dst object name
    src_value: str8, // src value
)
returns (str8)
raises (bad_conversion),

mdrib_put_value_as_str16:
procedure
(
    session: mdrib_handle, // session handle
    in octet_offset: integer, // starting offset for transfer at destination
    in octet_length: integer, // maximum number of octets to transfer
    out octet_transferred: integer, // maximum number of octets transferred
    dst_identifier: characterstring, // dst object name
    src_value: str16, // src value
)
returns (str16)
raises (bad_conversion),

mdrib_put_value_as_str32:
procedure
(
    session: mdrib_handle, // session handle
    in octet_offset: integer, // starting offset for transfer at destination
    in octet_length: integer, // maximum number of octets to transfer
    out octet_transferred: integer, // maximum number of octets transferred
    dst_identifier: characterstring, // dst object name
    src_value: str32, // src value
)
returns (str32)
raises (bad_conversion),

mdrib_put_value_as_int8:
procedure
(
    session: mdrib_handle, // session handle
    dst_identifier: characterstring, // dst object name
    src_value: int8, // src value
)
returns (int8)
raises (bad_conversion),
```

```
mdrib_put_value_as_uint8:
procedure
(
    session: mdrib_handle, // session handle
    dst_identifier: characterstring, // dst object name
    src_value: uint8, // src value
)
returns (uint8)
raises (bad_conversion),

mdrib_put_value_as_int16:
procedure
(
    session: mdrib_handle, // session handle
    dst_identifier: characterstring, // dst object name
    src_value: int16, // src value
)
returns (int16)
raises (bad_conversion),

mdrib_put_value_as_uint16:
procedure
(
    session: mdrib_handle, // session handle
    dst_identifier: characterstring, // dst object name
    src_value: uint16, // src value
)
returns (uint16)
raises (bad_conversion),

mdrib_put_value_as_int32:
procedure
(
    session: mdrib_handle, // session handle
    dst_identifier: characterstring, // dst object name
    src_value: int32, // src value
)
returns (int32)
raises (bad_conversion),

mdrib_put_value_as_uint32:
procedure
(
    session: mdrib_handle, // session handle
    dst_identifier: characterstring, // dst object name
    src_value: uint32, // src value
)
returns (uint32)
raises (bad_conversion),

mdrib_put_value_as_int64:
procedure
(
    session: mdrib_handle, // session handle
    dst_identifier: characterstring, // dst object name
    src_value: int64, // src value
)
returns (int64)
raises (bad_conversion),
```

```
mdrib_put_value_as_uint64:
procedure
(
    session: mdrib_handle, // session handle
    dst_identifier: characterstring, // dst object name
    src_value: uint64 // src value
)
returns (unit64)
raises (bad_conversion),

mdrib_put_value_as_int128:
procedure
(
    session: mdrib_handle, // session handle
    dst_identifier: characterstring, // dst object name
    src_value: int128, // src value
)
returns (int128)
raises (bad_conversion),

mdrib_put_value_as_uint128:
procedure
(
    session: mdrib_handle, // session handle
    dst_identifier: characterstring, // dst object name
    src_value: uint128, // src value
)
returns (unit128)
raises (bad_conversion),

mdrib_put_value_as_real32:
procedure
(
    session: mdrib_handle, // session handle
    dst_identifier: characterstring, // dst object name
    src_value: real32, // src value
)
returns (real32)
raises (bad_conversion),

mdrib_put_value_as_real64:
procedure
(
    session: mdrib_handle, // session handle
    dst_identifier: characterstring, // dst object name
    src_value: real64, // src value
)
returns (real64)
raises (bad_conversion),

mdrib_put_value_as_real80:
procedure
(
    session: mdrib_handle, // session handle
    dst_identifier: characterstring, // dst object name
    src_value: real80, // src value
)
returns (real80)
raises (bad_conversion),
```

```
mdrib_put_value_as_datetime:
procedure
(
    session: mdrib_handle, // session handle
    dst_identifier: characterstring, // dst object name
    src_value: datetime, // src value
)
returns (datetime)
raises (bad_conversion),

mdrib_put_value_as_duration:
procedure
(
    session: mdrib_handle, // session handle
    dst_identifier: characterstring, // dst object name
    src_value: duration, // src value
)
returns (duration)
raises (bad_conversion),

mdrib_put_value_as_uri:
procedure
(
    session: mdrib_handle, // session handle
    dst_identifier: characterstring, // dst object name
    src_value: uri, // src value
)
returns (uri)
raises (bad_conversion),
```

Description

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

- str8: A characterstring of "8-bit* characters, i.e., characters with a repertoire of ISO/IEC 8859-1.
- str16: A characterstring of "16-bit* characters, i.e., characters with a repertoire of ISO/IEC 10646 Basic Multilingual Plane (BMP).
- str32: A characterstring of "32-bit* characters, i.e., characters with a repertoire of ISO/IEC 10646.
- 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 60559, 32-bit floating point number.
- real64: An IEC 60559, 64-bit floating point number.
- real80: An IEC 60559, 80-bit floating point number.
- datetime: A date-time value.
- duration: A time duration value.
- uri: A uniform resource identifier.

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

Example

```
// C/C++ illustration
mdrib_handle sh; // session handle
mdrib_handle ch; // child session handle
characterstring city_name; // city name in postal address
sh = mdrib_connect("http://xyz.com/repository_2",
    "option_x=p option_y=q option_z");
if ( sh != NULL )
{
    // open "postal_address"
    ch = mdrib_open(sh,"postal_address","read-only");
    city_name = mdrib_put_value_as_str8(ch,
        "city","New York")
}
```

6.6 Miscellaneous

6.6.1 Make Object service

Synopsis

```
mdrib_make_object:
procedure
(
    in session: mdrib_handle, // session handle
    in octet_offset: integer, // starting offset for transfer at destination
    in octet_length: integer, // maximum number of octets to transfer
    out octet_transferred: integer, // maximum number of octets transferred
    in src_object_type: characterstring, // source
    value: typeof, // the type of object
    in src_object_ptr: pointer, // source value: ptr to
)
returns (state(success,failure)),
```

Description

Creates a new implementation-defined object. The parameters are the same as the mdrib_put_object.

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

Example

```
// C/C++ illustration
mdrib_handle sh; // session handle
mdrib_handle ch; // child session handle
sh = mdrib_connect("http://xyz.com/repository_2",
    "option_x=p option_y=q option_z");
if ( sh != NULL )
{
    // open "postal_address"
    ch = mdrib_open(sh,"postal_address","read-write");
    city_name = mdrib_make_object(ch,"city",-1,-1,NULL,
        "city_name", string, city_name);
}
```

6.6.2 Delete Object service

Synopsis

```
mdrib_delete_object:
procedure
(
    in session: mdrib_handle, // session handle
    in src_identifier: characterstring, // src object name
    in qualifier: characterstring, // delete service qualifier
)
returns (state(success,failure)),
```

Description

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

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

Example

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

6.6.3 Link Object service

Synopsis

```
mdrib_link_object:
procedure
(
    session: mdrib_handle, // session handle
    in src_identifier: characterstring, // src object name
    in dst_identifier: characterstring, // dst object name
    in qualifier: 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 mdrib_link_object is successful, it returns success, otherwise error return is indicated by a return of failure.

Example

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

6.6.4 Property Find service

Synopsis

```
mdrib_property_find
(
    in session: mdrib_handle, // session handle
    in src_identifier: characterstring, // src object name (wildcard)
    in qualifier: characterstring, // propfind service qualifier
)
returns (characterstring),
```

Description

Lists all objects that match src_identifier. See RFC 4518 PROPFIND service for a list of parameters.

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

Example

```
// C/C++ illustration
mdrib_handle sh; // session handle
mdrib_handle ch; // child session handle
characterstring match_list; // resulting list
sh = mdrib_connect("http://xyz.com/repository_2",
    "option_x=p option_y=q option_z");
if ( sh != NULL )
{
    // open "postal_address"
    ch = mdrib_open(sh,"postal_address","read-only");
    // list objects
    match_list = mdrib_prop_find(ch, "*", "");
    if ( match_list == NULL )
    {
        // error
    }
    // success
}
```

6.6.5 Lock Object service

Synopsis

```
mdrib_lock_object
(
    in session: mdrib_handle, // session handle
    in src_identifier: characterstring, // src object name
    in qualifier: characterstring, // lock service qualifier
)
returns (state(success,failure)),
```

Description

Locks object src_identifier according to src_lock specification. See RFC 4518 for the LOCK specification.

Example

```
// Note 1: See RFC 4518 for lock specification
// Note 2: This example adapted from RFC 4518, subclause 9.10.7
// Locking the resource "0123/value_domain_datatype" ...
mdrib_lock_object(mdrib_handle,
        "0123/value_domain_datatype",
        "<?xml version="1.0" encoding="utf-8" ?> "
        "<D:lockinfo xmlns:D='DAV:'> "
        "  <D:lockscope><D:exclusive/></D:lockscope>"
        "  <D:locktype><D:write/></D:locktype>"
        "  <D:owner>"
        "    <D:href>http://example.org/~ff/contact.html</D:href>"
        "  </D:owner>"
        "  </D:lockinfo>"
        );
```

6.6.6 Unlock Object service

Synopsis

```
mdrib_unlock_object
(
    in session: mdrib_handle, // session handle
    in src_identifier: characterstring, // src object name
    in qualifier: characterstring, // unlock service qualifier
)
returns (state(success,failure)),
```

Description

Unlocks object src_identifier according to src_unlock specification. See RFC 4518 for the UNLOCK specification.

Example

```
// Note 1: See RFC 4518 for unlock specification
// Note 2: This example adapted from RFC 4518, subclause 9.11.2
// Locking the resource "0123/value_domain_datatype" ...
mdrib_unlock_object(mdrib_handle,
        "0123/value_domain_datatype",
        ""
        );
```

6.6.7 Copy Object service

Synopsis

```
mdrib_copy_object
(
    in session: mdrib_handle, // session handle
    in src_identifier: characterstring, // src object name
    in dst_identifier: characterstring, // dst object name
    in qualifier: characterstring, // copy service qualifier
)
returns (state(success,failure)),
```

Description

Duplicates the object src_identifier with the new name dst_identifer. See RFC 4518 for the COPY features.

Example

```
// Note 1: See RFC 4518 for copy specification
// Note 2: This example adapted from RFC 4518, subclause 9.8.7
// Copying resource "0123/value_domain_datatype" to "4567/value_domain_datatype"
// with no overwrite
mdrib_copy_object(mdrib_handle,
        "0123/value_domain_datatype",
        "4567/value_domain_datatype",
        "Overwrite: F",
        );
```

6.6.8 Move Object service

Synopsis

```
mdrib_move_object
(
    in session: mdrib_handle, // session handle
    in src_identifier: characterstring, // src object name
    in dst_identifier: characterstring, // dst object name
    in qualifier: characterstring, // move service qualifier
)
returns (state(success,failure)),
```

Description

Renames the object src_identifier with the new name dst_identifer. See RFC 4518 for the MOVE features.

Example

```
// Note 1: See RFC 4518 for move specification
// Note 2: This example adapted from RFC 4518, subclause 9.9.5
// Moving resource "0123/value_domain_datatype" to "4567/value_domain_datatype",
// i.e., renaming it
mdrib_move_object(mdrib_handle,
        "0123/value_domain_datatype",
        "4567/value_domain_datatype",
        "",
        );
```

7 Bindings

This part of ISO/IEC 20944 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.

8 Administration

There are no administrative requirements.

9 Conformance

9.1 API conformance paradigm

The conformance paradigm for ISO/IEC 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 environment 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.

9.4 Conformance labels

The following labels indicate certain kinds of conformity to this part of ISO/IEC 20944:

— Pattern 1: "ISO/IEC 20944-3/B/prefix" where "prefix" represents the kind of binding, as defined in Clauses 11 and onward. For example, "ISO/IEC 20944-3/B/C" corresponds to the C API binding.

— Pattern 2: ISO/IEC 20944-3/B/prefix/role" where "prefix" represents the kind of binding, as defined in Clauses 11 and onward, and "role" is one of "A" or "E" corresponding to API application and API environment, as defined in this Clause. For example, "ISO/IEC 20944-3/B/C/A" corresponds to an application using the C API binding.

The location of the placement of conformance labels is outside the scope of this International Standard.

10 Reserved for future standardization

This Clause is reserved for future standardization.

11 C API binding

The binding-independent API is mapped to the C programming language according to the following rules. The C API binding datatypes are based upon the C programming language.

11.1 Datatype mapping

The datatypes of binding-independent API are mapped as follows:

Binding-Independent	C API Binding
mdrib_handle	mdrib_handle
characterstring	string (maximum length 32767)
pointer	void *
str8	unsigned char []
str16	wchar_t[]
str32	wchar_t[]
int8, uint8, int16, uint16, int32, uint32, int64, uint64, int128, uint128	int8_t, uint8_t, int16_t, uint16_t, int32_t, uint32_t, int64_t, uint64_t, int128_t, uint128_t
datetime	time_t
duration	time_t
uri	unsigned char []
real32, real64, real80	float, double, long double
procedure	(void *)()

11.2 Function parameter mapping

The parameters of binding-independent API are the same as the parameters for the C binding.

11.3 Function return mapping

The return values of binding-independent API are mapped as follows:

Binding-Independent	C API Binding
state(success,failure)	int, success=0, failure=-1
characterstring	string (maximum length 65535)
str8	unsigned char []
str16	wchar_t[]
str32	wchar_t[]
int8, uint8, int16, uint16, int32, uint32, int64, uint64, int128, uint128	int8_t, uint8_t, int16_t, uint16_t, int32_t, uint32_t, int64_t, uint64_t, int128_t, uint128_t
datetime	time_t
duration	time_t
uri	unsigned char []
real32, real64, real80	float, double, long double

11.4 Function exception mapping

The function exceptions are mapped as follows:

Binding-Independent	C API Binding
bad_conversion	errno is set

11.5 Procedure call signatures

11.5.1 Session establishment services

11.5.1.1 System information

Synopsis

```
unsigned char *mdrib_system_info
(
    mdrib_handle session // session handle
)
```

11.5.1.2 Configure

Synopsis

```
int mdrib_configure
(
    mdrib_handle session, // session handle
    unsigned char *parameter_name, // set parameter "name"
    unsigned char *parameter_value // to the value "value"
)
```

11.5.1.3 Connect

Synopsis

```
mdrib_handle mdrib_connect
(
    unsigned char *target, // repository to connect to
    unsigned char *options // connect options
)
```

11.5.2 Disconnect

Synopsis

```
int mdrib_disconnect
(
    mdrib_handle session // session handle
)
```

11.5.2.1 Open

Synopsis

```
mdrib_handle mdrib_open
(
    mdrib_handle session, // session handle
    unsigned char *node, // portion of repository to open
    unsigned char *options // connect options
)
```

11.5.2.2 Close

Synopsis

```
int mdrib_close
(
    mdrib_handle session // session handle
)
```

11.5.3 Session parameter services

11.5.3.1 Get path

Synopsis

```
unsigned char *mdrib_get_path
(
    mdrib_handle session // session handle
)
```

11.5.4 Security services

11.5.4.1 Request Authorization/Authentication

Synopsis

```
int mdrib_request_auth
(
    mdrib_handle session, // session handle
    unsigned char *auth_type, // auth type
    unsigned char *auth_options // auth options
)
```

11.5.4.2 Response Authorization/Authentication

Synopsis

```
int mdrib_response_auth
(
    mdrib_handle session, // session handle
    unsigned char *auth_type, // auth type
    (*int)() auth_handler() // auth handler function
)
```

11.5.5 Data transfer services

11.5.5.1 Get value

Synopsis

```
int mdrib_get_value
(
    mdrib_handle session, // [in] session handle
    unsigned char *src_identifier, // [in] src object name
    unsigned char *dst_label_type, // [in] saved label: typeof
    mdrib_object *dst_label_ptr, // [out] saved label: ptr to
    unsigned char *dst_type_type, // [in] saved datatype: typeof
    mdrib_object *dst_type_ptr, // [out] saved datatype: ptr to
    unsigned char *dst_object_type, // [in] saved value: typeof
    mdrib_object *dst_object_ptr, // [out] saved value: ptr to
    unsigned char *dst_proplist_type, // [in] saved proplist: typeof
    mdrib_object *dst_proplist_ptr, // [out] saved proplist: ptr to
)
```

11.5.5.2 Typed get value

Synopsis

```
unsigned char *mdrib_get_value_as_str8
(
    mdrib_handle session, // [in] session handle
    unsigned char *src_identifier // [in] src object name
)

wchar_t *mdrib_get_value_as_str16
(
    mdrib_handle session, // [in] session handle
    unsigned char *src_identifier // [in] src object name
)
```

```
wchar_t *mdrib_get_value_as_str32
(
    mdrib_handle session, // [in] session handle
    unsigned char *src_identifier // [in] src object name
)

int8_t mdrib_get_value_as_int8
(
    mdrib_handle session, // [in] session handle
    unsigned char *src_identifier // [in] src object name
)

uint8_t mdrib_get_value_as_uint8
(
    mdrib_handle session, // [in] session handle
    unsigned char *src_identifier // [in] src object name
)

int16_t mdrib_get_value_as_int16
(
    mdrib_handle session, // [in] session handle
    unsigned char *src_identifier // [in] src object name
)

uint16_t mdrib_get_value_as_uint16
(
    mdrib_handle session, // [in] session handle
    unsigned char *src_identifier // [in] src object name
)

int32_t mdrib_get_value_as_int32
(
    mdrib_handle session, // [in] session handle
    unsigned char *src_identifier // [in] src object name
)

uint32_t mdrib_get_value_as_uint32
(
    mdrib_handle session, // [in] session handle
    unsigned char *src_identifier // [in] src object name
)

int64_t mdrib_get_value_as_int64
(
    mdrib_handle session, // [in] session handle
    unsigned char *src_identifier // [in] src object name
)

uint64_t mdrib_get_value_as_uint64
(
    mdrib_handle session, // [in] session handle
    unsigned char *src_identifier // [in] src object name
)

int128_t mdrib_get_value_as_int128
(
    mdrib_handle session, // [in] session handle
    unsigned char *src_identifier // [in] src object name
)
```

```
uint128_t mdrib_get_value_as_uint128
(
    mdrib_handle session, // [in] session handle
    unsigned char *src_identifier // [in] src object name
)

float mdrib_get_value_as_real32
(
    mdrib_handle session, // [in] session handle
    unsigned char *src_identifier // [in] src object name
)

double mdrib_get_value_as_real64
(
    mdrib_handle session, // [in] session handle
    unsigned char *src_identifier // [in] src object name
)

long double mdrib_get_value_as_real80
(
    mdrib_handle session, // [in] session handle
    unsigned char *src_identifier // [in] src object name
)
```

11.5.5.3 Put value

Synopsis

```
int mdrib_put_value
(
    mdrib_handle session, // [in] session handle
    unsigned char *src_identifier, // [in] src object name
    unsigned char *src_label_type, // [in] assigned label: typeof
    mdrib_object src_label_ptr, // [in] assigned label: ptr to
    unsigned char *src_type_type, // [in] assigned datatype: typeof
    mdrib_object src_type_ptr, // [in] assigned datatype: ptr to
    unsigned char *src_object_type, // [in] assigned value: typeof
    mdrib_object src_object_ptr, // [in] assigned value: ptr to
    unsigned char *src_proplist_type, // [in] assigned proplist: typeof
    mdrib_object src_proplist_ptr, // [in] assigned proplist: ptr to
)
```

11.5.5.4 Typed put value

Synopsis

```
int mdrib_put_value_as_str8
(
    mdrib_handle session, // [in] session handle
    unsigned char *dst_identifier, // [in] dst object name
    str8 src_value // [in] src value
)

int mdrib_put_value_as_str16
(
    mdrib_handle session, // [in] session handle
    unsigned char *dst_identifier, // [in] dst object name
    str16 src_value // [in] src value
)
```

```
int mdrib_put_value_as_str32
(
    mdrib_handle session, // [in] session handle
    unsigned char *dst_identifier, // [in] dst object name
    str32 src_value // [in] src value
)

int mdrib_put_value_as_int8
(
    mdrib_handle session, // [in] session handle
    unsigned char *dst_identifier, // [in] dst object name
    int8 src_value // [in] src value
)

int mdrib_put_value_as_uint8
(
    mdrib_handle session, // [in] session handle
    unsigned char *dst_identifier, // [in] dst object name
    uint8 src_value // [in] src value
)

int mdrib_put_value_as_int16
(
    mdrib_handle session, // [in] session handle
    unsigned char *dst_identifier, // [in] dst object name
    int16 src_value // [in] src value
)

int mdrib_put_value_as_uint16
(
    mdrib_handle session, // [in] session handle
    unsigned char *dst_identifier, // [in] dst object name
    uint16 src_value // [in] src value
)

int mdrib_put_value_as_int32
(
    mdrib_handle session, // [in] session handle
    unsigned char *dst_identifier, // [in] dst object name
    int32 src_value // [in] src value
)

int mdrib_put_value_as_uint32
(
    mdrib_handle session, // [in] session handle
    unsigned char *dst_identifier, // [in] dst object name
    uint32 src_value // [in] src value
)

int mdrib_put_value_as_int64
(
    mdrib_handle session, // [in] session handle
    unsigned char *dst_identifier, // [in] dst object name
    int64 src_value // [in] src value
)

int mdrib_put_value_as_uint64
(
    mdrib_handle session, // [in] session handle
    unsigned char *dst_identifier // [in] dst object name
    unit64 src_value // [in] src value
)
```

```
int mdrib_put_value_as_int128
(
    mdrib_handle session, // [in] session handle
    unsigned char *dst_identifier, // [in] dst object name
    int128 src_value // [in] src value
)

int mdrib_put_value_as_uint128
(
    mdrib_handle session, // [in] session handle
    unsigned char *dst_identifier, // [in] dst object name
    uint128_t src_value // [in] src value
)

int mdrib_put_value_as_real32
(
    mdrib_handle session, // [in] session handle
    unsigned char *dst_identifier, // [in] dst object name
    float src_value // [in] src value
)

int mdrib_put_value_as_real64
(
    mdrib_handle session, // [in] session handle
    unsigned char *dst_identifier, // [in] dst object name
    double src_value // [in] src value
)

int double mdrib_put_value_as_real80
(
    mdrib_handle session, // [in] session handle
    unsigned char *dst_identifier, // [in] dst object name
    long double src_value // [in] src value
)
```

11.5.6 Miscellaneous

11.5.6.1 Make Object service

Synopsis

```
integer mdrib_make_object
(
    mdrib_handle session, // [in] session handle
    unsigned char *src_identifier, // [in] src object name
    unsigned char *src_label_type, // [in] assigned label: typeof
    mdrib_object src_label_ptr, // [in] assigned label: ptr to
    unsigned char *src_type_type, // [in] assigned datatype: typeof
    mdrib_object src_type_ptr, // [in] assigned datatype: ptr to
    unsigned char *src_object_type, // [in] assigned value: typeof
    mdrib_object src_object_ptr, // [in] assigned value: ptr to
    unsigned char *src_proplist_type, // [in] assigned proplist: typeof
    mdrib_object src_proplist_ptr, // [in] assigned proplist: ptr to
)
```

11.5.6.2 Delete Object service

Synopsis

```
integer mdrib_delete_object
(
    mdrib_handle session, // [in] session handle
    unsigned char *src_identifier, // [in] src object name
    unsigned char *qualifier // [in] delete service qualifier
)
```

11.5.6.3 Link Object service

Synopsis

```
integer mdrib_link_object
(
    mdrib_handle session, // [in] session handle
    unsigned char *src_identifier, // [in] src object name
    unsigned char *dst_identifier, // [in] dst object name
    unsigned char *qualifier // [in] link type: soft, hard
)
```

11.5.6.4 Property Find service

Synopsis

```
characterstring mdrib_property_find
(
    mdrib_handle session, // [in] session handle
    unsigned char *src_identifier, // [in] src object name (wildcard)
    unsigned char *qualifier // [in] propfind service qualifier
)
```

11.5.6.5 Lock Object service

Synopsis

```
integer mdrib_lock_object
(
    mdrib_handle session, // [in] session handle
    unsigned char *src_identifier, // [in] src object name
    unsigned char *qualifier // [in] lock service qualifier
)
```

11.5.6.6 Unlock Object service

Synopsis

```
integer mdrib_unlock_object
(
    mdrib_handle session, // [in] session handle
    unsigned char *src_identifier, // [in] src object name
    unsigned char *qualifier // [in] unlock service qualifier
)
```

11.5.6.7 Copy Object service

Synopsis

```
integer mdrib_copy_object
(
    mdrib_handle session, // [in] session handle
    unsigned char *src_identifier, // [in] src object name
    unsigned char *dst_identifier, // [in] dst object name
    unsigned char *qualifier // [in] copy service qualifier
)
```

11.5.6.8 Move Object service

Synopsis

```
integer mdrib_move_object
(
    mdrib_handle session, // [in] session handle
    unsigned char *src_identifier, // [in] src object name
    unsigned char *dst_identifier, // [in] dst object name
    unsigned char *qualifier // [in] move service qualifier
)
```

11.6 Error/exception returns

The following error/exception returns are passed via the errno variable:

— MDRIB_ERR_ACCESS: Permission denied executing operation

— MDRIB_ERR_AGAIN: Resource currently unavailable in specified mode due to locking mechanism, try again later.

— MDRIB_ERR_BADH: Invalid session or connection handle

— MDRIB_ERR_INVAL: The value of one of the options contains illegal syntax, or the target paramter syntax is invalid.

— MDRIB_ERR_IO: I/O Error when executing operation

— MDRIB_ERR_NOENT: The specified target or path does not exist.

— MDRIB_ERR_NOMEM: Insufficient memory to open connection

— MDRIB_ERR_NOSPC: No more space to create specified repository.

— MDRIB_ERR_NOTOBJ: A component of the path does not point to a valid object

— MDRIB_ERR_PERM: Operation not permitted due to access violation

— MDRIB_ERR_NOSUP: Operation not supported

— MDRIB_ERR_TIMEDOUT: Time out while executing operation

— MDRIB_ERR_MAXHANDLE: Maximum number of open connections allowed reached

— MDRIB_WRN_INVAL: One of the parameters passed was invalid and was ignored

11.7 Conformance label prefix

The prefix "C" shall be used, in conjunction with 9.4, to indicate conformity to this Clause.

12 Java API binding

The binding-independent API is mapped to the Java programming language according to the following rules. The API services are methods of the mdrib_handle class. The Java API binding datatypes are based upon the Java programming language.

12.1 Datatype mapping

The datatypes are mapped as follows:[7]

Binding-Independent	Java API Binding
mdrib_handle	mdrib_handle
characterstring	String
pointer	reference
str8	String
str16	String
str32	--
int8, uint8, int16, uint16, int32, uint32, int64, uint64, int128, uint128	byte, --, short, --, int, --, long, --, --, --
datetime	time_t
duration	time_t
uri	String
real32, real64, real80	float, double, --
procedure	--

12.2 Function parameter mapping

The parameters of the Binding-Independent API are the same as the parameters for the Java API binding.

7 The symbol "--" means there is no corresponding mapping.

12.3 Function return mapping

The return values are mapped as follows:

Binding-Independent	Java API Binding
state(success,failure)	int, success=0, failure=-1
characterstring	String
str8	String
str16	String
str32	String
int8, uint8, int16, uint16, int32, uint32, int64, uint64, int128, uint128	byte, --, short, --, int, --, long, --, --, --
datetime	time_t
duration	time_t
uri	unsigned char []
real32, real64, real80	float, double, --

12.4 Function exception mapping

The function exceptions are mapped as follows:

Binding-Independent	Java API Binding
bad_conversion	exception thrown

12.5 Procedure call signatures

12.5.1 Session establishment services

Synopsis

```
/*
 * This class represents a high level connection to
 * a Metadata Registry database. From this connection
 * sessions to the actual data can be retrieved.
 */

package org.iso.mdrib;

import java.util.Properties;

/**
 *
 */
```

```
public interface MDRConnection {


    /** Opens and creates a session  starting
     *  at the specified node with the specified options.
     *
     *  @returns A valid MDRSession, or null in case of
     *  error.
     */
    public MDRSession open(String node, Properties options) throws MDRException;

    /**
     * Closes the specified session
     */
    public void close(MDRSession session) throws MDRException;

    /**
     * Authenticates the specified session.
     */
    public boolean authenticate(MDRSession session, String authType, Properties
options) throws MDRException;

    /**
     * Configures either the current connection, or the session
     */
    public void setConfiguration(MDRSession session, String key, String val) throws
MDRException;

    /**
     * Returns the configuration of the system as a Properties class
     */
    public Properties getConfiguration(MDRSession session) throws MDRException;

}
```

12.5.2 Data transfer services

Synopsis

```
/*
 * This contains an interface that must be implemented by
 * the different low-level accessors to the MDR API. It contains
 * the API to access the data for a session.
 */

package org.iso.mdrib;

import java.net.URL;
import java.sql.Date;
import java.sql.Time;
import java.sql.Timestamp;
import java.math.BigDecimal;


public interface MDRSession {

  /**
   * Lists all objects that match the mask.
   *
   * @returns An array of identifiers in the current
```

```
     * active path (ot absolute path) that match the specified
     * mask.
     */
    String[] list(String mask) throws MDRException;

    /**
     * Links an existing object to a new name.
     *
     */
    void link(String srcIdentifier, String dstIdentifier, String qualifier)
      throws MDRException;

    /**
     * Deletes an object.
     *
     */
    void delete(String srcIdentifier, String qualifier) throws MDRException;

    /**
     * Creates an object
     *
     */
    //void create(String src_identifier) throws MDRException;

    /**
     * Locks an object.
     *
     */
    void lock(String srcIdentifier, String qualifier) throws MDRException;

    /**
     * Unlocks an object.
     *
     */
    void unlock(String srcIdentifier, String qualifier) throws MDRException;

    /**
     * Copies an object.
     *
     */
    void copy(String srcIdentifier, String, dstIdentifier, String qualifier)
            throws MDRException;

    /**
     * Moves an object.
     *
     */
    void move(String srcIdentifier, String, dstIdentifier, String qualifier)
            throws MDRException;

    /**
     *  Retrieves the value of the designated object as a
     *  java.math.BigDecimal with full precision.
     */
   BigDecimal getBigDecimal(String srcIdentifer)  throws MDRException;

   /**
    *  Retrieves the value of the designated object
    *  as a boolean in the Java programming language.
    */
```

```
boolean getBoolean(String srcIdentifier) throws MDRException;

/**
 * Retrieves the value of the designated object as a byte
 * in the Java programming language.
 */
byte getByte(String srcIdentifier)  throws MDRException;

/**
 * Retrieves the value of the designated object as a byte array
 * in the Java programming language.
 */
byte[] getBytes(String srcIdentifier)   throws MDRException;

/**
 * Retrieves the value of the designated object as a byte array
 * in the Java programming language using offsets and maximum lengths.
 *
 * @returns The number of bytes actually transferred,
 */
int getBytes(String srcIdentifier, byte[] dst, int srcOffet, int dstOffset, int
count)   throws MDRException;

/**
 * Retrieves the value of the designated object as a java.sql.Date
 * in the Java programming language.
 */
Date getDate(String srcIdentifier)  throws MDRException;

/**
 * Retrieves the value of the designated object as a double
 * in the Java programming language.
 */
double getDouble(String srcIdentifier)  throws MDRException;

/**
 * Retrieves the value of the designated object as a float
 * in the Java programming language.
 */
float getFloat(String srcIdentifier)  throws MDRException;
/**
 * Retrieves the value of the designated object as an int
 * in the Java programming language.
 */
int getInt(String srcIdentifier)  throws MDRException;
/**
 * Retrieves the value of the designated object as a long
 * in the Java programming language.
 */
long getLong(String srcIdentifier)  throws MDRException;
/**
 * Retrieves the value of the designated object as a short
 * in the Java programming language.
 */
short getShort(String srcIdentifier)  throws MDRException;
/**
 * Retrieves the value of the designated object as a String
 * in the Java programming language.
 */
String getString(String srcIdentifier)  throws MDRException;
```

```
/**
 * Retrieves the value of the designated object as a java.sql.Time
 * in the Java programming language.
 */
Time getTime(String srcIdentifier)  throws MDRException;
/**
 * Retrieves the value of the designated object as a java.sql.Timestamp
 * in the Java programming language.
 */
Timestamp getTimestamp(String srcIdentifier)  throws MDRException;
/**
 * Retrieves the value of the designated object as a java.net.URL
 * in the Java programming language.
 */
URL getURL(String srcIdentifier)  throws MDRException;

/**
 *  Updates the designated object with a java.sql.BigDecimal value.
 */
void updateBigDecimal(String dstIdentifier, BigDecimal x) throws MDRException;

/**
 *  Updates the designated object with a boolean value.
 */
void updateBoolean(String dstIdentifier, boolean x)  throws MDRException;

/**
 *  Updates the designated object with a byte value.
 */
void updateByte(String dstIdentifier, byte x)  throws MDRException;
/**
 * Updates the value of the designated object with the specified byte array
 * in the Java programming language using offsets and maximum lengths.
 *
 * @returns The actual number of bytes transferred.
 */
int updateBytes(byte[] src, String dstIdentifier, int srcOffet, int dstOffset, int
count)   throws MDRException;

/**
 *  Updates the designated object with a byte array value.
 */
void updateBytes(String dstIdentifier, byte[] x)  throws MDRException;
/**
 *  Updates the designated object with a java.sql.Date value.
 */
void updateDate(String dstIdentifier, Date x)  throws MDRException;
/**
 *  Updates the designated object with a double value.
 */
void updateDouble(String dstIdentifier, double x)  throws MDRException;
/**
 *  Updates the designated object with a float value.
 */
void updateFloat(String dstIdentifier, float x)  throws MDRException;
/**
 *  Updates the designated object with an int value.
 */
void updateInt(String dstIdentifier, int x)  throws MDRException;
/**
```

```
     *  Updates the designated object with a long value.
     */
    void updateLong(String dstIdentifier, long x)  throws MDRException;
    /**
     *  Updates the designated object with a null value.  ???
     */
    void updateNull(String dstIdentifier)  throws MDRException;
    /**
     *  Updates the designated object with a short value.
     */
    void updateShort(String dstIdentifier, short x)  throws MDRException;
    /**
     *  Updates the designated object with a String value.
     */
    void updateString(String dstIdentifier, String x)  throws MDRException;
    /**
     *  Updates the designated object with a java.sql.Time value.
     */
    void updateTime(String dstIdentifier, Time x)  throws MDRException;
    /**
     *  Updates the designated object with a java.sql.Timestamp value.
     */
    void updateTimestamp(String dstIdentifier, Timestamp x)  throws MDRException;

    /**
     *  Sets the current object node path.
     */
    void setCurrentPath(String node) throws MDRException;

    /**
     *  Retrieves the current object node path.
     */
    String getCurrentPath() throws MDRException;

}
```

12.5.3 Exception class

Synopsis

```
/*
 * This implements an exception class that can be thrown
 * by any of the interfaces and classes in this package.
 *
 */

package org.iso.mdrib;

/**
 *
 */
public class MDRException extends Exception {

    /** Error status: Resource currently not available in specified
        mode due to locking mechanism, try again later
     */
    public static final short MDRIB_ERR_AGAIN = (short)-1;
    // ... Other error codes should be defined here...
```