API Function Descriptions for the C Wrapper

This chapter describes the API Functions available for the C Wrapper.

- API Function Descriptions
- API Function Descriptions for Variable-length Data Types AV, BV, KV and UV
- API Function Descriptions for Unbounded Arrays
- API Function Descriptions for Reliable RPC

API Function Descriptions

The API for the RPC C runtime is defined in the following header file:

```
#include <erx.h>
```

ERXCall

Remote Procedure Call - Conversationless or Conversational.

Syntax

```
extern ERXeReturnCode ERXAPI ERXCall(
        ERX_CLIENT_IDENTIFICATION  ERXPTR *pClient,
        const ERX_SERVER_ADDRESS   ERXPTR *pServer,
        ERXCallId                  ERXPTR *pCallId,
        const ERX_CALL_INFORMATION_BLOCK ERXPTR *pCallInfoBlock,
        void                       ERXPTR *(ERXPTR pParameterBlock)[],
        const ERXeControlFlags             fFlags);
```

Description

This performs a remote procedure call. It is used for both connectionless and connection-oriented calls:

- **Connectionless Calls**
  The server is identified by the `&Server` parameter

```
ERXCall(&Client,&Server,&CallId,&CIB,&Parameter,ERX_CF_NOTHING|ERX_CF_STRUCTURED);
```

- **Connection-oriented Calls**
  The server is identified by referring to an established connection: `&Server_Connection` parameter returned by an `ERXConnect` call, as follows:

```
ERX_SERVER_ADDRESS Server_Connection
...
ERXConnect(&Client,&Server,&Server_Connection);
ERXCall(NULL,&Address,&CallId,&CIB,&Parameter,ERX_CF_NOTHING|ERX_CF_STRUCTURED);
...
ERXDisconnectCommit(&Address);
```

We suggest using `ERX_TM_BROKER_LIBRARY` as the medium of the server address (see `ERX_SERVER_ADDRESS`). Appropriate values must be provided for all fields. See also `ERXConnect`.

The called procedure is identified by the call information block (see `ERX_CALL_INFORMATION_BLOCK`), which contains its name and location (library). The call information block also points to an array of parameter definitions (`ERX_PARAMETER_DEFINITION_V3`). The parameter definition contains the type, size, count of indices, occurrences in the respective dimensions, the addresses of the parameters, etc. The `pParameterBlock` array contains the pointers to each parameter's data.

The Software AG IDL Compiler (with the template files *client.tpl* and *server.tpl*) generates interface objects in which the parameter data is grouped in consecutive storage. This is referred to as *structured mode* and is indicated by specifying the `ERX_CF_STRUCTURED` flag as part of the `fFlags` `ERXCall`. In structured mode, only one address, the address of the structure, is passed in the `pParameterBlock` array. That is, the `pParameterBlock` array only has one entry.

For information on the messages, see *Error Messages and Codes*.

Parameters

pClient

in out: The client's identification, see `ERX_CLIENT_IDENTIFICATION`.

pServer

in: The address of the server, see `ERX_SERVER_ADDRESS`.

pCallId

out: The `CallId` returned to the caller, used in asynchronous communication to receive the reply with the `ERXWait` API call.

pCallInfoBlock

in: The description of the program to call and its parameter definition, see `ERX_CALL_INFORMATION_BLOCK`

pParameterBlock

in out: The array of pointers to the actual parameter data.

fFlags

in: `ERX_CF_STRUCTURED`, i.e. the parameters are collected in one data structure.

Return Codes

Value	Meaning
00000000	ERX_S_SUCCESS
00010008	ERX_E_NOT_REGISTERED
00020002	ERX_ETB_USER_DOES_NOT_EXIST
0003nnnn	ERX_ETB_CONVERSATION_ENDED
00070007	ERX_ETB_SERVICE_NOT_AVAILABLE
00740074	ERX_ETB_WAIT_TIMEOUT
02150148	ERX_ETB_BROKER_NOT_AVAILABLE

Related Functions

ERXConnect
ERXDisconnect
ERXDisconnectCommit

ERX_Callback_SERVER_CALL

Syntax

```
void   ERX_Callback_SERVER_CALL (
       void                            *pUserInfo,
       ERX_CLIENT_IDENTIFICATION       *pClientInformation,
       ERX_CALL_INFORMATION_BLOCK      *pCallInformation,
       void                            *pParameterArea,
       ERX_ERROR_INFO                  *pReturnInfo
);
```

Description

This callback function (see *Writing the Callback*) is called by the Callable RPC Server (see *Writing Callable RPC Servers with the C Wrapper*). Success or failure is returned with the structure ERX_ERROR_INFO.

Parameters

***pUserInfo**

in: User specific data. The data is provided "as is" in the function ERXServingCallback. It can be used to provide a pointer to a memory location with user specific data in the callback.

***pClientInformation**

in: The client’s identification such as user ID, etc., see ERX_CLIENT_IDENTIFICATION.

***pCallInformation**

in: The description of the library and program to call and its parameter definition, see ERX_CALL_INFORMATION_BLOCK

***pParameterArea**

in: IDL in and inout Parameters from the client. Upon return IDL out and inout parameters are replied back to the client. Parameters are provided and expected in a contiguous memory location.

***pReturnInfo**

in: Returning success or errors from the callback function. Possible Return Codes to give back are:
`ERX_E_RPC_LIBRARY_NOT_FOUND`
`ERX_E_RPC_CALLEE_NOT_FOUND`
`ERX_E_RPC_OUT_OF_MEMORY`
`ERX_E_RPC_ABNORMAL_TERMINATION`
`ERX_S_SUCCESS`

Related Functions

`ERXRegisterEvent`
`ERXServingCallback`

ERXConnect

Establish a conversation to the named server.

Syntax

```
extern ERXeReturnCode ERXAPI ERXConnect(
       ERX_CLIENT_IDENTIFICATION  ERXPTR *pClient,
       const ERX_SERVER_ADDRESS   ERXPTR *pServer,
       ERX_SERVER_ADDRESS         ERXPTR *pAddress
);
```

Description

Establishes an RPC conversation (connection) to the named server.

The information supplied covers the identification of the client, for example user ID and password, and the server address.

We suggest using `ERX_TM_BROKER_LIBRARY` as the medium of the server address. Appropriate values must be provided for all fields.

See *Using Conversational RPC* for more information.

For information on the messages, see *Error Messages and Codes*.

Parameters

pClient

in out: The client's identification, see `ERX_CLIENT_IDENTIFICATION`.

pServer

in: The address of the server, see `ERX_SERVER_ADDRESS`.

pAddress

out: The connection ID returned to the caller. The pointers `pServer` and `pAddress` must not be the same. Otherwise an error will occur.

Return Codes

Value	Meaning
00000000	ERX_S_SUCCESS
00010008	ERX_E_NOT_REGISTERED
00020002	ERX_ETB_USER_DOES_NOT_EXIST
00070007	ERX_ETB_SERVICE_NOT_AVAILABLE
00740074	ERX_ETB_WAIT_TIMEOUT
02150148	ERX_ETB_BROKER_NOT_AVAILABLE

Related Functions

`ERXDisconnect`
`ERXDisconnectCommit`

ERXDisconnect

Give up the conversation with Backout.

Syntax

```
extern ERXeReturnCode ERXAPI ERXDisconnect(
       ERX_SERVER_ADDRESS         ERXPTR *pAddress
);
```

Description

Aborts the specified RPC conversation (connection). In contrast to `ERXDisconnectCommit`, calling this function leads to an abnormal, unsuccessful end of the RPC Conversation. See *Using Conversational RPC* for more information.

For information on the messages, see *Error Messages and Codes*.

Parameters

pAddress

in: The pointer to the connection to the RPC Server that is to be aborted. The connection ID contained in the `ERX_SERVER_ADDRESS` data structure was retrieved by a previous `ERXConnect` call.

Return Codes

Value	Meaning
`00000000`	`ERX_S_SUCCESS`
`00010008`	`ERX_E_NOT_REGISTERED`
`00020002`	`ERX_ETB_USER_DOES_NOT_EXIST`
`00070007`	`ERX_ETB_SERVICE_NOT_AVAILABLE`
`00740074`	`ERX_ETB_WAIT_TIMEOUT`
`02150148`	`ERX_ETB_BROKER_NOT_AVAILABLE`

Related Functions

`ERXConnect`
`ERXDisconnectCommit`

ERXDisconnectCommit

Give up the conversation with Commit.

Syntax

```
extern ERXeReturnCode ERXAPI ERXDisconnectCommit(
       ERX_SERVER_ADDRESS         ERXPTR *pAddress
);
```

Description

Close the specified RPC conversation (connection). In contrast to `ERXDisconnect`, calling this function leads to a normal, successful end of the RPC Conversation. See *Using Conversational RPC* for more information.

For information on the messages, see *Error Messages and Codes*.

Parameters

pAddress

in: The pointer to the connection to the RPC Server to close. The connection ID contained in the `ERX_SERVER_ADDRESS` data structure was retrieved by a previous `ERXConnect` call.

Return Codes

Value	Meaning
00000000	ERX_S_SUCCESS
00010008	ERX_E_NOT_REGISTERED
00020002	ERX_ETB_USER_DOES_NOT_EXIST
00070007	ERX_ETB_SERVICE_NOT_AVAILABLE
00740074	ERX_ETB_WAIT_TIMEOUT
02150148	ERX_ETB_BROKER_NOT_AVAILABLE

Related Functions

`ERXConnect`
`ERXDisconnect`

ERXGetBrokerSecurity

Get the current setting of broker kernel security value.

Syntax

```
extern ERXeReturnCode ERXAPI ERXGetBrokerSecurity(
       char                  * pKernelSecurity
);
```

Description

With this function you retrieve the current settings for security set by a previously issued `ERXSetBrokerSecurity` function call maintained internally by the RPC C runtime on a per-thread basis. See *Using EntireX Security* for more information. For information on the messages, see *Error Messages and Codes*.

Parameters

pKernelSecurity

in

Return Codes

Value	Meaning
00000000	ERX_S_SUCCESS
00010008	ERX_E_NOT_REGISTERED

Related Functions

`ERXSetBrokerSecurity`

ERXGetCodepage

Get the current setting of the codepage.

Syntax

```
extern ERXeReturnCode ERXAPI ERXGetCodepage(
       char szCodepage [ERX_MAX_CODEPAGE_LENGTH + 1]
);
```

Description

With this function you retrieve the current settings for the locale string set by a previously issued `ERXSetCodepage` function call maintained internally by the RPC C runtime on a per-thread basis. See *Using Internationalization with the C Wrapper* for more information. For information on the messages, see *Error Messages and Codes*.

Parameters

szCodepage

in

Return Codes

Value	Meaning
00000000	ERX_S_SUCCESS
00010008	ERX_E_NOT_REGISTERED

Related Functions

`ERXSetCodepage`

ERXGetContext

Get the current assigned context.

Syntax

```
extern ERXeReturnCode ERXAPI ERXGetContext(
       ERX_CONTEXT_BLOCK          ERXPTR ** ppContextBlock
);
```

Description

This function supports RPC clients in multithreaded environments. It is used to retrieve (thread-safe) RPC and broker context information, which was supplied by a preceding `ERXSetContext` call. See *Programming Multithreaded RPC Clients*.

For information on the messages, see *Error Messages and Codes*.

Parameters

ppContextBlock

in out: Pointer to context block, see `ERX_CONTEXT_BLOCK`

Return Codes

Value	Meaning
00000000	ERX_S_SUCCESS
00010008	ERX_E_NOT_REGISTERED

Related Functions

`ERXSetContext`

ERXGetIAFToken

Get the IAF token.

Syntax

```
extern ERXeReturnCode ERXAPI ERXGetIAFToken(
       char                  sIAFToken[ERX_IAF_TOKEN_LENGTH]
);
```

Description

With this function you can programmatically get the IAF Token.

Parameters

sIAFToken

out

Return Codes

Value	Meaning
00000000	ERX_S_SUCCESS
00010008	ERX_E_NOT_REGISTERED

Related Functions

`ERXSetIAFToken`

ERXGetLastError

Get Information on Return Codes.

Syntax

```
extern ERXeReturnCode ERXAPI ERXGetLastError(
       ERX_ERROR_INFO             ERXPTR *pErrorInfo
);
```

Description

Retrieve information about the error that occurred last (the status of the last executed RPC C runtime function). When an API function is called, the error information is reset and, in case of error, the applicable information is placed in the error information structure. Exceptions to this rule are the functions `ERXRegister` and `ERXUnregister`, which only return the `ERXeReturnCode`. If `ERXGetLastError` itself is erroneous, the error information structure will be empty. For information on the messages, see *Error Messages and Codes*.

Parameters

pErrorInfo

out: A pointer to the data structure receiving the error information, see `ERX_ERROR_INFORMATION`.

Return Codes

Value	Meaning
00000000	ERX_S_SUCCESS
00010008	ERX_E_NOT_REGISTERED

Related Functions

`ERXGetMessage`

ERXGetMessage

Get Message Text.

Syntax

```
extern int ERXGetMessage(
       ERXeReturnCode  rc,
       char           *szMsg,
       unsigned int    ulMsg
);
```

Description

The function `ERXGetMessage` delivers the message text of the error codes in the following error classes. See also *Error Messages and Codes*:

- *Message Class 0001 - RPC C Runtime*
- *Message Class 1000 - RPC C Runtime System*
- *Message Class 1001 - RPC Protocol*
- *Message Class 1003 - Conversion*
- *Message Class 1004 - IDL Compiler*
- *Message Class 1005 - RPC Server*
- *Message Class 1006 - DCOM Wrapper*
- *Message Class 1008 - EntireX License*

Message texts from other error classes cannot be retrieved with this function. Use `ERXGetMessage` only to access errors from the error classes listed above. To always retrieve the correct error message after an C Wrapper API function call (`ERX` call), use the function `ERXGetLastMessage`.

Parameters

rc

in: ID of the message text to retrieve.

szMsg

out: Pointer to message text buffer.

ulMsg

in: Length of message text buffer.

Return Codes

Value	Meaning
`int==0`	OK
`int!=0`	something has failed.

Related Functions

`ERXGetLastError`

ERXGetSecurityToken

Get the current setting of the Security Token.

Syntax

```
extern ERXeReturnCode ERXAPI ERXGetSecurityToken(
       char                      szSecurityToken
                                 [ERX_MAX_securityToken_LENGTH + 1]
);
```

Description

Returns the current value of the Broker's Security Token maintained internally by the RPC C runtime on a per-thread basis. See *Using EntireX Security* for more information.

For information on the messages, *Error Messages and Codes*.

Parameters

szSecurityToken

out: The security token returned

Return Codes

Value	Meaning
00000000	ERX_S_SUCCESS
00010008	ERX_E_NOT_REGISTERED

Related Functions

ERXSetSecurityToken

ERXGetTraceLevel

Get the current trace level setting of the RPC C runtime and the broker stub.

Syntax

```
extern ERXeReturnCode ERXAPI ERXGetTraceLevel(
       long                 *puTraceLevel
);
```

Description

With this function you can retrieve the current trace level setting of the RPC C runtime and the broker stub.

Parameters

puTraceLevel

out

Return Codes

Value	Meaning
`00000000`	`ERX_S_SUCCESS`
`00010008`	`ERX_E_NOT_REGISTERED`

Related Functions

`ERXSetTraceLevel`

ERXGetVersion

Determine Version of RPC C runtime.

Syntax

```
extern int ERXGetVersion(
       char   *pMessage,
       size_t  uMessageLength
);
```

Description

Determine version of RPC C runtime. See *Examine the RPC Runtime and Interface Object Version* for more information.

Parameters

pMessage

in out: Pointer to buffer for version string.

uMessageLength

in: Length of buffer

Return Codes

Value	Meaning
`int==0`	OK
`int!=0`	something has failed.

ERXIsServing

Ping the Server.

Syntax

```
extern ERXeReturnCode ERXAPI ERXIsServing(
       ERX_CLIENT_IDENTIFICATION  ERXPTR *pClient,
       ERX_SERVER_ADDRESS         ERXPTR *pAddress,
       ERX_IS_SERVING             ERXPTR *pIsServing
);
```

Description

Check whether the server is available. Before issuing `ERXIsServing`, you must provide the following:

- the client identification
- the server address
- a pointer to the message area
- the length of the message area

For information on the messages, see *Error Messages and Codes*.

Parameters

pClient

in out: Pointer to the client identification, see `ERX_CLIENT_IDENTIFICATION`.

pAddress

in: Pointer to the server address, see `ERX_SERVER_ADDRESS`.

pIsServing

in out: Pointer to `ERX_IS_SERVING` structure.

Return Codes

Value	Meaning
00000000	ERX_S_SUCCESS
00010008	ERX_E_NOT_REGISTERED
00020002	ERX_ETB_USER_DOES_NOT_EXIST
0003nnnn	ERX_ETB_CONVERSATION_ENDED
00070007	ERX_ETB_SERVICE_NOT_AVAILABLE
00740074	ERX_ETB_WAIT_TIMEOUT
02150148	ERX_ETB_BROKER_NOT_AVAILABLE

ERXLogoff

Logoff by Broker and EntireX Security, i.e. free Broker resources.

Syntax

```
extern ERXeReturnCode ERXAPI ERXLogoff(
       ERX_CLIENT_IDENTIFICATION  ERXPTR *pClient,
       char szEtbidName [ERX_BROKER_ETBID_NAME_LENGTH + 1]
);
```

Description

Logs off from the Broker, frees the resources within the Broker and makes them available to other users. For information on the messages, see *Error Messages and Codes*.

Parameters

pClient

in out: The client's identification, see `ERX_CLIENT_IDENTIFICATION`

szEtbidName

in: Identification of the Broker. Correponds to the `BROKER-ID` field of the Broker ACI control block.

Return Codes

Value	Meaning
00000000	ERX_S_SUCCESS
00010008	ERX_E_NOT_REGISTERED
00020002	ERX_ETB_USER_DOES_NOT_EXIST
02150148	ERX_ETB_BROKER_NOT_AVAILABLE

Related Functions

`ERXLogon`

ERXLogon

Logon by EntireX Broker and EntireX Security.

Syntax

```
extern ERXeReturnCode ERXAPI ERXLogon(
       ERX_CLIENT_IDENTIFICATION  ERXPTR *pClient,
       char                               szEtbidName
                                          [ERX_BROKER_ETBID_NAME_LENGTH + 1]
);
```

Description

Logon to Broker.

This function allows the client or server application to logon to Broker, which allocates the necessary structures to handle the new participant. If the Broker is running in a secure environment, `ERXLogon` performs the authentication process. Whether `ERXLogon` is required depends on the customization of the Broker. See `AUTOLOGON` in the Broker attribute file and `cForceLogon` in the `ERX_CLIENT_IDENTIFICATION` structure.

We suggest using `ERXLogon` and `ERXLogoff` to logon/logoff to/from Broker.

For information on the messages, see *Error Messages and Codes*.

Parameters

pClient

in out: The client's identification, see `ERX_CLIENT_IDENTIFICATION`

szEtbidName

in: Identification of the Broker. Correponds to the `BROKER-ID` field of the Broker ACI control block.

Return Codes

Value	Meaning
00000000	ERX_S_SUCCESS
00010008	ERX_E_NOT_REGISTERED
02150148	ERX_ETB_BROKER_NOT_AVAILABLE

Related Functions

`ERXLogoff`

ERXRegister

Prepare use of RPC C runtime.

Syntax

```
extern ERXeReturnCode ERXAPI ERXRegister(
       const unsigned long                ulVersionRequested
);
```

Description

Register with RPC C runtime. Any thread within a process requiring RPC C runtime must register with it. When the RPC C runtime is no longer needed, any registered thread should unregister itself (see `ERXUnregister`). See *Using the RPC Runtime* for more information. For information on the messages see *Error Messages and Codes*.

Parameters

ulVersionRequested

in: The RPC C runtime version to be used. (See `ERX_CURRENT_VERSION` in *erx.h* for the most recent version). If the version is not supported, an error will occur.

Return Codes

Value	Meaning
00000000	ERX_S_SUCCESS
00010007	ERX_E_ALREADY_REGISTERED

Related Functions

`ERXUnregister`

ERXRegisterEvent

Syntax

```
extern ERXeReturnCode ERXAPI ERXRegisterEvent(
       const long                         EventId,
       void                            (* Callback)()
);
```

Description

The function registers events to the RPC C runtime used by the callable RPC Server during execution of the `ERXServingCallback` function. All events must be registered prior to the execution of the `ERXServingCallback` function. The following events are supported:

Event ID	Callback Prototype	Description
`ERX_EVENT_SERVER_CALL`	`ERX_Callback_SERVER_CALL`	Event for calling the server.

See *Writing the Callback* for more information. For information on the messages, see *Error Messages and Codes*.

Parameters

EventID

in: Event to register

(* Callback)()

in: Callback function belonging to the event

Return Codes

Value	Meaning
`00000000`	`ERX_S_SUCCESS`
`00010008`	`ERX_E_NOT_REGISTERED`

Related Functions

`ERX_Callback_SERVER_CALL`
`ERXServingCallback`

ERXReset

Reset a parameter block.

Syntax

```
extern ERXeReturnCode ERXAPI ERXReset(
 const ERX_CALL_INFORMATION_BLOCK ERXPTR *pCallInfoBlock,
       void                      ERXPTR *(ERXPTR pParameterBlock)[ ],
       ERXeParameterDirection            eDirection,
       const ERXeControlFlags            fFlags
);
```

Description

Reset the specified program parameters:

Software AG IDL	Value
A, AV, K, KV	blank
B, BV, I, F	0
D	1.1.1582
T	0:00, the date portion is reset as type D
N	+0
P	+0

For information on the messages, see *Error Messages and Codes*.

Parameters

pCallInfoBlock

in: The pointer to the description of the program and its parameter definition, see `ERX_CALL_INFORMATION_BLOCK`.

pParameterBlock

in out: The array of pointers to the actual parameter data.

eDirection

in: Type of parameters to be reset, input and/or output parameters. For `ERX_IN_PARM`, parameters with the `IN` attribute; for `ERX_OUT_PARM` parameters with the `OUT` attribute; for `ERX_INOUT_PARM`, all parameters are reset.

fFlags

in: `ERX_CF_STRUCTURED`, i.e. the parameters are collected in one data structure.

Return Codes

Value	Meaning
`00000000`	`ERX_S_SUCCESS`
`00010008`	`ERX_E_NOT_REGISTERED`

Related Functions

`ERXCall`

ERXServingCallback

Syntax

```
extern ERXeReturnCode ERXAPI ERXServingCallback(
       char                           *pConfigurationFile,
       void                           *pUserInfo,
       const ERXeControlFlags          fFlags
);
```

Description

This function implements the main function of the Callable RPC Server, see *Writing the Callback*. For information on the messages, see *Error Messages and Codes*.

Parameters

***pConfigurationFile**

in: Location consisting of path and file name of the configuration file in relative and absolute notation.

***pUserInfo**

in: User-specific data. The data is provided "as is" and can be used to provide a pointer to a memory location with user-specific data in callback functions.

fFlags

in: For future use.

Return Codes

Value	Meaning
00000000	ERX_S_SUCCESS
00010008	ERX_E_NOT_REGISTERED
00100050	ERX_ETB_SHUTDOWN_IMMED
02150148	ERX_ETB_BROKER_NOT_AVAILABLE

Related Functions

ERX_Callback_SERVER_CALL

ERXSetBrokerSecurity

Set the broker kernel security value.

Syntax

```
extern ERXeReturnCode ERXAPI ERXSetBrokerSecurity(
       char                       cKernelSecurity
);
```

Description

This function exposes the Broker ACI field `KERNELSECURITY` as a method to users of C Wrapper. The security settings are maintained internally by the RPC C runtime on a per-thread basis. See *Using EntireX Security* for more information.

For information on the messages, see *Error Messages and Codes*.

Parameters

cKernelSecurity

in

Return Codes

Value	Meaning
`00000000`	`ERX_S_SUCCESS`
`00010008`	`ERX_E_NOT_REGISTERED`

Related Functions

`ERXGetBrokerSecurity`

ERXSetCodepage

Set the codepage.

Syntax

```
extern ERXeReturnCode ERXAPI ERXSetCodepage(
       char                      szCodepage [ERX_MAX_CODEPAGE_LENGTH + 1]
);
```

Description

This function exposes the Broker ACI field `LOCALE-STRING` as a method to users of C Wrapper. The codepage is maintained internally by the RPC Runtime on a per-thread basis, see *Using Internationalization with the C Wrapper*.

For information on the messages, see *Error Messages and Codes*.

Parameters

szCodepage

in

Return Codes

Value	Meaning
00000000	ERX_S_SUCCESS
00010008	ERX_E_NOT_REGISTERED

Related Functions

ERXGetCodepage

ERXSetContext

Set a context (thread-safe RPC information).

Syntax

```
extern ERXeReturnCode ERXAPI ERXSetContext(
       ERX_CONTEXT_BLOCK     ERXPTR * pContextBlock
);
```

Description

This function supports RPC clients in multithreaded environments. It is used to set (thread-safe) RPC and broker context information, see *Programming Multithreaded RPC Clients*. For information on the messages, see *Error Messages and Codes*.

Parameters

pContextBlock

in: Pointer to context block, see ERX_CONTEXT_BLOCK

Return Codes

Value	Meaning
00000000	ERX_S_SUCCESS
00010008	ERX_E_NOT_REGISTERED

Related Functions

ERXGetContext

ERXSetIAFToken

Set the IAF token.

Syntax

```
extern ERXeReturnCode ERXAPI ERXSetIAFToken(
       char                  sIAFToken[ERX_IAF_TOKEN_LENGTH]
);
```

Description

With this function you can programmatically set the IAF Token.

Parameters

sIAFToken

in

Return Codes

Value	Meaning
00000000	ERX_S_SUCCESS
00010008	ERX_E_NOT_REGISTERED

Related Functions

ERXGetIAFToken

ERXSetSecurityToken

Get the current setting of the Security Token.

Syntax

```
extern ERXeReturnCode ERXAPI ERXSetSecurityToken(
       char                      szSecurityToken[ERX_MAX_securityToken_LENGTH + 1]);
```

Description

Sets the Broker Security Token. The security settings are maintained internally by the RPC C runtime on a per-thread basis. See *Using EntireX Security* for more information.

For information on the messages, see *Error Messages and Codes*.

Parameters

szSecurityToken

in: The security token to set.

Return Codes

Value	Meaning
00000000	ERX_S_SUCCESS
00010008	ERX_E_NOT_REGISTERED

Related Functions

ERXGetSecurityToken

ERXSetTraceLevel

Set the trace level of the RPC C runtime and the broker stub's trace.

Syntax

```
extern ERXeReturnCode ERXAPI ERXSetTraceLevel(
       long                  uTraceLevel
);
```

Description

With this function you can programmatically set the trace level of the RPC C runtime and the broker stub's trace. Use the provided defines in the erx.h header file for assigning trace levels:

```
#define ERX_TRACE_NONE     (0L)
#define ERX_TRACE_LEVEL1 (1L)
#define ERX_TRACE_LEVEL2 (2L)
#define ERX_TRACE_LEVEL3 (3L)
#define ERX_TRACE_LEVEL4 (4L)
```

Parameters

uTraceLevel

in

Return Codes

Value	Meaning
00000000	ERX_S_SUCCESS
00010008	ERX_E_NOT_REGISTERED

Related Functions

`ERXGetTraceLevel`

ERXTerminateServer

Terminate running Server.

Syntax

```
extern ERXeReturnCode ERXAPI ERXTerminateServer(
       ERX_CLIENT_IDENTIFICATION  ERXPTR *pClient,
       ERX_SERVER_ADDRESS         ERXPTR *pAddress,
       ERX_TERMINATE_SERVER       ERXPTR *pTerminateServer
);
```

Description

Shut down the running server. Before issuing `ERXTerminateServer` you must provide the following:

- the client identification
- the server address
- the shutdown command
- a pointer to a message area
- the length of the message area

See description of the `ERX_SERVER_ADDRESS` and control block.

For information on the messages, see *Error Messages and Codes*.

Parameters

pClient

in out: Pointer to the client identification, see `ERX_CLIENT_IDENTIFICATION`.

pAddress

in: Pointer to the server address, see `ERX_SERVER_ADDRESS`.

pTerminateServer

in out: Pointer to terminate structure, see `ERX_TERMINATE_SERVER`.

Return Codes

Value	Meaning
00000000	ERX_S_SUCCESS
00010008	ERX_E_NOT_REGISTERED
00020002	ERX_ETB_USER_DOES_NOT_EXIST
0003nnnn	ERX_ETB_CONVERSATION_ENDED
00070007	ERX_ETB_SERVICE_NOT_AVAILABLE
00740074	ERX_ETB_WAIT_TIMEOUT
02150148	ERX_ETB_BROKER_NOT_AVAILABLE

Related Functions

`ERXCall`

ERXUnregister

RPC C runtime is not needed anymore, i.e. free local resources.

Syntax

```
extern ERXeReturnCode ERXAPI ERXUnregister(void);
```

Description

Unregister from RPC C runtime. When a thread no longer needs the RPC C runtime, it must unregister itself from the runtime. See *Using the RPC Runtime*. For information on the messages, see *Error Messages and Codes*.

Return Codes

Value	Meaning
00000000	ERX_S_SUCCESS
00010008	ERX_E_NOT_REGISTERED

Related Functions

`ERXRegister`

ERXWait

Wait for the completion of asynchronous non-conversational call.

Syntax

```
extern ERXeReturnCode ERXAPI ERXWait(
       ERXCallId                          CallId,
 const ERX_CALL_INFORMATION_BLOCK ERXPTR *pCallInfoBlock,
       void                       ERXPTR *(ERXPTR pParameterBlock)[]
);
```

Description

Wait for an incoming request. Only applicable to connection-oriented processing.

For information on the messages, see *Error Messages and Codes*.

Parameters

CallId

in: The `CallId` returned by a previous `ERXCall`.

pCallInfoBlock

in: Pointer to the description of the program and its parameter definition, see `ERX_CALL_IDENTIFICATION`.

pParameterBlock[]

in out: The array of pointers to the actual parameter data.

Return Codes

Value	Meaning
00000000	ERX_S_SUCCESS
00010008	ERX_E_NOT_REGISTERED
00020002	ERX_ETB_USER_DOES_NOT_EXIST
0003nnnn	ERX_ETB_CONVERSATION_ENDED
00070007	ERX_ETB_SERVICE_NOT_AVAILABLE
00740074	ERX_ETB_WAIT_TIMEOUT
02150148	ERX_ETB_BROKER_NOT_AVAILABLE

API Function Descriptions for Variable-length Data Types AV, BV, KV and UV

The API of the RPC C runtime for variable-length data is defined in the following header file:

```
#include <erxvdata.h>
```

erxVDataAllocBytes

Allocates a new `VData` instance and copies `uLen` bytes from `pSource` into it. Intended for use with the IDL data types AV, BV, KV and UV together with the C programming language `mem...` functions. See *Using Variable-length Data Types AV, BV, KV and UV*. Any allocated `VData` instance must be freed with `erxVDataFree` if no longer used.

Syntax

```
extern ERX_HVDATA erxVDataAllocBytes(void *pSource, size_t uLen);
```

Parameters

pSource

in: A pointer to `uLen` bytes to copy.

uLen

in: Number of bytes to copy from `pSource` location.

Return Values

Points to a copy of the `VData` instance. No `VData` instance is allocated and null is returned in the following case:

- Insufficient memory

An empty `VData` instance (which holds an empty data area) is allocated in the following cases:

- A null pointer is passed for `pSource`.
- A value of zero is passed for `uLen`.

Related Functions

`erxVDataFree`
`erxVDataGetByteAddress`
`erxVDataGetLength`
`erxVDataReAllocBytes`

---

erxVDataAllocString

Allocates a new `VData` instance, and copies the string from `pSource` into it. Intended for use with the IDL data types AV and KV together with the C programming language `str...` functions. See *Using Variable-length Data Types AV, BV, KV and UV*. Any allocated `VData` instance must be freed with `erxVDataFree` if no longer used.

Syntax

```
extern ERX_HVDATA erxVDataAllocString(char *pSource);
```

Parameters

pSource

in: A pointer to a string to copy

Return Values

Points to a copy of the `VData` instance. No `VData` instance is allocated and null is returned in the following case:

- Insufficient memory

An empty `VData` instance, which holds an empty (null-terminated) string, is allocated in the following case:

- A null pointer is passed for `pSource`

Related Functions

`erxVDataFree`
`erxVDataReAllocString`

---

erxVDataAllocWideString

Allocates a new `VData` instance, copies the wide character string from the passed `pSource` location into it. Intended for use with the IDL data type UV together with the C programming language `wcs...` functions. See *Using Variable-length Data Types AV, BV, KV and UV*. Any allocated `VData` instance must be freed with `erxVDataFree` if no longer used.

Syntax

```
extern ERX_HVDATA erxVDataAllocWideString(wchar_t *pSource);
```

Parameters

pSource

in: A pointer to a wide character string to copy.

Return Values

Points to a copy of the `VData` instance. No `VData` instance is allocated and null is returned in the following case:

- Insufficient memory exists

An empty `VData` instance, which holds an empty (null-terminated) string) is allocated in the following case:

- A null pointer is passed for `pSource`

Related Functions

```
erxVDataFree
erxVDataReAllocWideString
```

erxVDataCopy

Copies an existing source `VData` instance to an existing target `VData` instance. Can be used for all IDL data types AV, BV and KV and UV. See *Using Variable-length Data Types AV, BV, KV and UV*.

Syntax

```
extern ERX_HVDATA erxVDataCopy(ERX_HVDATA hVDataTo, ERX_HVDATA hVDataFrom);
```

Parameters

hVDataTo

Handle of existing `VData` target instance.

hVDataFrom

Handle of existing `VData` source instance.

Return Values

Points to the target `VData` instance. The `VData` instance is not copied and null is returned in the following cases:

- An invalid handle is passed for `hVDataTo`
- An invalid handle is passed for `hVDataFrom`

Related Functions

```
erxVDataAllocBytes
erxVDataAllocString
erxVDataAllocWideString
erxVDataReAllocBytes
erxVDataReAllocString
erxVDataReAllocWideString
```

erxVDataFree

Frees all the memory used by a `VData` instance of IDL data types AV, BV, KV and UV. See *Using Variable-length Data Types AV, BV, KV and UV.*

Syntax

```
extern void erxVDataFree(ERX_HVDATA hVData);
```

Parameters

hVData

in: Handle of existing `VData` instance to free the resources.

Return Values

None.

Related Functions

`erxVDataAllocBytes`
`erxVDataAllocString`
`erxVDataAllocWideString`

---

erxVDataGetByteAddress

Get the address of binary data held by a `VData` instance of IDL data types AV, BV, KV and UV. Intended for use with the function `erxVDataGetLength` together with the C programming language `mem...` functions. See *Using Variable-length Data Types AV, BV, KV and UV.*

Syntax

```
extern void * erxVDataGetByteAddress(ERX_HVDATA hVData);
```

Parameters

hVData

in: Handle of `VData` instance from which to retrieve the address of the data

Return Values

Returns the address of the data held by the `VData` instance. A null pointer is returned in the following case:

- An invalid handle is passed

A pointer to an undefined area is returned in the following case:

- The `VData` instance is empty.

Related Functions

`erxVDataAllocBytes`
`erxVDataGetLength`
`erxVDataReAllocBytes`

erxVDataGetLength

Get the length in bytes of the data held by a `VData` instance of IDL data types AV, BV, KV and UV. Intended for use with the function `erxVDataGetByteAddress` together with the C programming language `mem...` functions. See *Using Variable-length Data Types AV, BV, KV and UV*.

Syntax

```
extern size_t erxVDataGetLength(ERX_HVDATA hVData);
```

Parameters

hVData

in: Handle of `VData` instance from which to retrieve the length.

Return Values

Number of bytes or length of string (excluding the terminating null or terminating wide-character null) held by the `VData` instance. Zero is returned in the following cases:

- An invalid handle is passed.
- The `VData` instance is empty.

Related Functions

`erxVDataAllocBytes`
`erxVDataGetByteAddress`
`erxVDataReAllocBytes`

erxVDataGetString

Get the address of the string held by a `VData` instance of IDL data types AV and KV. It will always have the address of a valid null-terminated string. The returned string can be used in conjunction with C language `str...` functions. See *Using Variable-length Data Types AV, BV, KV and UV*.

Syntax

```
extern char * erxVDataGetString(ERX_HVDATA hVData);
```

Parameters

hVData

in: Handle of `VData` instance from which to retrieve the string address.

Return Values

Returns a pointer to the address of the string held by the `VData` instance. A null pointer is returned in the following case:

- An invalid handle is passed.

A pointer to an empty string is returned in the following case:

- The `VData` instance is empty.

Related Functions

`erxVDataAllocString`
`erxVDataReAllocString`

---

erxVDataGetWideString

Get the address of the wide character string held by a `VData` instance of IDL data type UV. It will be guaranteed always to have the address of a valid null-terminated wide character string. The returned wide character string can be used in conjunction with C programming language `wcs...` functions. See *Using Variable-length Data Types AV, BV, KV and UV.*

Syntax

```
extern wchar_t * erxVDataGetWideString(ERX_HVDATA hVData);
```

Parameters

hVData

Handle to `VData` instance to get the wide character string address from.

Return Values

Returns a pointer to the address of the wide character string held by the `VData` instance. A null pointer is returned in the following case:

- An invalid handle is passed

A pointer to an empty wide character string is returned in the following case:

- The `VData` instance is empty

Related Functions

`erxVDataAllocWideString`
`erxVDataReAllocWideString`

erxVDataReAllocBytes

Assign new binary data to an existing `VData` instance. The function copies `uLen` bytes from `pSource` into the `VData` instance. Note that the address of the data held by the `VData` instance may have changed upon return. The location of the `VData` instance itself will always remain fixed. Intended for use with IDL data types AV, BV, KV and UV together with the C programming language `mem...` functions. See *Using Variable-length Data Types AV, BV, KV and UV*.

Syntax

```
extern void * erxVDataReAllocBytes(
              ERX_HVDATA hVData,
              void *pSource,
              size_t uLen);
```

Parameters

hVData

in: Handle of existing `VData` instance.

pSource

in: A pointer to `uLen` bytes to copy, or null to set the `VData` instance empty.

uLen

in: Number of bytes to copy from `pSource` location. Zero will set the `VData` instance empty.

Return Values

Returns a pointer to the data held by the `VData` instance. A null pointer is returned in the following case:

- an invalid handle is passed

A pointer to empty data is returned in the following cases:

- Insufficient memory exists to hold the new value
- A null pointer is passed for `pSource`
- Zero is passed for `uLen`.

Related Functions

`erxVDataAllocBytes`
`erxVDataGetByteAddress`

erxVDataReAllocString

Assign a new string to an existing `VData` instance of IDL data types AV and KV. The function copies the string from `pSource` into the `VData` instance. Note that the address of the data held by the `VData` instance may have changed upon return. The location of the `VData` instance itself will always remain fixed. Intended for use with IDL data type AV and KV together with the C programming language `str...` functions. See *Using Variable-length Data Types AV, BV, KV and UV*.

Syntax

```
extern char * erxVDataReAllocString(
              ERX_HVDATA hVData,
              char *pSource);
```

Parameters

hVData

in: Handle of existing `VData` instance.

pSource

in: A pointer to the string to copy, or null to set the `VData` instance to an empty string.

Return Values

Returns a pointer to the data held by the `VData` instance. A null pointer is returned in the following case:

- An invalid handle is passed

A pointer to a null string is returned in following cases:

- Insufficient memory exists to hold the new value
- A null pointer is passed for `pSource`

Related Functions

```
erxVDataAllocString
erxVDataGetString
```

erxVDataReAllocWideString

Copies the wide character string from the passed `pSource` location into the `VData` instance of IDL data type UV. Be aware that the address to the data held by the `VData` instance can be changed upon return. The location of the `VData` instance itself will always stay fixed. Intended for use with the IDL data type UV together with the C programming language `wcs...` functions. See *Using Variable-length Data Types AV, BV, KV and UV*.

Syntax

```
extern wchar_t * erxVDataReAllocWideString(ERX_HVDATA hVData, wchar_t *pSource);
```

Parameters

hVData

Handle to `VData` instance to put the wide character string into.

pSource

A pointer to the string to put, or null to set the `VData` instance to an empty wide character string.

Return Values

Returns a pointer to the data held by the `VData` instance. A null pointer is returned in the following case:

- An invalid handle is passed

A pointer to a null wide character string is returned in the following cases:

- Insufficient memory exists to hold the new value
- A null pointer is passed for `pSource`

Related Functions

`erxVDataAllocWideString`
`erxVDataGetWideString`

---

erxVDataReset

Sets an existing `VData` instance to empty (a null string for IDL data types AV and KV; a null wide-character string for IDL data type UV; and zero length for IDL data type BV). See *Using Variable-length Data Types AV, BV, KV and UV.*

Syntax

```
extern void erxVDataReset(ERX_HVDATA hVData);
```

Parameters

in hVData

Handle of existing `VData` instance.

Return Values

None.

Related Functions

```
erxVDataGetWideString
erxVDataGetString
erxVDataGetLength
```

API Function Descriptions for Unbounded Arrays

The API of the RPC C runtime for unbounded arrays is defined in the following header file:

```
#include <erxarray.h>
```

erxArrayAlloc

Allocates a new array instance of the given dimensions to be used as a so-called unbounded array. Array elements are initialized with their correct null value or zero corresponding to their IDL data type provided by the ERXeTypeCode. See *Using Unbounded Arrays*. Any allocated array instance must be freed with `erxArrayFree` if no longer used.

Syntax

```
extern ERX_HARRAY erxArrayAlloc(ERXeTypeCode      usType,
                                ERXeAttributes    usAttributes,
                                size_t            uLength,
                                unsigned int      uDimension,
                                ERX_ARRAY_INDEX   uArrayBound[]);
```

Description

usType	uLength	usAttributes	Note
ERX_TYPE_A	1 - 1GB	ERX_ATTR_STRING, ERX_ATTR_MF_ALPHA	1
ERX_TYPE_AV	0		2
ERX_TYPE_B	1 - 1GB		1
ERX_TYPE_BV	0		2
ERX_TYPE_D	0		2
ERX_TYPE_F	4,8		
ERX_TYPE_G	1 - 1GB		3
ERX_TYPE_I	1,2,4		
ERX_TYPE_K	1 - 1GB		1
ERX_TYPE_KV	0		2
ERX_TYPE_L	0		2
ERX_TYPE_N	1 - 29	ERX_ATTR_DOUBLE, ERX_ATTR_UNPACKED	4
ERX_TYPE_NU	1 - 29	ERX_ATTR_DOUBLE, ERX_ATTR_UNPACKED	4
ERX_TYPE_P	1 - 29	ERX_ATTR_DOUBLE, ERX_ATTR_PACKED	4
ERX_TYPE_PU	1 - 29	ERX_ATTR_DOUBLE, ERX_ATTR_PACKED	4
ERX_TYPE_S	1 - 1GB		3
ERX_TYPE_T	0		2
ERX_TYPE_U		ERX_ATTR_STRING, ERX_ATTR_MF_ALPHA	1
ERX_TYPE_UV	0		2

Notes:

1. When mapped to `ERX_ATTR_MF_ALPHA` the length is exactly the length given in the IDL file. When mapped to `ERX_ATTR_STRING` the length is the length + 1 given in the Software AG IDL file for the terminating null character or terminating wide-character null.
2. The length is implicitly defined by the IDL data type.
3. A Group or structure is normally associated with a `struct` typedef. The length to specify is the value of the `sizeof()` operator applied to the struct.
4. When mapped to `ERX_ATTR_UNPACKED` or `ERX_ATTR_PACKED` the length to specify relates to the IDL data type. The number of digits before and after the decimal point must be added. Example: For 5.2 specify 7. When mapped to `ERX_ATTR_DOUBLE` the length is implicit and thus obsolete.

Parameters

usType

The IDL data type stored in the array instance. See the description above.

usAttributes

The description above lists valid values for IDL data types. The values must exactly match the mapping options used when the RPC client is generated. See *Generate C Source Files from Software AG IDL Files*.

uLength

Depending on the data type (see table above) the length is required.

uDimension

Number of dimensions. The dimension must be at least 1. Up to 3 dimensions are allowed. The lower bound is always 0.

uArrayBound

Pointer to a vector containing the number of elements for each dimension. The number of vector elements must correspond to the number of array dimensions. The left (most significant) dimension is `uArrayBound[0]`.

Return Values

Points to the created copy of the array instance. No array instance is allocated and null is returned in the following cases:

- Insufficient memory exists
- The IDL data type provided by the `ERXeTypeCode` is invalid
- missing `uLength` depending on the data type
- `uDimension` is zero
- `uArrayBound` is invalid

Related Functions

`erxArrayFree`

---

erxArrayCopy

Copies an existing source array instance to an existing target array instance. Source and target array instance must exist, otherwise an error is returned. The contents of the target array instance are overwritten by the contents of the source instance. See *Using Unbounded Arrays*.

Syntax

```
extern ERXeReturnCode erxArrayCopy(ERX_HARRAY hArrayTo, ERX_HARRAY hArrayFrom);
```

Parameters

phArrayTo

Points to the target array instance created previously by `erxARrayAlloc`.

hArrayFrom

Points to the source array instance created previously by `erxArrayAlloc`.

Return Codes

Value	Meaning
`00000000`	`ERX_S_SUCCESS`
`00010005`	Out of Memory
`00010028`	Illegal Type
`00010079`	Invalid Unbounded Array

Related Functions

`erxArrayAlloc`

erxArrayFree

Frees all the memory used by the array instance. See *Using Unbounded Arrays*.

Syntax

```
extern ERXeReturnCode erxArrayFree(ERX_HARRAY hArray);
```

Parameters

hArray

Points to an array instance created by `erxArrayAlloc`.

Return Codes

Value	Meaning
`00000000`	`ERX_S_SUCCESS`
`00010079`	Invalid Unbounded Array

Related Functions

`erxArrayAlloc`

erxArrayGetAttributes

Returns all attributes defined for the array instance during allocation with `erxArrayAlloc`. See *Using Unbounded Arrays*.

Syntax

```
extern ERXeAttributes erxArrayGetAttributes(ERX_HARRAY hArray);
```

Parameters

hArray

Points to an array instance created by `erxArrayAlloc`.

Return Values

The attributes defined for the array instance.

Related Functions

`erxArrayAlloc`
`erxArrayGetElementLength`
`erxArrayGetTypeCode`

---

erxArrayGetBounds

Returns the array bound for a vector (the number of elements which can be stored in and retrieved from a given vector of an array instance). See *Using Unbounded Arrays*.

Syntax

```
extern ERX_ARRAY_INDEX erxArrayGetBounds(ERX_HARRAY      hArray,
                                         unsigned int    uDimension,
                                         ERX_ARRAY_INDEX uArrayIndex[]);
```

Parameters

hArray

Points to an array instance created by `erxArrayAlloc`.

uDimension

The dimension for which to set the array bound.

uArrayIndex

Pointer to a vector of indices defining an array position. The left-most (most significant) dimension is `uArrayIndex[0]`.

Return Values

The array bound for a given dimension. Zero is returned in the following cases:

- `hArray` is invalid
- `uDimension` is invalid or outside the current dimensions
- the unbounded array or specified vector has no elements
- `uArrayIndex` is invalid

Related Functions

```
erxArrayGetDimension
erxArrayRedimAll
erxArrayRedimVector
```

erxArrayGetDimension

Returns the number of dimensions of the array instance. See *Using Unbounded Arrays*.

Syntax

```
extern unsigned int erxArrayGetDimension(ERX_HARRAY hArray);
```

Parameters

hArray

Points to an array instance created by `erxArrayAlloc`.

Return Values

The number of dimensions of the unbounded array instance. Zero is returned in the following case:

- `hArray` is invalid

Related Functions

```
erxArrayGetBounds
erxArrayRedimAll
erxArrayRedimVector
```

erxArrayGetElement

Retrieves a single element of the array instance. The caller must provide a storage area of the correct size to receive the data. See *Using Unbounded Arrays*.

Syntax

```
extern ERXeReturnCode erxArrayGetElement(ERX_HARRAY        hArray,
                                         ERX_ARRAY_INDEX   uArrayIndex[],
                                         void            * pData);
```

Parameters

hArray

Points to an array instance created by `erxArrayAlloc`.

uArrayIndex

Pointer to a vector of indices defining an array position. The number of vector elements must correspond to the number of array dimensions. The left-most (most significant) dimension is `uArrayIndex[0]`.

pData

Pointer where to put the data stored in the given array position.

Return Codes

Value	Meaning
00000000	ERX_S_SUCCESS
00010079	Invalid Unbounded Array
00010084	Unbounded Array indices out of bounds
00010085	Invalid Data for Unbounded Array

Related Functions

`erxArrayReset`
`erxArraySetElement`

erxArrayGetElementLength

Retrieves the explicit logical length of the IDL data type of the array instance defined during allocation with `erxArrayAlloc`. See *Using Unbounded Arrays*.

Syntax

```
extern size_t erxArrayGetElementLength(ERX_HARRAY hArray);
```

Parameters

hArray

Points to an array instance created by `erxArrayAlloc`.

Return Values

The explicit logical length of the IDL data type of the array instance. Zero is returned in the following cases:

- the IDL data type has no explicit logical length, for example for the types L,D and T.
- `hArray` is invalid

Related Functions

```
erxArrayAlloc
erxArrayGetAttributes
erxArrayGetTypeCode
```

erxArrayGetTypeCode

Returns the IDL data type of the array instance defined during allocation with `erxArrayAlloc`. See *Using Unbounded Arrays*.

Syntax

```
extern ERXeTypeCode erxArrayGetTypeCode(ERX_HARRAY hArray);
```

Parameters

hArray

Points to an array instance created by `erxArrayAlloc`.

Return Values

The IDL data type of the array instance. `ERX_TYPE_UNKNOWN` is returned in the following case:

- `hArray` is invalid

Related Functions

```
erxArrayAlloc
erxArrayGetAttributes
erxArrayGetElementLength
```

erxArrayRedimAll

Changes all bounds of the array instance. The cardinality (number of dimensions) cannot be changed. For a 2 or 3-dimensional array, the result will be a square or a cube. See *Using Unbounded Arrays*.

Syntax

```
extern ERXeReturnCode erxArrayRedimAll(ERX_HARRAY          hArray,
                                       ERXeArrayPreserve   ePreserveData,
                                       ERX_ARRAY_INDEX     uNewArrayBound[]);
```

Parameters

hArray

Points to an array instance created by `erxArrayAlloc`.

ePreserveData

Determines whether the redimensioned array is to be initialized with null or whether the contents are to be kept (when elements exist in the old and new array). Valid values are: `ERX_PRESERVE_NO`, `ERX_PRESERVE_YES`

uNewArrayBound

Pointer to a vector containing the number of elements for each dimension. The number of vector elements must correspond to the number of array dimensions. The left-most (most significant) dimension is `uNewArrayBound[0]`.

Return Codes

Value	Meaning
`00000000`	`ERX_S_SUCCESS`
`00010005`	Out of Memory
`00010079`	Invalid Unbounded Array
`00010084`	Unbounded Array indices out of bounds
`00010086`	Invalid Preserve flag for Unbounded Array

Related Functions

`erxArrayGetBounds`
`erxArrayGetDimension`
`erxArrayRedimVector`

erxArrayRedimVector

Changes the specified bounds of the given vector of the array instance. For 2 and 3-dimensional arrays, the result can be a deformed array (that is, not a square or cube). See *Using Unbounded Arrays*.

Syntax

```
extern ERXeReturnCode erxArrayRedimVector(ERX_HARRAY          hArray,
                                          ERXeArrayPreserve   ePreserveData,
                                          unsigned int        uDimension,
                                          ERX_ARRAY_INDEX     uArrayIndex[],
                                          ERX_ARRAY_INDEX     uNewBound);
```

Parameters

hArray

Points to an array instance created by `erxArrayAlloc`.

ePreserveData

Determines whether the redimensioned vector (when it is the last dimension) is to be initialized with null or whether the contents are to be kept (when elements exist in the old and new array). New elements are always initialized with null. Valid values are:ERX_PRESERVE_NO, ERX_PRESERVE_YES

uDimension

The dimension for which to set new vector bound.

uArrayIndex

Pointer to a vector of indices defining an array position. The left-most (most significant) dimension is `uArrayIndex[0]`.

uNewBound

The number of elements to redimension the vector with.

Return Codes

Value	Meaning
`00000000`	`ERX_S_SUCCESS`
`00010005`	Out of Memory
`00010079`	Invalid Unbounded Array
`00010084`	Unbounded Array indices out of bounds
`00010086`	Invalid Preserve flag for Unbounded Array
`00010087`	Invalid Dimension

Related Functions

```
erxArrayGetBounds
erxArrayGetDimension
erxArrayRedimAll
```

erxArrayReset

Sets all elements of the array instance to null value or zero corresponding to the IDL data type given when the array was created. See *Using Unbounded Arrays*.

Syntax

```
extern ERXeReturnCode erxArrayReset(ERX_HARRAY hArray);
```

Parameters

hArray

Points to an array instance created by `erxArrayAlloc`.

Return Codes

Value	Meaning
`00000000`	`ERX_S_SUCCESS`
`00010079`	Invalid Unbounded Array

Related Functions

`erxArrayGetElement`
`erxArraySetElement`

---

erxArraySetElement

Stores the data element at a given location in the array instance. See *Using Unbounded Arrays*.

Syntax

```
extern ERXeReturnCode erxArraySetElement(ERX_HARRAY        hArray,
                                         ERX_ARRAY_INDEX   uArrayIndex[],
                                         void            * pData);
```

Parameters

hArray

Points to an array instance created by `erxArrayAlloc`.

uArrayIndex

Pointer to a vector of indices defining an array position. The number of vector elements must correspond to the number of array dimensions. The left-most (most significant) dimension is `uArrayIndex[0]`.

pData

Pointer to the data set into the given array position.

Return Codes

Value	Meaning
00000000	ERX_S_SUCCESS
00010079	Invalid Unbounded Array
00010084	Unbounded Array indices out of bounds
00010085	Invalid Data for Unbounded Array

Related Functions

erxArrayGetElement
erxArrayReset

API Function Descriptions for Reliable RPC

ERXGetReliableState

Get the current reliable RPC state.

Syntax

```
extern ERXeReturnCode ERXAPI ERXGetReliableState(
       unsigned long              *pulReliableState
);
```

Description

Get the current reliable RPC state. For a list of possible states with description, see ERXSetReliableState.

Parameters

pulReliableState

out: The current reliable RPC state

Return Codes

Value	Meaning
00000000	ERX_S_SUCCESS
00010009	ERX_E_PARAMETER_ERROR
00010008	ERX_E_NOT_REGISTERED

Related Functions

`ERXSetReliableState`

ERXSetReliableState

Set the reliable RPC state.

Syntax

```
extern ERXeReturnCode ERXAPI ERXSetReliableState(
       unsigned long              ulReliableState
);
```

Description

Set the current reliable RPC state to enable/disable reliable RPC.

State	Description
`ERX_RELIABLE_OFF`	The `ERX_RELIABLE_OFF` state represents the "normal" RPC.
`ERX_RELIABLE_AUTO_COMMIT`	The `ERX_RELIABLE_AUTO_COMMIT` puts each RPC request in a single reliable RPC message and commits each message automatically. To query the status of the sent reliable RPC message, you first have to resolve the reliable ID with `ERXGetReliableID`. With the retrieved reliable ID you can query the status of the reliable RPC message with `ERXGetReliableStatus` at any time. See also *Writing a Client using `AUTO COMMIT`*.
`ERX_RELIABLE_CLIENT_COMMIT`	On `ERX_RELIABLE_CLIENT_COMMIT` the client application can send a sequence of reliable RPC messages and can commit them whenever it is required. For this purpose `ERXReliableCommit` is offered. The client application also has the option to roll back the sequence of reliable RPC messages by using `ERXReliableRollback`. See also *Writing a Client*.

Parameters

ulReliableState

in: The reliable RPC state to set the values to

Return Codes

Value	Meaning
00000000	ERX_S_SUCCESS
00010009	ERX_E_PARAMETER_ERROR
00010008	ERX_E_NOT_REGISTERED

Related Functions

ERXGetReliableState

ERXReliableCommit

Commits a sequence of reliable RPC messages.

Syntax

```
extern ERXeReturnCode ERXAPI ERXReliableCommit(
       ERX_SERVER_ADDRESS         ERXPTR *pAddress
);
```

Description

Commits a sequence of reliable RPC messages in mode `ERX_RELIABLE_CLIENT_COMMIT`. See `ERXSetReliableState`.

Parameters

pAddress

in: The server address to send the commit to.

Return Codes

Value	Meaning
00000000	ERX_S_SUCCESS
00010009	ERX_E_PARAMETER_ERROR
00010010	ERX_E_CONTROL_BLOCK_NOT_FOUND
00010008	ERX_E_NOT_REGISTERED

Related Functions

ERXReliableRollback

ERXReliableRollback

Rolls back a sequence of reliable RPC messages.

Syntax

```
extern ERXeReturnCode ERXAPI ERXReliableRollback(
       ERX_SERVER_ADDRESS         ERXPTR *pAddress
);
```

Description

Rolls back a sequence of reliable RPC messages in mode `ERX_RELIABLE_CLIENT_COMMIT`. See `ERXSetReliableState`.

Parameters

pAddress

in: The server address to which to send the rollback.

Return Codes

Value	Meaning
00000000	ERX_S_SUCCESS
00010009	ERX_E_PARAMETER_ERROR
00010010	ERX_E_CONTROL_BLOCK_NOT_FOUND
00010008	ERX_E_NOT_REGISTERED

Related Functions

`ERXReliableCommit`

ERXGetReliableID

Get the reliable ID of the current reliable RPC message or message sequence.

Syntax

```
extern ERXeReturnCode ERXAPI ERXGetReliableID(
        ERX_SERVER_ADDRESS         ERXPTR *pAddress,
        ETB_CHAR                   ERXPTR *pReliableID
);
```

Description

Get the current reliable ID. The reliable ID is required to get the status of reliable RPC messages, see `ERXGetReliableStatus`.

In the case of `ERX_RELIABLE_CLIENT_COMMIT`, this method must be called before `ERXReliableCommit` or `ERXReliableRollback` is invoked, otherwise you might get the error 00010010.

In the case of `ERX_RELIABLE_AUTO_COMMIT`, this method must be called directly after the RPC message is sent and before any other RPC runtime calls, otherwise the reliable ID is lost and you cannot retrieve the message status.

Parameters

pAddress

in: The server address which was used for the interface object call.

pReliableID

out: The reliable ID of the current reliable RPC message.

Important:
The pointer `pReliableID` must point to a field defined like `ETB_CHAR szReliableID[16+1]`, otherwise unpredictable results occur.

Return Codes

Value	Meaning
00000000	ERX_S_SUCCESS
00010003	ERX_E_UNKNOWN_MEDIUM
00010009	ERX_E_PARAMETER_ERROR
00010010	ERX_E_CONTROL_BLOCK_NOT_FOUND
00010008	ERX_E_NOT_REGISTERED

Related Functions

`ERXGetReliableStatus`

ERXGetReliableStatus

Get the status of the reliable RPC messages.

Syntax

```
extern ERXeReturnCode ERXAPI ERXGetReliableStatus(
        ERX_CLIENT_IDENTIFICATION  ERXPTR *pClient,
        ERX_SERVER_ADDRESS         ERXPTR *pAddress,
        ETB_CHAR                   ERXPTR *pReliableID,
        ETB_BYTE                          *pReliableStatus
);
```

Description

Get the status of the reliable RPC messages given by the reliable ID. by given Reliable ID.

Status can be one of the values listed under *ACI Fields used for Units of Work*.

Parameters

pClient

in: Client information.

pAddress

in: Server information.

pReliableID

Important:
in: The reliable ID of the reliable RPC messages. The pointer `pReliableID` must point to a field defined like `ETB_CHAR szReliableID[16+1]`, otherwise unpredictable results occur.

pReliableStatus

Important:
out: the status of the requested reliable RPC message (identified by the given reliable ID). pReliableStatus points to a field of one byte.

Status can be one of the values listed under *ACI Fields used for Units of Work*.

Return Codes

Value	Meaning
`00000000`	`ERX_S_SUCCESS`
`00010003`	`ERX_E_UNKNOWN_MEDIUM`
`00010009`	`ERX_E_PARAMETER_ERROR`
`00010008`	`ERX_E_NOT_REGISTERED`

Related Functions

`ERXGetReliableID`

ERXControl

Control of RPC C runtime.

Syntax

```
extern ERXeReturnCode ERXAPI ERXControl
(
    const ERXCallId          CallId,
          ERXeControlCommand eCmd
);
```

Description

For future use.

Parameters

CallId

in: `ERXCallId`.

eCmd

in: command to use.

Return Codes

Value	Meaning
`00000000`	`ERX_S_SUCCESS`
`00010009`	`ERX_E_PARAMETER_ERROR`
`00010010`	`ERX_E_CONTROL_BLOCK_NOT_FOUND`
`00010008`	`ERX_E_NOT_REGISTERED`

Related Functions

None.