Writing a Single-threaded C RPC Client Application

This chapter describes in six steps how to write your first C client program.

- Step 1: Base Declarations Required by the C Wrapper
- Step 2: Required Settings for the C Wrapper
- Step 3: Register with the RPC Runtime
- Step 4: Issue the RPC Request
- Step 5: Examine the Error Code
- Step 6: Deregister with the RPC Runtime

The example given here demonstrates how to write a single-threaded C RPC client application. It demonstrates an implicit broker logon (because no broker logon/logoff calls are implemented), where it is required to switch on the `AUTOLOGON` feature in the broker attribute file.

The following steps describe how to write a single-threaded C client program. We recommend reading them first before writing your first RPC client program and following them where appropriate.

Step 1: Base Declarations Required by the C Wrapper

Step 1a: Include the Generated Header File

Define the generated client header file. This header file includes the RPC runtime header file *erx.h* and defines structures and prototypes for your RPC requests.

```
/* include generated header file */
#include "example.h"
```

Step 1b: Define Global Variables to Communicate with the Client Interface Objects

For single-threaded clients you have to declare in your main program the following global variables, used for communication with the interface objects:

```
/* Needed global variables for the CLIENT interface object */
ERXCallId                 ERXCallId;
ERXeReturnCode            ERXrc;
ERX_ERROR_INFO            ERXErrorInfo;
ERX_Server_ADRESS         ERXServer;
ERX_CLIENT_IDENTIFICATION ERXClient;
```

Step 2: Required Settings for the C Wrapper

Step 2a: Identify the User with a Broker User ID

For implicit broker logon, if required in your environment, the client password can be given here. It is provided then through the interface object call.

```
/* set client identification */
memset( &ERXClient, 0, sizeof(ERXClient) );
strcpy( (char*) ERXClient.szUserId, "ERX-USER" );
strcpy( (char*) ERXClient.szPassword, "ERX_PASS");
```

Step 2b: Set the Broker and Service to be Called

Your application will wait a maximum of 55 seconds for a server response. If the server does not answer within this period, the broker gives your program control again with an error code 00740074.

```
ERXServer.Medium = ERX_TM_BROKER;
ERXServer.ulTimeOut = 55;

/* set Broker-Id, server-name, class-name and service-name */
strcpy( (char*) ERXServer.Address.BROKER.szEtbidName,   "ETB001" );
strcpy( (char*) ERXServer.Address.BROKER.szServerName,  "SRV1" );
strcpy( (char*) ERXServer.Address.BROKER.szClassName,   "RPC" );
strcpy( (char*) ERXServer.Address.BROKER.szServiceName, "CALLNAT" );
```

Step 3: Register with the RPC Runtime

As a general rule, before using the RPC runtime you have to register it. After registration, the RPC runtime holds information on a per-thread basis. See *Using the RPC Runtime* for more information.

```
/* register to the RPC runtime */
ERXrc = ERXRegister( ERX_V81 );
If ( ERX_FAILED( ERXrc ) )
{
/* code for  error handling */
}
```

Step 4: Issue the RPC Request

The RPC interface object `CALC` is called as C function (see *Calling Servers as Procedures or Functions*).

```
/* do the remote procedure call */
result = CALC( '+', 123456, 78910 );
```

Step 5: Examine the Error Code

When a return from the RPC request has been received, check whether the call was successful with the macro ERX_FAILED.

```
if( ERX_FAILED( ERXrc ) )
{
/* code for error handling */
}
```

Detailed information about an error can be retrieved with the function `ERXGetLastError`. For the error messages returned, see *Error Messages and Codes*.

Step 6: Deregister with the RPC Runtime

As a general rule, after using the RPC runtime you should unregister from it. This will free all resources held by the RPC runtime for the caller. See *Using the RPC Runtime* for more information.

```
/* unregister to the RPC runtime */
ERXUnregister();
```