

EBSOTECH IX™ SDK 2.0

Technical Frequently Asked Questions

Contents

Technical Frequently Asked Questions	1
General Issues.....	1
1. What are the major components of Ebsotech IX™ SDK?.....	1
2. What programming languages does Ebsotech IX™ SDK support?.....	1
3. Where can I find sample source code that uses Ebsotech IX™ SDK?.....	1
Data Type Issues	1
1. Which string type should I use, LPSTR, LPWSTR, or LPTSTR?.....	1
2. When should I use DXALINKAGE and DXAFEXPORT definitions in function headers?	2
3. Why should my function headers have a DXAAPI definition?.....	2
4. What is the DXARESULT return value?.....	2
5. What is the DXASRESULT return value?	2
I/O Operations.....	3
1. Why does my Windows application crash while reading/writing a file handle?.....	3
Internet Programming	3
1. How do I avoid SIGPIPE signal when my server loses connection to client?.....	3
2. Why is it easier to use high-level Receiver objects than low-level sockets?.....	3
3. What is Listener useful for?	4
4. What is Message Receiver useful for?.....	4
5. What is the difference between Listener and Message Receiver?	4
6. Do I need threads when using Listener?.....	4
7. How do I get a pointer to my server object?.....	4
8. How do I get a pointer to my own session data object?.....	5
Threads.....	5
1. Does Ebsotech IX™ SDK support POSIX threads on all platforms?	5
2. Does Ebsotech IX™ SDK support Windows-styled threads on all platforms?.....	5
Process.....	5
1. Why does my application hang in dxaProcessCreate on Linux?	5
Configuration Maps.....	5
1. What configuration file formats are supported?	5
2. What is a configuration map?	5
3. Can I write comments in configuration files?.....	6
4. Does the configuration file handling preserve comments in files?	6
5. Can I read a configuration file directly to a data structure?.....	6
6. Can I iterate through a configuration map?.....	6
Collection Objects	6
1. How do I allocate collection objects from shared memory?.....	6
Internationalization.....	6
1. Do I need to do any initialization when I use internationalization?.....	6
2. Is there an easier way to get string resources?	7
3. Can a string resource contain newlines, and other special characters?.....	7
4. Can a string resource contain quotation marks?.....	7
5. Where can I find source code examples of internationalization?	8
6. Where can I find more information about internationalization?	8
Build System.....	8
1. What is the Ebsotech IX™ SDK build system?.....	8
2. What are the differences between Dxamake and GNU Make?	8
3. What environment variables do I need to define for the build system?	8

4.	Can I use spaces in paths in the environment variables?	9
5.	How do I enable dynamic compilation of libraries?	9

General Issues

1. What are the major components of Ebsotech IX™ SDK?

The following components are included in Ebsotech IX™ SDK Release 2.0:

- static (not included in evaluation version) and dynamic versions of Ebsotech IX™ SDK libraries
- resource compiler (`dxac1ng`) for internationalization and localization
- build system based on Dxamake

2. What programming languages does Ebsotech IX™ SDK support?

The Ebsotech IX™ SDK libraries are for C language in order to provide a maximally efficient application programming interface for building high-performance applications.

However, you can use the C libraries easily from C++, Java™, Python, Perl, and many other languages.

3. Where can I find sample source code that uses Ebsotech IX™ SDK?

Ebsotech IX™ SDK Software Accessories contain applications developed with the IX SDK. They come with full source code and most have an accompanying Technical Description document.

Data Type Issues

1. Which string type should I use, LPSTR, LPWSTR, or LPTSTR?

Typically, you should use `LPTSTR` if you want that your application can be compiled to use wide-character strings in a UNICODE compilation, and narrow-character string otherwise.

Hence, you should normally use the "generic" form of all function calls, such as `dxaStrDup` instead of `dxaStrDupA` or `dxaStrDupW`. Similarly, you should use the `_dt` version of the function mappings, such as `_dtprintf` instead of `_dnprintf` or `_dwprintf`.

However, notice that you often cannot use `LPTSTR` or `TCHAR` for certain tasks, such as in communication protocol messages, if the protocol is specifically defined to use either `LPSTR` or `LPWSTR` (or `CHAR` or `WCHAR`). In such cases, you need to do an appropriate conditional conversion between your `LPTSTR` and the protocol's `LPSTR` or `LPWSTR`, and possibly also make a character set conversion.

2. When should I use `DXALINKAGE` and `DXAFEXPORT` definitions in function headers?

`DXAFEXPORT` definition is used in source code in function definitions for functions that are exported in a library.

`DXALINKAGE` definition is used in library headers in function prototype declarations for exported library functions. That is, an exported library function that has `DXAFEXPORT` defined in source code has `DXALINKAGE` defined in header file.

The definitions have currently no other meaning but denotation. They should always be used, however, for possible future use.

3. Why should my function headers have a `DXAAPI` definition?

The `DXAAPI` definition defines the calling convention for the functions.

The definition has currently meaning only in Microsoft® Windows®, where it is defined as `__stdcall`, that is, a standard C calling convention.

4. What is the `DXARESULT` return value?

`DXARESULT` is the return value of many Ebsotech IX™ SDK functions. The value 0 indicates success, and any other value indicates an error. In a few rare functions, an "error code" can indicate a normal situation, such as expiration of a timeout.

The error codes are listed in `dxaerr.h`.

The `DXARESULT` error codes are always positive; for negative error codes, the functions use the `DXASRESULT` return type.

Notice that the `DXARESULT` return type name should only be used for error codes in Ebsotech IX™ SDK error code value space, as defined in `dxaerr.h`. For your own error (return) codes, you should use your own value space, and store it in variables of different type. With this convention, you avoid confusion between different error code spaces.

5. What is the `DXASRESULT` return value?

Whereas `DXARESULT` contains a positive-valued error code, `DXASRESULT` is intended for returning size and length values together with error codes. If the return value is zero or positive, it indicates a size or length, but if it is negative, it indicates an error with negated error code.

The error codes are listed in `dxaerr.h`.

Notice that the `DXASRESULT` return type name should only be used for error codes in Ebsotech IX™ SDK error code value space, as defined in `dxaerr.h`. For your own error (return) codes, you should use your own value space, and store it in variables of

different type. With this convention, you avoid confusion between different error code spaces.

I/O Operations

1. Why does my Windows application crash while reading/writing a file handle?

You may be writing or reading to or from the file handle in a different module than where you opened the file. This problem is caused by the way that Windows reserves memory; different modules have their own heap.

You should always read and write to files in the module where you opened it. You can accomplish this most trivially by making your own functions to open, read, write, flush, and close your files.

This problem does not normally occur in Linux® or UNIX® platforms.

Internet Programming

1. How do I avoid SIGPIPE signal when my server loses connection to client?

UNIX generates a SIGPIPE when connection to peer is lost while sending or receiving data. This problem can occur when using the low-level socket interface of the DXAINET library of Ebsotech IX™ SDK. With high-level Receiver, Listener, and Message Receiver objects, this is handled internally.

You can ignore the signal with a UNIX-specific request:

```
#ifdef unix
signal (SIGPIPE, SIG_IGN);
#endif
```

2. Why is it easier to use high-level Receiver objects than low-level sockets?

If you want to have your server to receive data from multiple open client connections, and at the same time listen to the server socket for new connections, you need to use `dxaInetGetSocketsByStatuses()` or raw BSD® socket functions `select()` or `poll()` to wait for status changes in the sockets. This procedure is not entirely simple, especially when you need to add new connections and remove old connection sockets. The client connection can also break unexpectedly, raising a signal, which has to be handled properly.

The Receiver object (inherited by Listener and Message Receiver) presents a simple and intuitive event-based solution to this problem. It is extensible, enabling you to build upper-level protocol events.

Creating a Listener is also easier than creating a server socket for listening.

3. What is Listener useful for?

You can use the Listener for listening and accepting new connections to the server, and after that for receiving messages from connected clients.

These both use cases generate "events", such as "new connection", "connection dropped", or "message received", which are passed to the user-defined callback function, where you can implement the actual application logic of your server.

4. What is Message Receiver useful for?

It has two main uses:

- After accepting a new connection in a Listener, you can take the connection object from the Listener and give it to a Message Receiver that handles further conversation with it. You need to run each Message Receiver in its own thread
- You can use it in a client to receive server-initiated requests. This is useful for cases such as shutdown messages. Also in this case, you probably need to run the Message Receiver in its own thread.

5. What is the difference between Listener and Message Receiver?

Listener has a socket to listen to for accepting new connections, while Message Receiver does not. There is no other real difference.

Both Listener and Message Receiver are "inherited" (conceptually) from the abstract Receiver object, which cannot be instantiated directly.

6. Do I need threads when using Listener?

No, threads are not necessary, as you can process the client requests in the same thread as where the Listener is running. Obviously, though, if processing a request takes very long, other clients will have to wait.

If you want better response times, you can use worker or transaction processor threads to process the client requests.

Alternatively, you can have a pool of message receiver threads, each handling one or more conversations with clients, while the main (Listener) thread only needs to accept new client connections.

7. How do I get a pointer to my server object?

You can get a user-defined void pointer in the `lpParam` value for the callback function by setting it in the initialization struct for the receiver object.

8. How do I get a pointer to my own session data object?

You can get a user-defined void pointer in the `lpParam` value for the callback function by setting it in the session object that you get for a new connection.

Threads

1. Does Ebsotech IX™ SDK support POSIX threads on all platforms?

Yes, POSIX® threads with mutex locks and variables are cross-implemented for all the supported platforms, including Microsoft Windows.

2. Does Ebsotech IX™ SDK support Windows-styled threads on all platforms?

Yes, Windows-styled threads with thread events are cross-implemented for all the supported platforms, including Linux and UNIX.

Process

1. Why does my application hang in `dxProcessCreate` on Linux?

You need to link your application with POSIX threads. Add `-lpthread` when linking your application. If you use the Ebsotech IX™ SDK build system you can set `LINK_DXASYS=1` which will do this for you.

Configuration Maps

1. What configuration file formats are supported?

The `DXACFG` library of Ebsotech IX™ SDK currently supports two kinds of configuration files: INI files (Microsoft Windows style) and GRC files (UNIX style).

In addition, you can customize the INI and GRC file formats with regard to various markup strings for comments, sections, tags, etc.

2. What is a configuration map?

Configuration map is an object stored in memory, while a configuration file is stored on a disk. When a configuration file is opened, it is read to memory as a configuration map.

After making changes to a configuration map, you can rewrite it as the configuration file.

Notice that the operations on the configuration map preserve the comments in configuration files as far as possible.

3. Can I write comments in configuration files?

Yes. DXACFG supports two kinds of comments: one-line, tagged with characters such as '#' or '//' in the beginning of the line, or multiple-line comment blocks, such as those in C, tagged with '/*' . . . '*/'.

The actual comment markup notation depends on the configuration file format. You can also customize the markup tags.

4. Does the configuration file handling preserve comments in files?

Yes, the comments in configuration files are preserved as far as possible.

5. Can I read a configuration file directly to a data structure?

Yes, by using the `dxaCfgParamValuesAssign()` function. It allows you to specify the items such as integers, strings, and time values as memory offsets in the data structure and loads the sections and tags from a configuration file directly to the data structure.

6. Can I iterate through a configuration map?

Yes, by using the `dxaConfMapGetFirstSection()` and `dxaConfMapGetSection()` functions to iterate through sections in a configuration map and `dxaConfMapGetFirstTag()` and `dxaConfMapGetTag()` functions to iterate through tags within a section.

The `dxaConfMapGetSection()` and `dxaConfMapGetTag()` functions retrieve object data, such as the name and value of a section or tag, respectively. They also retrieve a reference to the next section or tag, respectively, allowing iteration.

See the source code of CFGMGRSA and EGSA Software Accessories for examples on using these functions.

Collection Objects

1. How do I allocate collection objects from shared memory?

You need to allocate the needed memory yourself from shared memory, and then give the memory block as the optional argument to the creation function of the collection object.

Internationalization

1. Do I need to do any initialization when I use internationalization?

Yes, you need to:

- set the current locale

- check that all the string resources required by the application exist for the current locale

You may also need to first add some directories to the string resources search path, in addition to those given in the `DXALOCALEPATH` environment variable.

2. Is there an easier way to get string resources?

Yes, by following a simple convention for defining a few macros to access the resources. You need to define the following (replace the "MY" prefix with your module prefix):

```
#define MYSTR_MYMSG TEXT("mymsg") /* Application message strings. */
#define MYSTR_MYERR TEXT("myerr") /* Error message strings. */

/* Returns a message string from 'mymsg' resource. */
#define MYMSG(id) dxaStrCollectionGetStringPtr(MYSTR_MYMSG, id)

/* Returns a message string from 'myerr' resource. */
#define MYERR(id) dxaStrCollectionGetStringPtr(MYSTR_MYERR, id)
```

With these macro definitions, you can simply write:

```
_dtprintf (MYMSG (MYMSG_SOME_MESSAGE));
_dtprintf (MYMSG (MYMSG_FMT2_SOME_ERROR), /* Print formatting string.
*/
iMyError, /* Print error code. */
MYERR (iMyError)); /* Print error string. */
```

However, if you want to have the macros to return the string for the default language in situations where the default language resource file is not available or not found in the search path, you need to use more advanced macros together with a special template file for generating the default strings as macros.

For more detailed examples, you should examine the source code of Language Resources Software Accessory (LNGSA) and other Software Accessories that use internationalization.

3. Can a string resource contain newlines, and other special characters?

There is currently no standard convention for marking newline, carriage return, tab, linefeed, or other special characters in localized string resources. In principle, you should use the notation `"\n"`, `"\r"`, or `"\t"`, but there is currently no function to translate such instances to actual newline characters.

This applies also to other special characters.

4. Can a string resource contain quotation marks?

Yes, unless you are using the string generation template and macros described in the previous question. In that case, you cannot use quotation marks directly, but you need to

mark them with for example octal code notation `\042` and then interpret it with a string parser.

5. Where can I find source code examples of internationalization?

You can find source code examples of internationalization from many of the Software Accessories distributed in the Software Accessory CD of Ebsotech IX™ SDK.

The following Software Accessories use an advanced internationalization scheme:

- Language Resource Software Accessory (LNGSA)
- Entity Guessing Software Accessory (EGSA)
- E-Shop Software Accessory (ESHOPSA)

6. Where can I find more information about internationalization?

The primary source for information about internationalization is Chapter 6 'Globalization Support' in *Ebsotech IX™ SDK Developer's Guide*.

Build System

1. What is the Ebsotech IX™ SDK build system?

The build system is a makefile framework that provides an easy way to build libraries and large applications transparently in different platforms.

Building with the makefiles is done with the Dxamake build engine, a slightly modified version of GNU Make. Dxamake is Free Software; its source code is available on request.

The build system framework is implemented in three makefiles: `dxadef.mki`, `dxacmp.mki`, and `dxalnk.mki`, located in `$DXASDKDIR/proj`.

2. What are the differences between Dxamake and GNU Make?

Currently there are only two differences:

- Dxamake accepts also space characters (in addition to tabs) in make rules
- Dxamake handles the `\` directory separator better in Windows platforms

3. What environment variables do I need to define for the build system?

You need to define the following environment variables:

`DXASDKDIR` - installation path of Ebsotech IX™ SDK

`DXAMAIN` - development root directory

`DXAOUT` - output root directory

Notice that if you are working on multiple projects and the development directory root is different for each project, you may not want to define `DXAMAIN` in the environment, but rather define it with relative path in the top-level makefile of your projects:

```
export DXAMAIN = ../../..
```

For example, if you are compiling the software accessories for Ebsotech IX™ SDK, you should not have `DXAMAIN` defined.

If you have `DXAMAIN` defined in your environment, you can undefine it in Linux and UNIX with:

```
export -n DXAMAIN
```

and in Windows:

```
set DXAMAIN=
```

4. Can I use spaces in paths in the environment variables?

No, the paths must not have spaces, especially in Microsoft Windows platforms. For example, the following DOES NOT WORK:

```
set DXASDKDIR=C:\Program Files\Ebsotech IX SDK
```

If your Windows paths have spaces, you need to use the short names of the directories. For example, the short name of the above path is usually (but not necessarily):

```
set DXASDKDIR=C:\PROGRA~1\EBSOTE~1
```

You can find out the short names with the following command:

```
dir /x
```

Alternatively, you can also quote the spaces with the backslash character:

```
set DXASDKDIR=C:\Program\ Files\Ebsotech\ IX\ SDK
```

However, this notation is not recommended, as it may bring unexpected problems, especially in the future.

5. How do I enable dynamic compilation of libraries?

Dynamic compilation is used by default.

If you want static compilation, you should use the definitions

```
DXACMP_STATIC = 1
DXALINK_STATICALLY = 1
```

to build your project. The former requests static compilation of libraries and the latter static linking of applications.

EBSOTECH IX™ SDK 2.0

Document Information

This Ebsotech FAQ provides answers to technical questions about Ebsotech IX™ SDK Release 2.0.

Information in this document is subject to change without notice and does not represent a commitment on the part of Ebsotech. Ebsotech is not liable for errors contained in this document or for incidental or consequential damages in connection with furnishing or use of this material.

Trademarks

Ebsotech, the Ebsotech logo, Ebsotech IX, Ebsotech AMS, Ebsotech DXA, Ebsotech GDP and Ebsotech SPE are trademarks or registered trademarks of Ebsotech.

BSD is a registered trademark of Berkeley Software Design, Inc. Linux is a registered trademark of Linus Torvalds. Microsoft and Windows are registered trademarks of Microsoft Corporation in the United States and/or other countries. Java is a trademark of Sun Microsystems, Inc. in the United States and other countries. POSIX is a registered trademark of the IEEE, Inc. UNIX is a registered trademark of The Open Group in the United States and other countries. All other trademarks and registered trademarks are the property of their respective owners.

© Copyright 2003 Ebsotech

About Ebsotech

Founded in 2003, Ebsotech was formed to create a new level of software tools previously unavailable to the IT-industry for multi-platform high-performance software development. As a part of this mission company has released Ebsotech IX™ product-line offering high productivity C/C++ development and high-performance state-of-the-art results. As supplementary, Ebsotech offers support services and technology consulting services in co-operation with various technology partners.

Ebsotech • www.ebsotech.com
Rajakyläntie 28 A, FIN-01280 Vantaa
Phone: +358 44 2885299
E-mail: sales@ebsotech.com