IBM Education Assistance for z/OS V2R1

Item: Language Environment
Element/Component: Standard I/O Extensions

Material is current as of June 2013

Agenda

- Trademarks
- Presentation Objectives
- Overview
- Usage & Invocation
- Presentation Summary
- Appendix

Trademarks

- See url http://www.ibm.com/legal/copytrade.shtml for a list of trademarks.

Presentation Objectives

- Describe the addition of a set of non-standard APIs that extend the current XL C/C++ runtime library I/O support.

Overview

- Problem Statement / Need Addressed
  - Many other UNIX platforms provide APIs that allow access to select internals of the FILE structure. No functions currently exist in the XL C/C++ runtime library to support this type of behavior.
- Solution
  - Provide a set of non-standard FILE I/O extension functions originally introduced by Solaris and also implemented for glibc.
- Benefit / Value
  - The ability to port applications from other UNIX platforms to z/OS.
  - Controlled access to portions of the FILE structure that were previously inaccessible is allowed using new set of APIs.

Usage & Invocation

- Applications can query an open FILE stream for information about the current status of the stream. Other types of modifiable requests can be performed against the open FILE stream as well.
- 12 new APIs are defined in a new header called <stdio_ext.h>
- Non thread-safe versions of 6 of the new APIs are provided
- The header also defines new macros for use with the APIs
- The majority of the new functions return 0 when in error and set errno accordingly.
- Boolean type querying functions return 1 when true and 0 otherwise.

Usage & Invocation

```
#include <stdio.h>
#include <stdio_ext.h>

size_t __fbufsize(FILE *stream);
```

- Takes an open FILE stream pointer, *stream*. Returns the size of the buffer in bytes.

Usage & Invocation

```
#include <stdio.h>
#include <stdio_ext.h>

int __flbf(FILE *stream);
```

- Takes an open FILE stream pointer, *stream*. Determines if the specified stream is line buffered.

Usage & Invocation

```
#include <stdio.h>
#include <stdio_ext.h>

void _flushlbf(void);

#define _OPEN_SYS_UNLOCKED 1
#include <stdio.h>
#include <stdio_ext.h>

void _flushlbf_unlocked(void);
```

- Flushes all open line-buffered streams. Behaves similar to the fflush() function.
- _flushlbf_unlocked() is functionally equivalent to _flushlbf() with the exception that it is not thread-safe.

Usage & Invocation

```
#include <stdio.h>
#include <stdio_ext.h>

size_t __fpending(FILE *stream);

#define _OPEN_SYS_UNLOCKED 1
#include <stdio.h>
#include <stdio_ext.h>

size_t __fpending_unlocked(FILE *stream);
```

- Takes an open FILE stream pointer, *stream*. Returns the number of bytes pending to be written in the output buffer. When the stream is opened for text processing, the function returns the number of characters pending (can be wide characters).
- __fpending_unlocked() is functionally equivalent to __fpending() with the exception that it is not thread-safe.

Usage & Invocation

```
#include <stdio.h>
#include <stdio_ext.h>

void __fpurge(FILE *stream);

#define _OPEN_SYS_UNLOCKED 1
#include <stdio.h>
#include <stdio_ext.h>

void __fpurge_unlocked(FILE *stream);
```

- Takes an open FILE stream pointer, *stream*. The function requests that any pending data in the stream be discarded.
- Note: The function is supported for UNIX files only.
- __fpurge_unlocked() is functionally equivalent to __fpurge() with the exception that it is not thread-safe.

Usage & Invocation

```
#include <stdio.h>
#include <stdio_ext.h>

int __freadable(FILE *stream);
```

- Takes an open FILE stream pointer, *stream*. Determines if the specified stream has been opened for reading.

Usage & Invocation

```
#include <stdio.h>
#include <stdio_ext.h>

size_t __freadahead(FILE *stream);

#define _OPEN_SYS_UNLOCKED 1
#include <stdio.h>
#include <stdio_ext.h>

size_t __freadahead_unlocked(FILE *stream);
```

- Takes an open FILE stream pointer, *stream*. Returns the number of bytes remaining to read in the input buffer. When the stream is opened for text processing, the function returns the number of characters remaining to be read (can be wide characters).
- __freadahead_unlocked() is functionally equivalent to __freadahead() with the exception that it is not thread-safe.

Usage & Invocation

```
#include <stdio.h>
#include <stdio_ext.h>

int __freading(FILE *stream);

#define _OPEN_SYS_UNLOCKED 1
#include <stdio.h>
#include <stdio_ext.h>

int __freading_unlocked(FILE *stream);
```

- Takes an open FILE stream pointer, *stream*. Determines if the last operation on the specified stream was a read operation or if the specified stream is open for read-only.
- __freading_unlocked() is functionally equivalent to __freading() with the exception that it is not thread-safe.

Usage & Invocation

```
#include <stdio.h>
#include <stdio_ext.h>

void __fseterr(FILE *stream);
```

- Takes an open FILE stream pointer, *stream*. The function sets the specified stream in error.

Usage & Invocation

```
#include <stdio.h>
#include <stdio_ext.h>

int __fsetlocking(FILE *stream, int type);
```

- Takes an open FILE stream pointer, *stream* and a locking request, *type*. The function allows the type of locking on an opened stream to be controlled or queried by the application.
  - FSETLOCKING_INTERNAL – subsequent stdio functions perform implicit locking. This is the default behavior.
  - FSETLOCKING_BYCALLER – subsequent stdio functions assume that the caller is responsible for maintaining the integrity of the stream in the face of access by multiple threads.
  - FSETLOCKING_QUERY – returns the current locking type of the stream without changing it.

Usage & Invocation

```
#include <stdio.h>
#include <stdio_ext.h>

int __fwritable(FILE *stream);
```

- Takes an open FILE stream pointer, *stream*. Determines if the specified stream has been opened for writing.

Usage & Invocation

```
#include <stdio.h>
#include <stdio_ext.h>

int __fwriting(FILE *stream);

#define _OPEN_SYS_UNLOCKED 1
#include <stdio.h>
#include <stdio_ext.h>

int __fwriting_unlocked(FILE *stream);
```

- Takes an open FILE stream pointer, *stream*. Determines if the last operation on the specified stream was a write operation or if the specified stream is open for write-only or append-only.
- __fwriting_unlocked() is functionally equivalent to __fwriting() with the exception that it is not thread-safe.

Presentation Summary

- A new set of non-standard I/O extension functions is provided by the XL C/C++ runtime library.

Appendix

- XL C/C++ Runtime Library Reference (SA22-7821)
- XL C/C++ Programming Guide (SC09-4765)