IBM Toolkit for Swift on z/OS Community Edition

Documentation

Version 4.0

SC27-9267-00

IBM Toolkit for Swift on z/OS Community Edition

Documentation

Version 4.0

SC27-9267-00

Note
Before using this information and the product it supports, read the information in "Notices" on page 35.

First edition (May 2018)

This edition applies to IBM Toolkit for Swift on z/OS (Program 5655-SFT) and to all subsequent releases and modifications until otherwise indicated in new editions. Make sure you are using the correct edition for the level of the product.

© Copyright IBM Corporation 2018.
US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

© Copyright IBM Corp. 2018

Chapter 1. Overview of IBM Toolkit for Swift on z/OS

IBM® Toolkit for Swift on z/OS® is an implementation of Swift 4.0.1 that has been extended to support the z/OS operating system.

Swift is a modern programming language that focuses on 4 key design principles: safety, performance, modern software design, and language conciseness.

Swift is designed to make writing programs easier for developers by ensuring that code is written in a safe manner. Swift prevents undefined behavior typically exhibited at run time by catching such issues early at compile time. Type-safety plays a major role in Swift.

Swift is a statically compiled language, akin to COBOL, C, and C++. It differs from these languages in that memory is managed by the Swift run time. Its modern features are what separate Swift from other statically compiled languages.

Swift has a large community of libraries and tools, which are a critical part of the Swift ecosystem. The Swift Package Manager is a build management system that can be used to leverage existing libraries from the Swift community and enhance your applications.

IBM Toolkit for Swift on z/OS presents a safe and modern compiler that can be used to extend or replace existing z/OS assets.

© Copyright IBM Corp. 2018

Chapter 2. Summary of changes

This section lists the updates to IBM Toolkit for Swift, V4.0 since it was first released in May 2018.

July 2018 Update

- Runtime performance is improved as the benchmarks are comparable or better than x86.
- To enhance EBCDIC codepage, EBCDIC 037 support is added.
- To improve Swift Package Manager, informative diagnostic messages are supported.
- SwiftyJSON is added to shipped packages as JSON Parser and JSON Generator.
- The example of IBM Kitura RESTful web service with backend calling z/OS Connect is added.
- SSL CA is now packaged with Swift on z/OS, enabling remote https swift packages such as those on https://github.com.
- Db2 package functional improvements are made.
- The -g option support for DWARF Debug generation is added.
- To improve the Foundation library functionality, RunLoop timer is improved.

© Copyright IBM Corp. 2018

Chapter 3. Understanding IBM Toolkit for Swift on z/OS

IBM Toolkit for Swift on z/OS ships with 6 core components.

Swift compiler
The Swift compiler (`swiftc`) is responsible for compiling Swift source code into executable machine code. The Swift compiler generates 64-bit executables using the XPLINK calling convention model. It also relies on the GOFF object format.

Core libraries
The Core libraries include the following 4 components:

Libc A set of C Language Environment®® (LE) routines exposed to Swift such as cos or malloc.

Foundation
A set of classes to complement the Swift standard library, including classes for Collections such as Sets or HashTables, File I/O, Units/Measurements, and more. Further details are available in the Foundation reference for Swift.

XCTest
A set of classes to create and run unit tests and performance tests.

libdispatch
A set of classes to execute code concurrently on multicore hardware by submitting work to dispatch queues managed by the system.

Swift standard library
The Swift standard library encompasses a number of basic data types such as Int or Double, collections such as Array or Dictionary, protocols and functions. It also consists of the Swift runtime environment which is responsible for the dynamic features of the compiler and memory management.

Swift Package Manager
Swift Package Manager is a tool for managing the distribution of Swift code. It is integrated with the Swift build system to automate the process of downloading, compiling, and linking dependencies.

zOSSwift
zOSSwift is an extra Swift library providing additional APIs and classes for the z/OS operating system.

Examples
The examples shipped with IBM Toolkit for Swift on z/OS are as follows:

- REST API
- JSON Parsing
- IBM Kitura Web Server
- Swift and Db2® integration
- Swift to PL/I interlanguage calls
- Swift to ASM (31-bit/24-bit) calls

© Copyright IBM Corp. 2018

Chapter 4. Hardware and software requirements

IBM Toolkit for Swift on z/OS runs on the following IBM Z servers:

- z14
- z13™
- z13s
- zEnterprise® EC12
- zEnterprise BC12

IBM Toolkit for Swift on z/OS requires z/OS V2R1 or later. It is recommended that APAR OA53548 is applied as this APAR provides a major link time performance boost.

© Copyright IBM Corp. 2018

Chapter 5. Package listing

The installed directory contains the extracted contents of the tar file you obtain. The installed directory is shown as follows:

```
   |-swift-v*                  (Extracted installation of Swift)
   |---setup.sh                (Setup etc/profile script containing environment variables)
   |---README                  (Instructions on how to install and use Swift)
   |---LICENSE/                (Contains License text files)
   |---bin                     (Location of all binaries, including swiftc and swift)
   |---etc                     (Example .profile to set up environment variables for Swift)
   |---include                 (Include headers, used by Swift)
   |---examples                (All Swift Examples, including Db2, Foundation, REST APIs)
   |---lib                     (All static and dynamic libraries, used by Swift)
    |-----swift/CoreFoundation  (Foundation C headers/module map)
    |-----swift/libc            (Libc C LE headers)
    |-----swift/bsd             (Libc C library extension headers)
    |-----swift/ebcdic_unicode  (Libc C ASCII extension headers)
    |-----swift/dispatch        (Libdispatch C headers/module map)
    |-----swift/os              (Libdispatch C headers)
    |-----swift/pm              (Swift Package swiftmodules)
    |-----swift/shims           (Swift C runtime headers/module map)
    |-----swift/zos/
    |-------HSZCORE             (Swift Core DLL)
    |-------HSZLIBC             (Swift Libc DLL)
    |-------HSZFNDTN            (Swift Foundation DLL)
    |-------HSZDSPTC            (Swift Dispatch DLL)
    |-------HSZXCTST            (Swift XCTest DLL)
    |-------HSZONONE            (Swift Onone DLL)
    |-------HSZBSD              (Libbsd DLL)
    |-------HSZCURL             (Curl DLL)
    |-------HSZCRYPT            (Cyrpto DLL)
    |-------HSZSSL              (SSL DLL)
    |-------HSZUUID             (UUID DLL)
    |-------HSZZLIB             (ZLIB DLL)
    |-------HSZXML2             (Libxml2 DLL)
    |-------HSZBLOCKS           (C Blocks DLL)
    |-------HSZI58*             (ICU DLLs)
    |-------libFoundation.a     (Swift Foundation archive (contains DLL sidedecks))
    |-------libXCTest.a         (Swift XCTest archive (contains DLL sidedecks))
    |-------libdispatch.a       (Swift Dispatch archive (contains DLL sidedecks))
    |-------libswiftCore.a      (Swift Swift Core archive (contains DLL sidedecks))
    |-------libswiftDeps.a      (Swift Dependencies archive (contains DLL sidedecks))
    |-------libswiftLibc.a      (Swift Libc archive (contains DLL sidedecks))
    |-------libswiftSwiftOnoneSupport.a (Swift SwiftOnone archive (contains DLL sidedecks))
    |-------libswiftZInit.a     (Swift ZInit static archive)
    |-----swift/zos/s390x/
    |-------Dispatch.swiftdoc   (Libdispatch interface in binary form)
    |-------Dispatch.swiftmodule(Libdispatch Swift Module (used in import Dispatch))
    |-------Foundation.swiftdoc (Foundation interface in binary form)
    |-------Foundation.swiftmodule (Foundation Swift Module (used in import Foundation))
    |-------Libc.swiftdoc       (Libc interface in binary form)
    |-------Libc.swiftmodule    (Libc Swift Module (used in import Libc))
    |-------Swift.swiftdoc      (Swift Standard Library interface in binary form)
    |-------Swift.swiftmodule   (Swift Swift Module (used in import Swift (implicit))
    |-------SwiftOnoneSupport.swiftdoc (SwiftOnoneSupport interface in binary form)
    |-------SwiftOnoneSupport.swiftmodule (SwiftOnoneSupport Swift Module /
(used in import SwiftOnoneSupport - implicit))
    |-------XCTest.swiftdoc     (XCTest interface in binary form)
    |-------XCTest.swiftmodule  (XCTest Swift Module (used in import XCTest))
    |-------libc.modulemap      (Libc module map (used in Import Libc))
   |---usr                     (Manpages, timezone files, used by Swift)
   |---extras                  (Extra zOS swift libraries, including zOSSwift)
```

© Copyright IBM Corp. 2018

```
    |-------libzOSSwift.dll     (zOS Swift DLL)
    |-------libzOSSwift.a       (zOS Swift Library archive (contains DLL sidedeck))
    |-------zOSSwift.swiftdoc   (z/OS Swift library swiftdoc binary documentation)
    |-------zOSSwift.swiftmodule (z/OS Swift library swiftmodule binary, /
describing the module's interface)
```

Chapter 6. Installing IBM Toolkit for Swift on z/OS

To download the latest version of IBM Toolkit for Swift on z/OS, go to IBM Toolkit for Swift on z/OS in Developer Centers. Follow these instructions to install IBM Toolkit for Swift on z/OS.

About this task

You must install the product under IBM UNIX System Services. All installed files are in the form of HFS files.

Procedure

1. Upon downloading the distributed `tar` file, unpack it as follows:

   ```
   tar -xvfo swift-v*.tar
   ```

   Then you get the installed directory that contains the extracted contents.

   Note: IBM Toolkit for Swift on z/OS requires approximately 2.5 GB of storage space. To estimate the total uncompressed size of the tar file, use the following command:

   ```
   /bin/tar -vtf swift-v*.tar 2>/dev/null | /bin/awk '{i+=$5+512}
   END{ printf "%d MB\n", (i+999999)/1000000 }'
   ```

2. After extracting the contents of the `tar` file, use the convenience script `setup.sh` to set up environment variables, such as **LIBPATH**:

   ```
   ./setup.sh
   ```

What to do next

After installation, source the profile file generated during the installation with the dot command as follows:

```
. {INSTALL_DIR}/etc/profile
```

This will adjust your environment so that you can compile and run Swift programs.

To compile and link a sample program, use the following command:

```
swiftc program.swift
```

Use the `-v` flag to see each stage of compilation.

It is also recommended that you set the following environment variables to enable z/OS Enhanced ASCII support:

```
export _BPXK_AUTOCVT=ON
export _CEE_RUNOPTS="FILETAG(AUTOCVT,AUTOTAG)"
```

These environment variables automatically convert ASCII tagged files to EBCDIC on read.

For more details, see "Setting up Enhanced ASCII" in the IBM z/OS V2R3 Knowledge Center.

© Copyright IBM Corp. 2018

Chapter 7. Building the examples

Several examples are available under the `examples` directory. You can use `/bin/make` to build the majority of examples. To build the core examples, run the following command:

```
make
```

To execute the generated executables, run them as you would any other UNIX executable:

```
./args grapes peanuts
```

This runs the examples Swift `args` program with 2 arguments, grapes and peanuts.

Swift executables rely on the **LIBPATH** environment variable. More information about this environment variable is available in Compiling and running Swift programs.

© Copyright IBM Corp. 2018

Chapter 8. Compiling and running Swift programs

IBM Toolkit for Swift on z/OS is designed to run under IBM UNIX System Services only. Swift compilation and execution are dependent on the presence of the C Language Environment. The C Language Environment is installed by default on z/OS and therefore it does not require additional setup.

Compiling Swift programs

The IBM Toolkit for Swift on z/OS `swiftc` program compiles swift source files into program objects. IBM Toolkit for Swift on z/OS uses `/bin/cxx` as the linker. The linker takes input object files in the GOFF object format. The output of the linker is a program object.

You can specify the following supported options in the `swiftc` invocation:

-I<*search path*>
Includes the search path, which is typically for module maps.

-c Specifies compilation phase only.

-o Specifies the output file. If **-c** is specified, the output file is the object file.

-v Enables verbose display.

-D<*value*>
Marks a conditional compilation flag as true.

-Xcc Specifies the options to the C-based module-map headers.

-emit-assembly
Specifies to emit assembly output.

-Xlinker<*arg*>
Specifies the additional link arguments passed down to `/bin/cxx`.

-L<*search path*>
Specifies the link search path.

-l<*library*>
Specifies the library to link to.

-o<*outputfile*>
Specifies the output name.

-use-ld=<*value*>
Specifies an alternative linker to `/bin/cxx`.

-emit-executable
Builds executables. By default, this option is enabled.

-emit-library
Builds DLLs.

-O Specifies to optimize code during compilation. By default, this option is disabled

-Onone
Specifies not to optimize code during compilation. By default, this option is enabled.

© Copyright IBM Corp. 2018

-[no]unicode-output
-unicode-output displays compiler output as Unicode. **-nounicode-output** displays output as IBM-1047. By default, **-nounicode-output** is enabled.

-g Emits debug information.

When the **-emit-library** option is specified, the linker option **-Wl,dll** is automatically added. This causes an exports file (`.x`) to be produced for the user DLL. You must specify the `.x` file when dynamic binding against your program that is referencing the DLL.

The `swiftc` compiler accepts the following inputs:

- Path to swift source files with the extension `.swift`
- Path to static libraries with the extension `.a`
- Path to DLL sidedecks with the extension `.x`
- Path to GOFF objects with the extension `.o`

Typical invocations are as follows:

`swiftc -c -v -o a.o a.swift`
Generates an object file `a.o` and displays verbose output.

`swiftc -emit-assembly -o a.s a.swift`
Generates a psuedy assembly file `a.s`.

`swiftc -o a.exe -L/usr/lib -lmylib a.o`
Links `a.o` and `libmylib.a` into `a.exe`.

`swiftc -v a.swift`
Generates compile and link `a.swift` and generates an executable `a`, showing verbose output.

Running Swift programs

By default, when the `swiftc` compiler generates an executable, it dynamically links to the Swift core component DLL sidedecks. In order to execute the Swift compiler and the resulting Swift executables, you must set the **LIBPATH** environment variable to the location of the installed DLL libraries. The location is typically set to `${INSTALL_DIR}/lib/swift/zos`, where `INSTALL_DIR` is the location of your Swift installation. This setting is already handled by the `profile` script generated by the installer.

Chapter 9. Codepage considerations

IBM Toolkit for Swift on z/OS is a Unicode-compliant compiler. It supports Unicode strings, characters, and identifier names. Internally, the Swift compiler considers all strings, characters, and variables as Unicode (ASCII/UTF-8/16/32).

Compile time codepage considerations

The IBM Toolkit for Swift on z/OS `swiftc` compiler supports Swift source files in the following codepages:

- EBCDIC IBM-1047 or IBM-037
- ASCII/UTF-8/16/32

To support EBCDIC, Swift performs an automatic conversion on data read from EBCDIC IBM-1047 or IBM-037 to ISO8859-1 at compile time and run time.

If you are using the EBCDIC IBM-1047 or IBM-037 codepage for Swift source files, the following limitations apply:

- EBCDIC IBM-1047 or IBM-037 Swift source files cannot have Unicode strings, characters, or identifiers, which will result in a compiler error. If you do want to use special Unicode symbols, you can convert the file from EBCDIC to UTF-8 using iconv.
- As of the current release, EBCDIC codepages other than IBM-1047 or IBM-037 are not supported and will result in a compile time error. To workaround the restriction, it is recommended that you convert such files to UTF-8 using iconv:

  ```
  iconv -f <ebcdiccodepage> -t UTF-8 file.swift > newfile.swift
  ```

How Swift determines when to convert data to Unicode

Both the Swift compiler and Swift runtime have mechanisms for detecting when to convert IBM-1047 or IBM-037 text to ISO8859-1. The Swift compiler and Swift runtime only convert IBM-1047 or IBM-037 data to ISO8859-1 in any of the following conditions:

- A file is read and tagged as EBCDIC IBM-1047 or IBM-037 with `txtflag=ON`.
- A file is read and not tagged, and that file is determined as IBM-1047 or IBM-037 when the first 4096 or the maximum bytes of data is in the encoding of IBM-1047 or IBM-037 text.
- A pipe is read for more than 10 bytes and the data is in the encoding of IBM-1047 or IBM-037 text.

Once the determination is made that the file is EBCDIC IBM-1047 or IBM-037, it is cached, and subsequent reads to the same file descriptor result into a conversion from IBM-1047 or IBM-037 to ISO8859-1.

Encoding of Swift compiler messages

The `swiftc` compiler can emit compiler warnings, error messages, or verbose messages. By default, such messages are emitted as IBM-1047 or IBM-037. This behavior does present some limitations; for example Swift source files that contain Unicode symbols will not be converted properly to EBCDIC. If you have a Unicode-aware z/OS with a Unicode-aware shell, you can use the **-unicode-output**

© Copyright IBM Corp. 2018

option. With the **-unicode-output** option set, stdout and stderr messages are no longer converted from the internal Unicode format to EBCDIC IBM-1047 or IBM-037 if redirected to a file.

Runtime codepage considerations

There are several encoding considerations to consider when you write applications with IBM Toolkit for Swift on z/OS.

Writing to stderr or stdout and reading from stdin

When you write Swift applications, it is important to remember that all strings are represented as Unicode in Swift. When writing text or data to stderr or stdout, it is written verbatim and no conversion is performed except in the following cases:

- When writing is performed to a z/OS EBCDIC terminal via stdout or stderr, the data is automatically converted from the internal Swift codepage of Unicode to EBCDIC IBM-1047 or IBM-037.

You can control this behaviour by specifying an additional `to:` parameter to the Swift `print(_:separator:terminator:to:)` function. The `to:` parameter accepts an inout Target. The zOSSwift library located in the `extras` directory of the installed directory consists of the following IBM-1047 and IBM-037 Target struct definitions:

- ebcdic037_stdout for stdout
- ebcdic037_stderr for stderr
- ebcdic1047_stdout for stdout
- ebcdic1047_stderr for stderr

The following examples illustrates how to print Swift String data to stdout and how to print Swift String data to stderr in IBM-1047 or IBM-037 using the zOSSwift library.

```
import zOSSwift
import Foundation

// Print IBM-037 to stdout
var stdout = ebcdic037_stdout()
print("Printing to Stdout", to: &stdout)

// Print IBM-037 to stderr
var stderr = ebcdic037_stderr()
print("Printing to Stderr", to: &stderr)

// Print IBM-1047 to stdout
var stdout = ebcdic1047_stdout()
print("Printing to Stdout", to: &stdout)

// Print IBM-1047 to stderr
var stderr = ebcdic1047_stderr()
print("Printing to Stderr", to: &stderr)

// Print to stdout without additional to: parameter
print("Hello World") // If stdout is a terminal, it will convert \
String to IBM-1047 or IBM-037, otherwise it will print in UTF-8
```

For stdin, IBM Toolkit for Swift on z/OS exposes the Swift standard library **readLine** function and the zOSSwift `extras` library **zOSSwift.readLine** function. The Swift standard library **readLine** function performs the heuristic (described above) to determine if the data is in IBM-1047, IBM-037, or UTF-8, and converts to UTF-8 only if input data is in IBM-1047 or IBM-037. On the other hand, you can

avoid this heuristic and be explicit in the conversion by using the **zOSSwift.readLine** function, which accepts an `encodingFrom:` parameter. This parameter specifies that input data will be converted from the specified encoding to the UTF-8 Internal swift encoding. The following example demonstrates how to read IBM-1047 or IBM-037 EBCDIC data and convert it to the internal UTF-8 Swift String representation:

```
import zOSSwift
import Foundation
// Read from stdin ebcdic1047
var x = zOSSwift.readLine(encodingFrom: String.Encoding.ebcdic1047.rawValue) \
// String.Encoding.ebcdic1047.rawValue found in Foundation library
print(x, to: &stdout)
```

Writing or reading text data to a file

IBM Toolkit for Swift on z/OS ships with a set of classes under the Foundation library that aids in File I/O.

The following example demonstrates how to write a text file using the Foundation FileManager extensions. The text is written in the ebcdic037 encoding. If you inspect the resulting file `ebcdic.txt`, it is written in ebcdic037. To read the file, you must create a new instance of a string with the path as a parameter to the constructor.

```
// ebcdic.swift
import Foundation

let writeString = "Text written as EBCDIC"

let fileName = "ebcdic.txt"
do {
    // Write to the file ebcdic.txt, specifying encoding to be .ebcdic037 \
    //(exposed on Foundation library)
    try writeString.write(toFile: fileName, atomically: true, encoding: .ebcdic037)

} catch let error as NSError {
    print("Failed writing to file: " + error.localizedDescription)
}

// Read text data back
do {
    let file = URL(fileURLWithPath: fileName);

    // Convert string back to Swift internal represenation of UTF-8
    let readString = try String(contentsOf: file, encoding: .utf8)
    print("Reading string: \(readString)")
} catch let error as NSError {
    print("Failed reading file: " + error.localizedDescription)
}
```

If you compile with `swiftc ebcdic.swift` and run `./ebcdic`, you get the following output:

```
Reading string: Text written as EBCDIC
```

Writing or reading binary data using Foundation Data class

The Foundation Data class is recommended to handle raw bytes in memory, and to read and write binary data in Swift. This approach is recommended if you do not read textual data and to prevent a loss of information from possible conversion of data to UTF-8.

The following example illustrates how to write binary data using an array of bytes. It then reads the created binary file and converts it into a Data object. The example concludes by iterating through each byte and printing the hexadecimal value:

```
//bin.swift
import Foundation

let path = "bin.txt"

// File needs to be created first
FileManager.default.createFile(atPath: path, contents: nil)

// Files are represented as URLs in Swift
let purl = URL(fileURLWithPath: path);

// Binary data
var d = Data(bytes: [0xF1, 0xF1]);
d.append(contentsOf: [0xF2, 0xF2]);

print("Writing data to file: \(path)");
print("Data written: \(d)");

// Open file handle and write data
let fh = try! FileHandle(forWritingTo: purl)
fh.write(d)
fh.closeFile()

// Read from file
print("Reading binary data from file: \(path)");
let data = try! Data(contentsOf: purl)

print("Iterating and printing out each byte")
data.forEach { byte in
    let hex = String(format:"%2X", byte)
    print(hex)
}
```

If you compile with `swiftc bin.swift` and run `./bin`, you get the following output:

```
Writing data to file: bin.txt
Data written: 4 bytes
Reading binary data from file: bin.txt
Iterating and printing out each byte
F1
F1
F2
F2
```

Tagging files

To aid in reading ASCII or UTF-8 files, make use of the z/OS Enhanced ASCII support, including the chtag commmand. This command tags a file with the appropriate codepage. Once tagged, a file can be read by virtually any command, such as `cat` or `vi`.

```
chtag -tc ISO8859-1 <filename>
```

To verify that a file is tagged as an ASCII text file, use the following command:

```
ls -T <path/to/ascii/text/file>
```

You get the following output:

```
t ISO8859-1   T=on  path/to/ascii/text/file
```

Chapter 10. Interoperating with other languages

With IBM Toolkit for Swift on z/OS, you can interoperate with existing z/OS libraries written in ASM, PL/I, and C/C++ as long as they are compiled with XPLINK 64-bit. If you need to communicate to 31/24-bit libraries via Swift, the recommended approach is to communicate via z/OS Connect. More information about z/OS Connect is available at z/OS Connect Enterprise Edition on IBM Marketplace.

Using Libc to call extended C runtime routines or wrap C LE routines with ASCII-aware routines

IBM Toolkit for Swift on z/OS ships the Libc Swift module as part of its main components. You can call extended C runtime routines using Libc, as this module exposes the z/OS C runtime library, as well as newly implemented POSIX routines. To use the Libc module, simple import it and call one of the routines as follows:

```
import Libc

var x = cos(5.0) // Call C double cos(double x)

print(x)
```

In addition to exposing z/OS C runtime libraries, the Libc module wraps C LE routines with ASCII-aware routines. These routines convert ASCII strings (passed from Swift) into EBCDIC 1047 strings, which are subsequently passed into the C LE routine.

For example, a call to **fputs** from Swift does not mean it is a call to the C LE **fputs** routine, but a call to a wrapper routine **__fputs_u** in the Libc module. **fputs** is mapped to **__fputs_u** in a header file. The header files that define the mappings are located in the `/lib/swift/ebcdic_unicode/` installed directory.

If you wish to expose the original EBCDIC 1047 C LE routines, you can create a C module map to do so.

Using a module map to expose a C header to Swift

IBM Toolkit for Swift on z/OS also provides a module map mechanism for interlanguage calls between Swift and the desired language. Interlanguage calls are made possible via a module map file which exposes the location to a C header interface to Swift containing the set of routines. Since Swift is a natively compiled language, it can directly link to existing libraries. On z/OS, these libraries must be compiled with 64-bit and XPLINK and must be in the GOFF object format.

The module map format requires a C header. Note that a C header interface can be provided for routines written in C++, ASM, and PL/I.

C interoperability

In this case, the header `myheader.h` provides the interface (function declarations) to Swift via the module named `MyModule`.

© Copyright IBM Corp. 2018

Assuming `myheader.h` contains the following function declaration:

```
int sum(int a, int b);
```

The corresponding `myheader.c` implementation is provided below:

```
int sum(int a, int b) { return a + b; }
```

In Swift, in order to make use of the module `MyModule`, you must specify the following import statement at the top of the source file:

```
//main.swift
import MyModule

let x = sum(5, 5)
print(x)
```

The C function can then be called directly in the Swift source file. Swift has mappings of C data types to Swift data types. More information is available at Interacting with C APIs.

In order to build this example, you must first compile the C source file using the IBM XL C compiler with the **-q64** option into an object, static library (`.a`) or dynamic library (DLL). Then you must compile and link the Swift source file with the corresponding library, making sure that you are indicating the path to the module map using the **-I** option.

```
xlc -q64 -o myheader.o myheader.c
swiftc -I/location/to/module.map -o main main.swift myheader.o
```

PL/I interoperability

In order to expose routines from non-C languages, you must use a C header file as the bridge. In the case of PL/I, you must create a function declaration for every given PL/I function that you wish to expose to Swift.

For example, given the following PL/I routine:

```
*process display(std);
 write: proc(k,v) ext("writepair")
   returns(fixed bin(63) byvalue);
   dcl k pointer byvalue;
   dcl v pointer byvalue;
   dcl akey char(16) varz based(k);
   dcl avalue char(32) varz based(v);
   dcl key char(16);
   dcl value char(32) varz;
   display("In PLI program");
   call pliebcdic(addr(key),addr(akey),length(akey));
   call pliebcdic(addr(value),addr(avalue),length(avalue));
   display(key);
   display(value);
   return(0);
 end;
```

This routine can be exposed via the C header `myheader.h`:

```
int writepair(const char *,const char *);
```

You can then call the writepair PL/I routine in Swift as follows:

```
let rc = writepair("stuff","of value")
```

The same mechanism can be applied to ASM and C++. Interlanguage calls with COBOL are currently not directly possible as the COBOL compiler can only produce 31-bit objects. IBM Toolkit for Swift on z/OS produces 64-bit modules at this time.

Swift is a Unicode language. Keep this in mind when making interlanguage calls and when passing in strings. Strings are almost always passed in as ASCII strings and must be converted or handled appropriately.

Chapter 11. Extra Swift modules

IBM Toolkit for Swift on z/OS ships with extra modules, located in the `extras` directory of the installed directory. The `extras` directory is always searched during your Swift compilation. It contains swiftmodules, DLLs, and archives containing DLL sidedecks. Compiling modules in the `extras` directory is automatically handled by the `swiftc` compiler as long as you have `import ModuleName` in your source file. Here is an example:

```
import zOSSwift
```

Note that currently zOSSwift is the only extra library available. In order to run applications compiled with the extra modules, your **LIBPATH** setting must also point to the `extras` directory in the installed directory. Here is an example:

```
export LIBPATH="${INSTALL_DIR}/extras":$LIBPATH
```

An example is provided in the `examples/zOSSwift-library/` path of the installed directory. Here's the zOSSwift Library Interface:

```
/// TextOutput Target for EBCDIC 037 Stderr
/// Supply as input parameter to print(to:)
struct ebcdic037_stderr : TextOutputStream {
init()
mutating func _lock()
mutating func _unlock()
mutating func write(_ string: String)
}

/// TextOutput Target for EBCDIC 037 Stdout
/// Supply as input parameter to print(to:)
struct ebcdic037_stdout : TextOutputStream {
init()
mutating func _lock()
mutating func _unlock()
mutating func write(_ string: String)
}

/// TextOutput Target for EBCDIC 1047 Stderr
/// Supply as input parameter to print(to:)
struct ebcdic1047_stderr : TextOutputStream {
init()
mutating func _lock()
mutating func _unlock()
mutating func write(_ string: String)
}

/// TextOutput Target for EBCDIC 1047 Stdout
/// Supply as input parameter to print(to:)
struct ebcdic1047_stdout : TextOutputStream {
init()
mutating func _lock()
mutating func _unlock()
mutating func write(_ string: String)
}

/// Returns a string read from standard input through the end of the current
/// line or until EOF is reached.
///
/// Standard input is interpreted as Encoding specified and converted to UTF-8.
/// Invalid bytes are replaced by Unicode [replacement characters][rc].
///
```

© Copyright IBM Corp. 2018

```
/// - Parameter strippingNewline: If `true`, newline characters and character
/// combinations are stripped from the result; otherwise, newline characters
/// or character combinations are preserved. The default is `true`.
/// - Parameter encoding: Specifies the encoding to convert from.
/// - Returns: The string of characters read from standard input. If EOF has
/// already been reached when `readLine()` is called, the result is `nil`.
func readLine(strippingNewline: Bool = default, encodingFrom enc: UInt) -> String?
```

Chapter 12. Building Swift applications with Swift Package Manager

The Swift Package Manager can be used to build Swift packages as either libraries or executables. The Swift Package Manager documentation is available at https://swift.org/package-manager/.

An example package is provided under `examples/swift-package-example/`.

To build a package, you can run the following command:

```
swift build
```

To build and run a package (if an executable target exists), you can run the following command:

```
swift run
```

In order to build remote Swift Packages from github.com, you must set up Git. IBM Toolkit for Swift on z/OS ships with a version of Git which can be used to clone packages from github.com and subsequently build them via the Swift Package Manager.

You can find more information on the Swift Package Manager from https://swift.org/package-manager/.

© Copyright IBM Corp. 2018

Chapter 13. Limitations

Limitations for IBM Toolkit for Swift on z/OS are listed as follows.

Compile time and link time limitations

- The **-g** option does not generate DWARF information.
- You cannot statically link Swift objects with non-Swift 32-bit objects.
- Swift interpreter/REPL mode is not supported. For example, you must use `swiftc <sourcefile>` instead of `swift <sourcefile>`.

Run time limitations

- The CICS®, MVS™ Batch, and IMS™ subsystems are not supported.
- At this time, Swift does not support binary compatibility between applications and libraries compiled with different versions (including updates) of Swift. Run-time problems might result when either of the following conditions is met:
  - You mix applications and libraries compiled using different versions.
  - You run applications and libraries compiled from one version using a Swift runtime from another version.

© Copyright IBM Corp. 2018

Chapter 14. Troubleshooting

Troubleshooting compile time problems

If you are experiencing encoding related issues, note that only EBCDIC 1047 and ASCII/UTF-8 are supported. You might encounter unrecognized character errors if another codepage is used.

Specify **-v** on the `swiftc` command line to emit more information about the compilation steps. Diagnose by running each generated command manually one by one.

Troubleshooting run time problems

If the Swift application abends, you have the following options for debugging:

- Using `Thread.callStackSymbols` from Foundation to get stack trace in Swift. More details are available at callStackSymbols.
- Inserting **fatalError()** calls in the Swift source code. These calls generate a CEEDUMP, which includes a traceback.
- Using dbx for instruction level debugging and generating a psuedo-assembly via the **-emit-assembly** option in the `swiftc` compiler.

When you specify both the **-g** and **-emit-assembly** options, the compiler inserts **.loc** annotations into the resulting assembly output. These annotations are of the form `.loc` <*file*> <*line*> <*column*> <*optional flag*> as defined in the DWARF standard. Below is a code example:

```
let PI=3.14159;
let rad=2.0;
var area=PI*(rad*rad);
```

Below is the resulting assembly output. From the `.loc 1 5 18 is_stmnt 0` line, you can see that the <*optional flag*> is `is_stmnt 0`.

```
.loc     1 1 10 prologue_end  #test.swift:1:10
lg       %r1, 0(%r5)
llihf    &r0, 1074340345
oilf     &r0, 4028335726
stg      &r0, 0(%r1)
.loc     1 3 11               #test.swift:3:11
lg       %r3, 8(%r5)
llihh    %r0, 16384
stg      %r0, 0(%r3)
.loc     1 5 12               #test.swift:5:12
ld       %f0, 0(%r1)
.loc     1 5 18 is_stmt 0     #test.swift:5:18
ld       %f1, 0(%r3)
.loc     1 5 22               #test.swift:5:22
mdbr     %f1, %f1
.loc     1 5 15               #test.swift:5:15
mdbr     %f0, %f1
lg       %r1, 16(%r5)
std      %f0, 0(%r1)
lghi     %r3, 0
.loc     1 0 0                #test.swift:0:0
```

© Copyright IBM Corp. 2018

```
stg      %r2, 2176(%r4)       #8-byte Folded Spill
lg       %r7, 2072(%r4)
aghi     &r4, 160
b        2 %r7
```

Tip:

To improve the experience when you work with Unicode source files and data, you can set the `export _BPXK_AUTOCVT` and `export _CEE_RUNOPTS="FILETAG(AUTOCVT,AUTOTAG)"` environment variables. These environment variables perform an auto conversion from ASCII to EBCDIC on read and vice versa on write only if the file is tagged.

Chapter 15. Support

Contacting IBM

Read already-posted Swift on z/OS Github Issues to check if your question has already been answered. If not, open a new Swift on z/OS issue on Github for us. Include your version number in the post.

To establish what version of IBM Toolkit for Swift on z/OS is in use, run the following command:

```
swiftc --version
```

Other information

- IBM Toolkit for Swift on z/OS in Developer Centers
- Swift Documentation
- Swift programming language GitHub
- "Interacting with C APIs" in the Swift Documentation

© Copyright IBM Corp. 2018

Notices

Programming interfaces: Intended programming interfaces allow the customer to write programs to obtain the services of IBM Toolkit for Swift on z/OS.

This information was developed for products and services offered in the U.S.A. IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive, MD-NC119
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan, Ltd.
19-21, Nihonbashi-Hakozakicho, Chuo-ku
Tokyo 103-8510, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:
INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM websites are provided for convenience only and do not in any manner serve as an endorsement of those

© Copyright IBM Corp. 2018

websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who want to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

Intellectual Property Dept.
IBM Corporation
5 Technology Park Drive
Westford, MA 01886
U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrates programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating

platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided “AS IS”, without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

Each copy or any portion of these sample programs or any derivative work, must include a copyright notice as follows:

© (your company name) (year). Portions of this code are derived from IBM Corp. Sample Programs. © Copyright IBM Corp. 2018.

PRIVACY POLICY CONSIDERATIONS:

IBM Software products, including software as a service solutions, (“Software Offerings”) may use cookies or other technologies to collect product usage information, to help improve the end user experience, or to tailor interactions with the end user, or for other purposes. In many cases no personally identifiable information is collected by the Software Offerings. Some of our Software Offerings can help enable you to collect personally identifiable information. If this Software Offering uses cookies to collect personally identifiable information, specific information about this offering's use of cookies is set forth below.

This Software Offering does not use cookies or other technologies to collect personally identifiable information.

If the configurations deployed for this Software Offering provide you as customer the ability to collect personally identifiable information from end users via cookies and other technologies, you should seek your own legal advice about any laws applicable to such data collection, including any requirements for notice and consent.

For more information about the use of various technologies, including cookies, for these purposes, see IBM's Privacy Policy at http://www.ibm.com/privacy and IBM's Online Privacy Statement at http://www.ibm.com/privacy/details in the section entitled “Cookies, Web Beacons and Other Technologies,” and the “IBM Software Products and Software-as-a-Service Privacy Statement” at http://www.ibm.com/software/info/product-privacy.

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available at “Copyright and trademark information”.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Index

© Copyright IBM Corp. 2018

Printed in USA

SC27-9267-00