Table of Contents

Notices....1
Introduction....1
  Acquiring and Installing SATK....2
  Using SATK....3
  Bare-Metal Programs with SATK....3
    SATK Mainframe Platform Usage....3
Functional Area Capabilities....4
ASMA – A Small Mainframe Assembler....5
  Machine Specification Language (MSL)....6
  Architecture Target Integration....6
IPLASMA – IPL Medium Creation....8
XCARD – Extended Card Loader....9
Function Calls....10
Hardware Access....11
  Assigned Storage Area – ASA....11
  Input/Output Devices....11
Synchronous I/O....12
Specific Tools....13
SDL Hercules....14
Samples....15
  samples/asma....15
  samples/guide....15
Bibliography....17
Appendix A – SATK Functional Content....18
Appendix B – Historical or Deprecated Content....20

Notices

Introduction

The *General Information Manual* provides a broad overview of the Stand Alone Tool Kit (SATK) project. SATK’s focus is making **easier** development of software not requiring an operating system (stand alone or bare-metal). **Easier** is focused on reducing the learning

curve and time involved in development. In particular, SATK is always sensitive to where facilities can be provided or enable the developer to produce reusable program content.

Bare-metal programs target for execution specifically IBM® mainframes, current or vintage, or compatible platforms. In this document, simply “mainframes”.

As the name implies, SATK’s primary focus is on *tools*. Tools may be usable by a software developer or may be software supplied by SATK for inclusion in a developed program. Developer used tools execute on commonly available personal computing platforms. The developed programs using SATK execute on a mainframe. The goal of both forms of tools is making the effort to create the final result easier, namely, the bare-metal program.

SATK augments the tools with documentation. Frequently the documentation collects together specific developer used tools with related supplied software into a single functional area, or “use case”. Of primary interest to developers are the SATK tool providing an assembler, A Small Mainframe Assembler (`ASMA`), and the IPL medium creation tool, `IPLASMA`. These and additional functional areas are discussed below.

The reader of this manual is benefited by having a background in mainframe *systems,* their *operations*, and *assembly language* development. If the reader does not have this background, SATK is the perfect vehicle for acquiring experience in these areas. SATK is definitely the *only* effort explicitly targeting bare-metal mainframe programming.

Development of a bare-metal program requires access to the *IBM Principles of Operation* manual related to the system’s architecture targeted by the program and, when input/output (I/O) operations are performed, the related devices documentation, and the system’s I/O subsystem described in the *Principles of Operation* manual.

Acquiring and Installing SATK

SATK is a project available in a repository on `github`:

https://github.com/s390guy/SATK

Clone this repository or download it to your local system. SATK is now installed. No build process is required to install SATK.

If a `git` tool is available, cloning is recommended. Changes occur and it is easiest to simply pull from the `github` repository directly to get the latest software.

SATK is licensed under the GPL3 open source license.

All relative file system paths in this document are relative to the SATK root directory.

Using SATK

SATK requires **Python** 3. All SATK tools are written in **Python** 3. SATK may be used on any system on which **Python** 3 may be installed. The manual titled *A Small Mainframe Assembler* located in the SATK directory at `doc/asma/ASMA.pdf` contains details on getting started with SATK and in particular installing **Python** 3 and using the supplied assembler, ASMA.

Once **Python** is available, all SATK tools are usable. Software supplied by SATK is now available for inclusion in developed software.

Bare-Metal Programs with SATK

All SATK bare-metal programs are written using ASMA. ASMA assembler is similar to legacy IBM assemblers and is readily recognizable and usable by someone familiar with such assemblers. SATK supplied software takes the form of macros provided in macro libraries as part of the SATK repository contents. As with ASMA assembler, ASMA macros are readily recognized by someone familiar with IBM type assemblers.

The bare-metal program is placed in various output formats by ASMA. The List-Directed IPL Directory (LDID) format has proven to be the most useful. Some emulators can use this format directly. The LDID format is used as input to the IPL medium creation tool, IPLASMA, from which an emulated device may be used by a mainframe emulator or simulator.

For use with a physical mainframe platform, the bare-metal program must be transferred to it. An emulated card deck has been useful for such transfers and `IND$FILE` has proved useful for both legacy and modern platform transfers. The actual transfer is specific to the destination system and the system on which SATK is used. Emulated tapes, using the AWS format, have also proved useful for platforms supporting them.

The “ASMA Assembler” and “IPL Medium Creation” sections, below, provide additional details.

SATK Mainframe Platform Usage

Bare-metal programs have been successfully used on the following mainframe platforms:

- SDL Hercules (emulator),
- SimH (simulator),
- z/PDT (emulator), and
- z/VM® virtual machines, various software releases and hardware systems (physical).

Functional Area Capabilities

SATK delivers capabilities collected together into related usage. The capabilities are described by documentation for the supplied tools and reusable assembler macro libraries. Each functional area is described further in the following sections.

ASMA – The mainframe assembler used to create the binary content of a bare-metal program.

IPLASMA – Converts the bare-metal program's binary content into an emulated IPL medium.

Hardware Access – A set of reusable macros targeting commonly encountered aspects of hardware access by a bare-metal program.

Function Calls – Supports using and defining callable functions.

Synchronous I/O – Performs I/O operations without an explicit interrupt handler.

Extended Card Loader – Bare-metal object deck boot loader, eliminating itself from memory following the load.

SDL Hercules – Documentation and macros useful when using the SDL Hercules emulator.

ASMA – A Small Mainframe Assembler

ASMA is the only open source mainframe oriented assembler usable outside of a mainframe operating system. ASMA is specifically designed for use in bare-metal programming of mainframe systems. All mainframe architectures are supported. The assembly language supports all instruction and extended mnemonics provided by the first mainframe, announced in 1964, through the latest announced in May, 2022.

ASMA is an assembler similar in some respects to vintage mainframe assemblers. It is a two-pass assembler. The assembler and macro languages supported by ASMA are similar to early mainframe assemblers. The assembler language lacks the limitations of those early assemblers by supporting all machine instruction formats and extended mnemonics:

- all address modes: 24, 31, and 64;
- eight-byte maximum address and integer constants;
- long displacements;
- relative addressing;
- all immediate operands;
- binary and decimal floating point instructions; and
- vector instructions.

Capabilities available only with ASMA are:

- multiple architecture contexts: instruction mnemonics, `PSW` and `CCW` formats (see “Architecture Target Integration” below);
- multiple output formats;
- bare-metal oriented assembler operations, for example, architecture specific PSW operations with common syntax;
- enhanced binary content handling with regions, a collection of control sections, by an extended `START` operation;
- allows limiting of instruction and extended mnemonic recognition for the targeted architecture of the program;
- integrated linking of all address locations independent of the resident control section, eliminating the need for a linkage editor; and
- EBCDIC to/from ASCII code page modifications.

The following table describes in general terms the output formats available with ASMA.

Output Format	Description
Image	All regions concatenated together into a single binary file regardless of assembled load addresses.
Object Deck	80-byte card deck containing TXT and END records.
VM Store	Binary content stored by VM CP commands.
SDL Hercules	Binary content stored by SDL Hercules storage alteration commands.
Management Console	Binary content stored by management console commands.
List-Directed IPL Directory (LDID)	Assembled regions placed in LDID format maintaining region name and assembled load addresses.

Refer to *ASMA – A Small Mainframe Assembler* at `doc/ASMA.pdf` for details on the usage of ASMA, output format usage, and assembler and macro languages.

Machine Specification Language (MSL)

MSL supports ASMA's instruction mnemonic recognition, interpretation and binary content creation. Instruction formats are defined by MSL test files. While officially an internal component of ASMA, MSL text files externalize for the purpose of inspection this critical aspect of ASMA functionality. Making these internal details available is unique to ASMA.

A report of supported instructions by architecture in three sections: mnemonic, operation code, and MSL instruction format is provided by ASMA. This unique report is contained in the file: `asma/doc/Po0.txt` and is updated whenever the MSL database is altered. This report is intended for anyone who needs to understand how ASMA is processing (or not) any instruction mnemonic.

For a detailed description of MSL refer to *Machine Specification Language* at `doc/asma/MSL.pdf`.

Architecture Target Integration

ASMA is designed to support multiple architecture machine mnemonics. These architectures are:

- S/360 – System/360 (or S/370 in Basic Control Mode),
- S/370 – System/370 in Basic or Extended Control Mode,
- S/380 – System/380, S/370 Extended Mode with 31-bit addressing,

- 370-XA – 370 Extended Architecture,
- ESA/370 – Enterprise Systems Architecture 370,
- ESA/390 – Enterprise Systems Architecture 390,
- ESA/390 on z – Enterprise Systems Architecture on z/Architecture, and
- z/Architecture – z/Architecture.

The architecture in addition to supported base and extended instruction mnemonics, includes the concept of an “operational environment.” The operational environment includes PSW format, CCW formats and address mode. Each of the above target architectures has an implied operation environment.

In addition, all instruction mnemonics can be enabled in either of three operation environments:

- 24 bit address mode
- 31 bit address mode, and
- 64 bit address mode.

Architecture Integration exposes to the assembled program information about the target execution environment. This information is contained in various macro system symbols. By exposing this information to the program, it is possible to create programs targeting multiple architectures that change based only on the target architecture.

Macros supplied with ASMA provide this support. In addition to setting the PSW and CCW execution mode formats, various instruction mnemonics are supplied for common use across architectures, generating the correct instruction for the targeted architecture. A number of useful global macro symbols are initialized for use by macros supporting multiple architectures.

IPLASMA – IPL Medium Creation

`iplasma.py` is the tool that creates an IPL supporting medium containing a bare-metal program. IPLASMA uses one or two LDID's for creation of the IPL medium. IPLASMA assumes that any file supplied as input is an LDID control file created by ASMA.

When one file is supplied to IPLASMA, the IPL medium contains only the bare-metal program. When two files are supplied, the IPL medium contains, first, a boot loader and then, a bare-metal-program. In this second case, the boot loader is itself a bare-metal program. The boot loader participates in the IPL function and loads and passes control to the loaded bare-metal program.

Input files are only LDID control file paths. The output file is the emulated IPL medium. Supported output emulated media types are:

- 80-byte card deck, or
- Fixed Block Architecture (FBA) Direct Access Storage Device (DASD).

See "Extended Card Loader", below, for creation of AWS tape files capable of being IPL'd and additional card deck options.

See the document *Initial Program Load with ASMA*, in `doc/asma/IPLASMA.pdf`, for details concerning IPL media content creation and record formats used by a boot loader.

Some platforms can utilize directly the LDID format created by ASMA. These platforms allow the IPL function to be targeted to the LDID. Use of this option eliminates the need for IPL medium creation.

XCARD – Extended Card Loader

The Extended Card Loader is a unique offering of SATK. It allows an object deck containing the bare-metal program to perceive the system as if it had been IPL'd.

Most boot loaders using card input place themselves in low memory addresses, utilizing those addresses during the loading process. All boot loaders must use these areas to accomplish the loading process. This scenario precludes use of any traditional card oriented boot loader.

XCARD, the Extended Card Loader, addresses this need. Following the IPL, loading XCARD, the boot loader determines the end of memory and relocates itself there. It creates a surrogate page 0 into which it places all content of the loaded program destined for real page 0 (real addresses 0-4095). Following completion of the loaded object deck, the contents of the surrogate page 0 are placed in the real page 0 as if the boot loader had not been used. XCARD then erases itself from memory, clears registers, and initiates the loaded bare-metal program as if it had been IPL'd.

Unlike IPLASMA, the loaded program is an object deck. The two portions of the load stream, XCARD and the object deck, are combined into one contiguous set of card records using the `deck.py` tool. In turn the single stream can be converted to an AWS tape file by `deck.py`.

Presently, XCARD only supports the ESA/390 architecture. XCARD never changes the IPL architecture.

See *XCARD eXtended Loader* at `doc/XCARD.pdf` for details on XCARD usage and the `deck.py` tool.

Function Calls

SATK supports the use of C inspired function calls. The provided macro content aids:

- Function call entry definition,
- Function local memory usage,
- Function return to caller,
- Function call handling, and
- Function stack definition.

The macros utilize the *s390 ELF Application Binary Interface* specification for function call logic. The macros are sensitive to the target architecture, generating the appropriate instructions for the target architecture. Facilities offered by ASMA Architecture Target Integration makes this possible.

C-like function calls are the most compact and feature complete mechanism for creation of complex routine call mechanisms. C-like functions can be used both serially and re-entrant offering concurrent and simultaneous usage by multiple callers. Function processing can have implications for whether all of these usability attributes apply to a given function.

The function call macros were ported from the SATK GNU assembler macros developed during the initial, now deprecated, approach taken by SATK. Nevertheless, function calls proved exceedingly value and when SATK transitioned to use of ASMA, this value was brought forward.

Refer to *SATK Macro Category - `func`* in `doc/macros/func.pdf` for details of SATK functions, usage of the function related macros and background on ELF compatible functions.

Hardware Access

SATK provides through macros a lot of supplied support for access to the hardware. These macros are described in multiple documents, depending upon the hardware component being addressed.

While other systems may internally utilize this information, SATK specifically makes this information readily available to the bare-metal program developer. All of the macros in this SATK functional content are available in the `maclib` macro directory.

Assigned Storage Area – ASA

All mainframes regardless of architecture utilize an area of storage for specific hardware related functions. The details change with architecture, the ASA persists. IPLASMA allows for the loading of the ASA during IPL and ensures that the bare-metal program's entry PSW is placed in the ASA. ASA content uses a macro, using ASMA Architecture Integration, to tailor the content. Additionally, a target architecture sensitive DSECT is supplied that describes the structure of the ASA.

Details of usage of the ASA and the first 4096-byte page are compared in *Mainframe Architecture Comparisons*, at `doc/herc/Architecture Comparisons v0.3.pdf`.

Refer to *Stand Alone Tool Kit Common Macros* at `doc/macros/SATK.pdf` for details of ASA related macro usage.

Input/Output Devices

Performing input or output operations with a device involves some memory based structures. Over time the number of structures increased as did the complexity. The construction of these structures in the assembly and various DSECT definitions of their formats are supplied by SATK.

These macros are helpful when constructing a bare-metal program's device access capabilities.

Refer to *SATK Macro Category – io* at `doc/macros/io.pdf` for details on macro usage.

Synchronous I/O

Unlike mainframe operating systems performing input/output operations on multiple devices both concurrently and simultaneously, bare-metal programs frequently operate more like an application performing input/output operations with one device at a time. The SATK macros supporting synchronous I/O are independent of the input/output architecture in use by the system, allowing a single implementation to be used in different architectures.

These macros support:

- I/O subsystem initialization,
- Enabling of devices,
- Initiating I/O to a device,
- Waiting for its completion,
- Calculating the number of transferred bytes, and
- Detecting an error.

Information related to the device and I/O operations progress is maintained in a structure that is both created and defined by the synchronous I/O macros. This structure is required when using synchronous I/O.

The synchronous I/O content allows a bare-metal developer to implement a reusable routine for I/O operations with a specific device or device operation in multiple bare-metal contexts.

Refer to *SATK Macro Category – sync* at `doc/macros/sync.pdf` for details on macro usage.

Specific Tools

Tools used for specific narrow purposes do not have their own documentation. They rely upon the tool's command line `--help` argument to describe how to invoke the tool.

They following table describes in general terms the specific tools provided by ASMA.

Tool	Description
`tools/stfle.py`	Analyzes facility list content, displays descriptions of specific facility indicator assignments, and searches facility descriptions for facilities containing strings.
`tools/fp.py`	Decimal floating point character to interchange format conversions.

SDL Hercules

Mainframe platform options are limited, particularly outside of the professional developer arena. The SDL Hercules emulator is the most common. Emulated mainframe device content, previously mentioned, are designed to operate with SDL Hercules. IPLASMA is designed specifically for use with SDL Hercules, although other platforms can accept some device content created by IPLASMA.

Undocumented capabilities of Hercules are described in Undocumented Hercules, at `doc/herc/Undocumented.pdf`. One such capability is the DIAGNOSE X'008' instruction executing a Hercules command. Its use is integral to the macro `HRCCMD.mac` in the `maclib` directory.

Due to the simplicity of use, SATK has standardized on the use of emulated Fixed Block Architecture (FBA) Direct Access Storage Devices (DASD) rather than Count-Key-Data (CKD) DASD. As part of the suite of emulated devices, two documents are placed in the `doc/herc` directory:

- *Hercules Fixed Block Architecture Emulation Reference Manual*, at `doc/herc/FBA Manual.pdf`; and
- *Fixed-Block Architecture Structures*, at `doc/herc/FBA_DASD_Structures.pdf`.

These describe how to use FBA DASD on Hercules and the structures SATK expects to find on an FBA DASD volume.

Samples

Samples illustrating the use of various content areas are provided in the samples directory. Only the content in

- `samples/asma`, and
- `samples/guide`

contain examples oriented towards asma.

Other content within the `samples` directory relate to deprecated use of the GNU assembler and other related tools.

`samples/asma`

`samples/asma` contains some source examples that can be assembled using ASMA. They are intended to show some of ASMA's facilities. `sos.asm` is the source code of the Madnick Sample Operating System.

`samples/guide`

Each "Program **X**" chapter within the *SATK User Guide* at `doc/Guide.pdf` provides a description of the corresponding directory `samples/guide/pgm`***x***. The bare-metal programs are increasingly complex and the guide provides additional detailed descriptions of building, creating IPL media if required, and execution of the samples. All samples are designed for use with SDL Hercules.

Each program directory contains a complete implementation of the bare-metal program. This includes:

- assembler source, and
- scripts for assembling, IPL media creation when used, and execution.

Scripts are intended to be modified for the user's platform and file system structure.

This table describes in general terms each of the program directory's content documented by the *SATK User Guide*.

Directory	Content Description
pgm1	Build and execute a "Hello World" program from a List-Directed IPL Directory.
pgm2	Build and execute a "Hello World" program from a FBA DASD IPL device.
pgm3	Build and execute a "Hello World" program using console device I/O.

Directory	Content Description
pgm4	Build, boot from a FBA DASD device, and execute a “Hello World” program.
pgm5	Build and execute a Hello World program using Boot Loader Services.

Programs 1-4 are incremental increases in complexity of fundamentally similar functionality.

Program 5 (`pgm5`) is a complex sample. It creates a boot loader that can load the booted program and provide various services used by itself and the booted program. The macro library `lodrmac` contains macros for creation of the boot loader with its resident services. In addition, Program 5 has its own documentation: *BLS – Boot Loader Services*, at `doc/BLS.pdf`. This additional documentation describes how services are added and used, and the role of boot loader specific macros to services. In addition, Program 5 provides examples of using Boot Loader Services in four different target architectures. A legacy routine calling system is provided by the macros.

Program 5 is a deep dive into bare-metal programming. It also explores the idea of providing more than simple loading of a program by a boot loader.

Bibliography

The following table provides a list of SATK supplied documentation and where it can be found in the SATK repository.

Document Title	SATK Directory Location
ASMA – A Small Mainframe Assembler	`doc/asma/ASMA.pdf`
ASMA Instruction Status	`doc/asma/Instruction_Status.pdf`
BLS – Boot Loader Services	`doc/BLS.pdf`
Fixed-Block Architecture Structures	`doc/herc/FBA_DASD_Structures.pdf`
Hercules Fixed Block Architecture Emulation Reference Manual	`doc/herc/FBA Manual.pdf`
Initial Program Load with ASMA	`doc/asma/IPLASMA.pdf`
Machine Specification Language	`doc/asma/MSL.pdf`
Mainframe Architectural Comparisons	`doc/herc/Architectural Comparisons v0.3.pdf`
MSL DATABASE REPORT	`asma/msl/PoO.txt`
SATK General Information Manual	`doc/GIM.pdf`
SATK Macro Category - func	`doc/macros/func.pdf`
SATK Macro Category - io	`doc/macros/io.pdf`
SATK Macro Category - sync	`doc/macros/sync.pdf`
SATK User Guide	`doc/Guide.pdf`
Stand Alone Took Kit Common Macros	`doc/SATK.pdf`
Stand-Alone Tool Kit – Configuration System Manual	`doc/Configuration.pdf`
Undocumented Hercules	`doc/herc/Undocumented.pdf`
XCARD eXtended LOADER	`doc/XCARD.pdf`

Appendix A – SATK Functional Content

SATK treats a collection of content delivery falling within a subject area as a “functional area.” A functional area always contains documentation. It may additionally contain one or more developer tools, and/or one or more assembler macros or entire macro libraries.

These portions of a functional area are delivered using the following repository content:

- Documentation is delivered as a PDF format file (file extension `.pdf`) within the SATK `doc` directory, or a subdirectory. The PDF file is accompanied by a LibreOffice Writer document with the same name except the file extension is `.odt`. The LibreOffice document is used for documentation creation and modifications prior to being exported as the PDF file.
- Developer tools are delivered as **Python** modules in the SATK `tools` directory. All **Python** modules end with a `.py` file extension. Some reusable bare-metal programs are available in their own directory.

  Only **Python** modules in the `tools` directory referenced below are intended for public use. All other **Python** modules are considered internal for SATK use only.
- A macro library is delivered by SATK as a directory containing text files defining the macros and each file’s name matching the macro assembler name with a `.mac` file extension. Macros simplify a developer’s efforts in creating a bare-metal program.

This table documents the content, as delivered, by each functional area. “- -” means not part of the functional content.

Functional Content	Documentation	Tools	Macro Libraries
ASMA – Assembler	• asma/MSL/PoO.txt • doc/asma/ASMA.pdf • doc/asma/Instruction_Status.pdf • doc/asma/MSL.pdf • doc/Configuration.pdf	tools/asma.py tools/codepage.py tools/mslrpt.py	maclib
Extended Card Loader	• doc/XCARD.pdf	tools/deck.py xcard	--
Function Calls	• doc/macros/func.pdf	--	maclib
Hardware Access	• doc/herc/Architecture Comparisons v0.3.pdf • doc/macros/io.pdf • doc/macros/SATK.pdf	--	maclib
IPLASMA – IPL media	• doc/asma/IPLASMA.pdf	tools/iplasma.py	--
Samples	• doc/BLS.pdf • doc/Guide.pdf	samples/asma samples/guide	lodrmac

Functional Content	Documentation	Tools	Macro Libraries
SDL Hercules	• doc/herc/FBA Manual.pdf • doc/herc/FBA_DASD_Structures.pdf • doc/herc/Undocumented.pdf	--	maclib
Specific Tools	• doc/GIM.pdf	tools/fp.py	--
Synchronous I/O	• doc/macros/sync.pdf	--	maclib

Appendix B – Historical or Deprecated Content

SATK originally used the `binutils` GNU assembler and linkage editor. SATK usage of `binutils` is now deprecated. Directories remain primarily for historical purposes. The following table identifies the historical content with a brief description. These directories may usually be ignored.

Directory	Description
`doc/iplelf`	SATK use of the ELF file format
`lib`	GNU linkage editor scripts
`src`	GNU assembler source files
`tools/medfun`	Script to build a GNU as bare-metal program
`tools/satkfun`	Sourced script for use of GNU as
`tools/xbuild`	Scripts for building cross platform GNU mainframe tools
`tools/xtest`	Tests built cross platform GNU tools
`tools/xtoolfun`	Sourced script for using GNU tools
`tools/xverify`	Verifies tools built by `xbuild`.