Table of Contents

Notices

Linux® is the registered trademark of Linus Torvalds in the U. S. and other countries.

“Python” is a registered trademark of the Python® Software Foundation.

z/Architecture is a registered trademark of International Business Machines Corporation.

Introduction

XCARD is a replacement for similar proprietary boot loaders bringing into memory an object module formatted program, the booted program. XCARD is provided as part of the Stand

Alone Tool Kit (SATK). XCARD is **not** a disassembly of proprietary loaders. XCARD is a new implementation with features unavailable in proprietary loaders and made available under an open source license as is this documentation.

Additional features or adjustments can be legally made and shared as required (provided such use is consistent with the open source license or any other license under which the XCARD eXtended LOADER is released). Proprietary loaders prohibit such modifications or their distribution to others. Not so for XCARD.

As a replacement, XCARD can be used with a booted object module on either real mainframe systems or non-mainframe emulators, for example Hercules. Both the IPL stream construction and execution can occur on a mainframe system or with a mainframe emulator system. On mainframe systems, XCARD may be used in conjunction with a virtual machine system or a configured logical partition, LPAR.

Two methods exist for communication with the SATK community providing XCARD. SATK uses a `groups.io` email list: `satk`. Join the group or post on the SATK `github` repository.

Installing XCARD

XCARD is available from the SATK github repository https://github.com/s390guy/SATK. The repository may be either cloned or downloaded as a zip file to a local system.

While this documentation reflects Linux or similar usage, SATK is fully functional on all operating systems supporting **Python** 3 with a few adjustments for file paths and small execution scripts.

XCARD for the mainframe can be immediately used after SATK installation by simply transferring the binary XCARD image to the mainframe. The following table identifies which SATK file supports which target environment. “N/A” means not available. “TBD” means to be determined. “WIP” means work in progress.

IPL Architecture	Mainframe Support	Hercules Support	SATK XCARD Image
S/360	N/A	S/370 BC-mode	planned
S/370	N/A	S/370 EC-mode	planned
ESA/390 on z	WIP	WIP	`xcard/s390/xcard.deck`
z/Architecture®	TBD	TBD	not available

This documentation is available in the SATK directory: `doc/XCARD.odt` or `doc/XCARD.pdf`.

References

A Small Mainframe Assembler, `doc/asma/ASMA.odt` or `doc/asma/ASMA.pdf`

Initial Program Load with ASMA, `doc/asma/IPLASMA.odt` or `doc/asma/IPLASMA.pdf`

Using XCARD

XCARD is a boot loader for an object module. Each object module record must be 80-bytes in length. The IPL medium can be an emulated card deck (without line terminations) or an AWS tape file. The object module being booted by XCARD must immediately follow XCARD in the sequence of records.

The externally initiated IPL function will load and transfer control to XCARD. XCARD will then read the object module from the input stream. XCARD ceases reading when it finds the first object module END record.

XCARD only recognizes object module TXT records and the first encountered END record.

Consequently, XCARD ignores:

- all records read that are not object module record types (the first byte of the record is not X'02'),
- ESD type records, or
- RLD type records.

When XCARD encounters a TXT type record, memory will be loaded with the content specified by the record at the real address in the record. Memory content destined for the first 4,096 bytes of memory will be preserved elsewhere. Before control is passed to the object module, the first 4,096 bytes of memory will be set to the preserved values. Bytes not specified by the object module within the first 4,096 bytes are set to binary zeros.

Following the reading of the object module, XCARD clears to binary zero memory containing itself and any memory used during the booting process, including the 4096 bytes of preserved values. XCARD, as its last action, transfers control to the object module.

It is the users responsibility to ensure the record stream on the IPL medium contains in this order and without any interruptions:

- XCARD, and
- the boot loaded object module.

For example, any encountered tape marks on an AWS tape result in an interruption and an error condition. A missing END record will also result in an error condition.

Additional content including tape marks are allowed provided such content is accessed by the booted object module (or even later memory resident software) rather than XCARD itself.

The following sections assume that SATK has been installed on an available system. This installation is the foundation for XCARD usage.

XCARD either transfers control to its loaded object module or ends in a disabled wait state. The test program, XCARDTST, always ends in a disabled wait state. Termination conditions are broadly categorized by the source in the first byte of the abnormal termination code:

- 00xxxx – an unexpected interruption during XCARD execution
- 01xxxx – an abnormal condition detected by XCARD
- 02xxxx – a XCARDTST failure, unexpected interruption, or successful termination.

XCARD Abnormal Termination Conditions

Whenever XCARD is unable to pass control to the booted object module, it will terminate abnormally with a disabled wait PSW. The reason for the abnormal termination is indicated in the last three bytes of the PSW instruction address. No other output is generated by XCARD.

The following table explains the meaning of the abnormal termination codes.

Abend Code	Meaning
000001	Input/Output interruption failed to occur. XCARD is waiting.
000008	Unexpected restart interruption. Restart Old PSW at real address 000008
000018	Unexpected external interruption. External Old PSW at real address 000018
000020	Unexpected supervisor call interruption. Supervisor Call Old PSW at real address 000020
000028	Unexpected program interruption. Program Old PSW at real address 000028
000030	Unexpected machine check interruption. Machine Check Old PSW at real address 000030
000038	Unexpected input/output interruption. Input/Output Old PSW at real address 000038
010000	XCARD terminated without passing control to the object module
010004	Memory size could not be determined.
010008	Movement of data failed due to unequal source and destination lengths
01000C	Destructive overlap detected when moving memory contents
01xx10	IPL device not operational or available. xx indicates detecting instruction: • 33 – START SUBCHANNEL • 35 – TEST SUBCHANNEL
01xx14	IPL device is busy. xx indicates detecting instruction: • 33 – START SUBCHANNEL • 35 – TEST SUBCHANNEL
013518	IPL device interruption information does not contain status. Detecting instruction is TEST SUBCHANNEL

Abend Code	Meaning
01351C	TEST SUBCHANNEL returned a condition code of 2. Should not occur.
010020	Object module attempted to overlay XCARD or preserved values in high memory.
01xx24	Channel error occurred during input/output operation. xx indicates the channel status error condition(s).
01xx28	Unit error occurred during input/output operation. xx indicates the unit status error condition(s).
01002C	Physical end-of-file condition detected by device: • end of records on card device, or • tape mark on a tape device.

XCARDTST Termination Conditions

XCARDTST examines the state in which memory is passed to it by XCARD and inspects it for the conditions expected. Areas set by XCARDTST are not examined unless the content is part of the testing of XCARD. XCARDTST always terminates in a disabled wait state. The following table describes the termination conditions. All but the first are considered abnormal.

Abend Code	Meaning
020000	Successful execution. All tests passed.
020004	I/O IPL device information not passed by XCARD.
020008	Area from real addresses X'8' to X'67' is not binary zeros. See general register 4 for the address of the first non zero byte encountered.
02000C	Area from real addresses X'70' to X'B7' is not binary zeros. See general register 4 for the address of the first non zero byte encountered.
020010	Area from real addresses X'C4' to X'2FF' is not binary zeros. See general register 4 for the address of the first non zero byte encountered.
020014	Area from real addresses X'428' to X'FF5' is not binary zeros. See general register 4 for the address of the first non zero byte encountered.
020018	Area from real addresses X'103E' to end-of-memory minus 6 is not binary zeros. See general register 4 for the address of the first non zero byte encountered.
02001C	Data between X'FF6' and X'103D' has been corrupted during object module loading.
020028	Unexpected program interruption. Program Old PSW at real address 000028

With a Mainframe Virtual Machine

To be supplied

With a Mainframe Logical Partition

To be supplied

With Hercules

SDL Hyperion 4.x or later is assumed to be the Hercules platform in use. Older versions, such as Hercules 3.13 are possible, but not discussed here. Execution of SDL Hyperion is on a non-mainframe system. A prerequisite for use of Hercules is that Hercules is installed on an available system, typically the same system as SATK.

This section assumes that the XCARD IPL medium as a card deck is available. This is located in the `xcard/s390/xcard.deck` file. It also assumes that the object module to be booted by XCARD is available. The SATK assembler, ASMA, may be used to create this object module. See the ASMA manual for how to use the `-o` output option.

Testing XCARD

The SATK XCARD directory already contains two IPL media ready for testing XCARD.

- `load.deck` – the emulated virtual card deck file, and
- `load.aws` – the emulated virtual AWS tape file.

Each file contains an IPL capable medium stream containing the XCARD eXtended LOADER and the test program object module booted by XCARD.

The SATK XCARD directory also contains corresponding scripts for running the test program:

- `iplc` – IPL's XCARD and executes the test program from an emulated card deck file, and
- `iplt` – IPL's XCARD and executes the test program from an emulated AWS tape file.

Each script is accompanied by its corresponding Hercules run command file that performs the IPL and a configuration file for the IPL medium being used.

Use `iplc` or `iplt` with any local modifications from the SATK `xcard/s390` directory to test XCARD.

Emulated Card Deck Preparation

XCARD is available when SATK is installed. The object module booted by XCARD can be produced by using the `-o` output option of the SATK supplied assembler, ASMA. For details refer to the SATK ASMA documentation. The object module can be created using other assemblers, but must eventually reside on the SATK installed platform.

Combining the two output files into a single card deck uses the SATK `deck.py` utility. The utility is found at `tools/deck.py` within SATK. Refer to Appendix C of the IPLASMA documentation for details on the `deck.py` utility.

Other platform specific tools may be available to accomplish the same task.

An example of creating the emulated card deck using `deck.py` is below.

```
deck.py --card load.deck --boot xcard.deck module.deck
```

The emulated deck is the file `load.deck`. The two input files, the boot loader (`xcard.deck`) and object module (`module.deck`) are combined into the single output file emulating a card deck.

Use the `DECKS` environment variable directory search sequence to aid in locating the input files.

The supplied script `card` will create the virtual card medium used to test `XCARD`.

Virtual Tape (AWS) Preparation

While other tools may exist to create the emulated card deck, `deck.py` is likely the easiest (your mileage may of course vary) to create the AWS virtual tape file. Changing one option in the above example accomplishes the task:

```
deck.py --tape load.aws --boot xcard.deck module.deck
```

Instead of an emulated card deck, `--tape` produces the AWS virtual tape file. All virtual tape files must end with the `.aws` file extension.

A second option is to convert the emulated card deck directly into the virtual tape file:

```
deck.py --tape load.aws load.deck
```

This option assumes the emulated card deck already exists.

Use the `DECKS` environment variable directory search sequence to aid in locating the input files.

The supplied script `tape` will create the virtual AWS tape medium used to test `XCARD`.

Hercules Configuration and IPL

Whichever IPL medium is in use, the emulating file must be identified in the Hercules configuration file.

For an emulated card deck use:

```
000C 3505 load.deck eof ebcdic
```

For a virtual tape file use:

```
0120 3420 load.aws
```

Hercules supports Optical Media Attach (OMA) virtual tape files. Such files will **not** work because each individual file specified in the Tape Descriptor File (TDF) is treated as a separate file on the virtual tape. XCARD requires the XCARD loader and the object module to be in the same "file".

SDL Hyperion allows for the bypass of the `deck.py` utility for an emulated card deck. Multiple files may be specified in the Hercules configuration file. Hercules will automatically combine the multiple files into a single deck as presented to XCARD. Use this configuration statement for this purpose:

```
000C 3505 xcard.deck module.deck multifile eof ebcdic
```

To perform the IPL function with the device containing the IPL medium, issue manually or by means of a Hercules run command file an `ipl` command similar to these:

`ipl 00c` (for the card deck IPL)

or

`ipl 120` (for the AWS tape file IPL)

using the preceding configuration statement examples.

See the section "Testing XCARD" for details on the supplied Hercules related files used to test XCARD.

Memory Usage

This section describes how memory is used by XCARD. XCARD executes in two locations:

- Low memory starting at address X'0' loaded by the IPL function, and
- High memory into which XCARD relocates itself.

The XCARD processing that actually loads an object module occurs exclusively in the high memory locations. Additionally, during the loading of the object module, XCARD must make use of the first memory page when loading the individual object module cards (or records in the case of an AWS tape file). To allow the object module to initialize this same memory area while XCARD uses it, a surrogate page 0 is established by XCARD. Object module content destined for page 0 is placed in the surrogate page. The surrogate page is similar to a prefix page in concept but without the use of the SET PREFIX instruction. Once XCARD ceases use of page 0, it will replace its own contents of page 0 with that loaded into the surrogate page.

In the interests of reducing the number of cards/records loaded during the IPL process, XCARD uses dynamic areas without including the content in the IPL stream. This area is allocated at addresses higher than XCARD after it has "moved".

The followed two diagrams illustrate NCARD's use of memory, not drawn to any scale. Hexadecimal memory addresses are contained in the top line. Areas in green can be loaded by the booted object module. Areas in red can not be altered by the object module.

The 4096 bytes starting at area A is temporarily used to determine the size of available memory. Afterwards the entire area A is set to binary zeros.

Addresses A, B, X, Y, and Z are determined during XCARD execution. A, B, X and Y are always on page boundaries. Z starts when Y ends. Because XCARD uses memory dynamically, this description of memory usage is in terms of variable locations. However, in practical terms:

- A is at X'1000',
- B is at X'2000',
- X is at the end-of-memory less 8,191 bytes,
- Y is at the end-of-memory less 4,095 bytes, and

- Z immediately follows Y.

The green areas now identify the portions of memory loaded from the object module. Those in white have been set to binary zeros.

From the viewpoint of the booted object module, all addresses from X'0' through X-1 can be initialized by the module. Addresses X through EOM-6 are set to binary zeros before transferring control to the booted object module. The last six bytes are used to transfer control to the booted object module as described in the "Transfer Control" section.

XCARD has loaded the object module and then disappears from memory. The small exception is six bytes at the end of memory used by XCARD to transfer control to the booted object module.

On ESA/390 systems residing on a z/Architecture capable environment, reported memory size may exceed the ability of the ESA/390 XCARD program accessing memory with 31-bit address mode. In this case, end-of-memory is forced to be the end of two gigabytes of memory regardless of the memory configured for the environment.

Transferring Control

XCARD **never** changes the system architecture from that encountered when the IPL function is performed. If an architecture change is required by a booted object module, the module must perform the change itself.

The following sections describe the content upon entry to the booted object module of the:

- Program Status Word,
- Control Registers,
- General Registers,
- Main Storage (Memory),
- Storage Keys, and
- Input/Output System

In general, the **CPU** has the state as expected following resets for the following components:

- General Registers (Initial CPU Reset),
- Floating Point Registers (Initial CPU Reset) – not used by XCARD,
- Access Control Registers (Initial CPU Reset) – not used by XCARD,
- Control Registers (Initial CPU Reset),
- Vector Registers (Initial CPU Reset) – not used by XCARD, and
- Prefix Register (CPU Reset) – not used by XCARD.

Clock related registers are not altered by XCARD. No attempt is made by XCARD to create the conditions following a Subsystem Reset or storage changes made by a Clear Reset. Refer to the relevant *Principles of Operation* manual for reset details.

Program Status Word Content

Transferring control to the booted object module uses a LOAD PSW instruction in the last four bytes of available memory. The program status word made active by this instruction must be located at memory address X'0' as would an IPL PSW or a Restart New PSW. The booted program is expected to place into this address a PSW during the booting process. If the booted program fails to do so, an error condition occurs.

The PSW loaded from address X'0' dictates the operational state upon entry to the booted object module. This includes, but is not limited to, the PSW format, system mask and address mode. Refer to the relevant *Principles of Operation* manual for details on the PSW format and content.

Control Register Content

When control is transferred, any altered control registers have been restored to their initial values as dictated by the architecture in which the IPL occurs. Initial values are defined by the relevant *Principle of Operation* manual. All other Control Registers remain as initialized by the CPU reset occurring as part of the IPL function.

In systems not supporting a Channel Subsystem, XCARD resets Control Register 2 to binary zeros. Control Register 2 contains the channel interruption mask. All I/O interruptions from any channel is inhibited when this mask is reset to 0.

In systems supporting a Channel Subsystem, XCARD resets Control Register 6 to binary zeros. Control Register 6 contains the I/O interruption subclass mask. Any I/O interruption from the Channel Subsystem is inhibited when this mask is set to 0.

General Register Content

All General Registers are reset to their expected value following an Initial CPU Reset. All general registers contain binary zeros on entry to the booted object module.

Main Storage Content

Memory content is described in the previous section "Memory Usage." Memory content following XCARD is that specified by the booted object module. All locations not loaded by the object module contain binary zeros with the exception of the last 6 bytes of available memory.

Following the loading of the object module's content, the assigned storage areas for the IPL device's interruption information are passed to the object module. This will overlay any content loaded into these areas by the object module in the same manner as would occur during the IPL function.

The IPL device information is passed at these locations:

- X'B8' – IPL device Subsystem Identification Word,
- X'BC' – IPL device Interruption Parameter, and
- X'C0' – IPL device Interruption Identification.

Storage Key Content

The storage **key** is always zero for all memory pages.

Various storage key **access** settings are altered by the process of booting the object module. Storage Key access settings should be considered unpredictable upon entry to the booted object module.

If the object module requires certain values for the access setting in storage keys, it must establish those values itself.

Input/Output System State

The only mechanism that initializes the Input/Output system is the Subsystem Reset. XCARD can not initiate this reset. As a result, pending input/output interruptions may exist. The booted object module needs to understand that input/output interruptions from sources other than the one expected are possible depending on which portion of the Input/Output system is enabled.

Appendix A – XCARD Build Process

This section is intended as background for building of either XCARD or its test program when required. Building XCARD is normally not required. Only if a local change is being made is it required. In most cases `xcard.deck` can be directly used without change once available to the operating environment of choice on real or emulated mainframe systems.

The same can be said of the files that contain the XCARD and its test program: `load.deck` and `load.aws`. Either of these files can be moved to the operating environment of choice and used as supplied for validation of XCARD on the platform.

The XCARD build process starts as two separate processes that merge into one. The build process of XCARD and its test program both require SATK. They use macros and facilities specific to SATK. The following table illustrates the relationship of each step identified by the corresponding script available in the targeted architecture sub-directory.

The files that can be used directly on any supported platform are identified in **bold** font.

Create Emulated Decks		Emulated Decks	Create IPL Medium	Use IPL Medium
`asmx` - ASMA	`medx-iplasma.py`	**`xcard.deck`**	`card` – `deck.py` **`load.deck`**	`iplc`
`asmt` - ASMA		**`xcardtst.deck`**	`tape` – `deck.py` **`load.aws`**	`iplt`

Creating the Emulated Deck File

Building the XCARD Test Program

The XCARD test program is a object module booted using the XCARD loader. The object module is created assembling the program with an assembler that creates the a standard mainframe object module. The assembler must support instructions used by the architecture identified by the directory contained within the `xcard` directory, for example, `MVCLE`.

Use the script `xcard/s390/asmtst` to assemble the XCARD test program, `xcard/xcardtst.asm` source, into an object module, `xcard/s390/xcardtst.deck` using ASMA. Currently only `s390` is supported.

The output of this assembly may be input directly into the `card` or `tape` scripts for creation of the IPL medium.