MBT for MVS 3.8j Reference

Document Number ML02-0002-1

November 2026

Mike Großmann

First Edition (November 2026)

This edition applies to Version 3 Release 2 of MBT, the MVS build tool (mbt 3.2.0), and to all subsequent releases and modifications until otherwise indicated in new editions.

Comments on this book may be addressed to the issue tracker of the mvslovers/mbt repository on GitHub.

© Copyright Mike Großmann 2026. All rights reserved.

Contents

Figures . . . . . **v**

About This Book . . . . . **vii**
How This Book Is Organized vii
Related Publications vii

Chapter 1. The mbt Command **1**
mbt build 1
mbt check 1
mbt clean, mbt distclean 1
mbt compiledb 2
mbt deploy 2
mbt deps 2
mbt dist 2
mbt doctor 2
mbt migrate 3
mbt module-data 3
mbt package 3
mbt prerelease 3
mbt release 4
mbt run 4
mbt target 4
mbt test 5
mbt version 5
The Launcher 5
mbt ci-info 6

Chapter 2. mbt.toml **7**
Top Level 7
[project] 7
[toolchain] 7
[dependencies] 8
[build] 8
[build.host] 9
[lib] 9
[internal] 9
[module.NAME] 9
[module.NAME.smp] 10
Tests 11
[tests] 11
[test.NAME] 11
[tso.NAME] 12
[deploy] 12
[distribution] 12
[distribution.library.DIR] 12
[smp] 13
[release] 15
[tools] 15

[plugins] . . . 15
Reserved . . . 15

Chapter 3. targets.toml . . . 17
[target.NAME] . . . 17
[target.NAME.mvsmf] . . . 17
[target.NAME.mvsmf.users.ALIAS] . . . 17
[target.NAME.hercules] . . . 18
[target.NAME.tn3270] . . . 18
[target.NAME.ssh] . . . 18
[target.NAME.console] . . . 19
Passwords . . . 19
Selecting a Target . . . 19

Chapter 4. mbt.lock and deps.local.toml . . . 21
mbt.lock . . . 21
.mbt/deps.local.toml . . . 21

Chapter 5. The Lua API . . . 23
The Table mbt . . . 23
Tasks . . . 23
The Table ctx . . . 24
Plugins . . . 25

Chapter 6. The TSO Test API . . . 27
The Session . . . 27
The Screen Size . . . 28
Fixtures . . . 29
Logging Off . . . 29

Chapter 7. Environment Variables . . . 31
MBT Itself . . . 31
The Toolchain . . . 31
The Target . . . 32
Waiting for MVS . . . 32
MBT 2 Settings . . . 32

Chapter 8. Files and Directories . . . 35
In the Project . . . 35
In the Home Directory . . . 35

Chapter 9. The SMP Package . . . 37
Contents . . . 37
Data Sets . . . 37
The Volumes of the Allocation Job . . . 38
The Allocation Job, Submitted Again . . . 39
The Volume and the APF List . . . 39
DD Names . . . 40
Moving a Library to Another DD Name . . . 41
The Installation Job . . . 42
The Accept Job . . . 43
A Prerequisite That Cannot Be Accepted . . . 43
The SYSMOD . . . 44
A Module in a System Library . . . 44
Libraries as SMP Elements . . . 46
Prerelease Builds . . . 47
Replacing a Throwaway Installation . . . 48

Appendix A. Messages and Return Codes 51
Return Codes 51
Messages 51

Index 55

Figures

Tables

1. Setting the volume of a library with an APF entry 39
2. Some messages and what to do 51

About This Book

This book describes MBT (mbt), the MVS build tool, in full: the **`mbt`** command and its subcommands, the project file **`mbt.toml`**, the files MBT keeps for targets and dependencies, its Lua API, and its messages. How to use them together is the subject of the companion volume, the *MBT User's Guide*.

How This Book Is Organized

Chapter 1	"The mbt Command".
Chapter 2	"mbt.toml".
Chapter 3	"targets.toml".
Chapter 4	"mbt.lock and deps.local.toml".
Chapter 5	"The Lua API".
Chapter 6	"The TSO Test API".
Chapter 7	"Environment Variables".
Chapter 8	"Files and Directories".
Chapter 9	"The SMP Package".
Appendix A	"Messages and Return Codes".

Related Publications

ML02-0001	*MBT User's Guide*: building, testing, deploying, packaging and releasing an MVS project with MBT.
ML01-0001	*CC/370 User's Guide*: the cross-toolchain MBT drives.

Chapter 1. The mbt Command

```
mbt command [options] [arguments]
```

mbt is run in the directory of a project, the one that holds its **mbt.toml**, except **mbt target**, **mbt version** and **mbt help**, which need no project. **mbt** without a command lists the commands and ends with return code 2; **mbt help**, **-h** and **--help** list them with return code 0. Options may stand before or after the arguments, and **--** ends them. A long option is written with one or two hyphens. An unknown command, and an argument to a command that takes none, are return code 2.

When **mbt.toml** pins another version of MBT in **[toolchain] mbt**, the command is run by that version ("The Launcher" on page 5). The return codes are listed in Appendix A, "Messages and Return Codes" on page 51.

mbt build

```
mbt build [--all] [--tests] [-j n] [-v] [NAME...]
```

Compiles, assembles, archives and links. Without arguments it builds the main product: the library of a project of kind **library**, the load modules of any other.

--all	the load modules and the library.
--tests	also the test load modules.
-j *n*	run up to *n* steps at once. Default: the number of processors.
-v	print each command instead of a short line per step.
NAME	build only these load modules or tests, named in any case.

Before it links, **mbt build** runs the check of **mbt module-data**. It also checks that the installed LIBC/370 is not older than **[toolchain] libc370** (return code 1 otherwise), and that the declared dependencies are staged (return code 3). A *NAME* that is no module or test stops the build before any step runs (return code 2). **mbt deploy --module** checks its names only after the build and the logon.

mbt check

```
mbt check
```

Runs **mbt test** and, if it passed, **mbt test --mvs**.

mbt clean, mbt distclean

```
mbt clean
mbt distclean
```

clean removes **build/** and **dist/**. **distclean** also removes **.mbt/** with the staged dependencies, tools and plugins. **mbt.lock** is never removed.

mbt compiledb

```
mbt compiledb
```

Writes **compile_commands.json** in the project directory, one entry per C source of the modules, tests, **[lib]** and **[internal]**, for the language server **clangd**: **clang** commands with the options **clangd** needs to read the sources as CC/370 does (**-D__MVS__**, no host headers), **-std=gnu99**, **[build] cflags** and the include directories.

mbt deploy

```
mbt deploy [--target name] [--linklib dsn] [--module M]... [--dry-run] [-v]
           [--ignore-space] [--reallocate]
```

Builds the load modules, as **mbt build** does, and puts them into the development library on a target, replacing members of the same name. The library is never deleted unless **--reallocate** is given.

--target *name*	the target (“Selecting a Target” on page 19).
--linklib *dsn*	deploy into this library instead of **[deploy] target**.
--module *M*	deploy only this module. Repeatable.
--dry-run	pack the modules and report what would be done; no logon.
-v	print the commands and jobs.
--ignore-space	copy although the free space of the library looks too small.
--reallocate	delete the library and allocate it again on the same volume – room for several deploys, never smaller than before, and a directory for all members and aliases – then deploy. A probe job checks the volume has the room before the delete. Refused while the library is held.

A project without load modules has nothing to deploy and ends with return code 0. An unknown **--module** is return code 2, a failed build or pack 1, every failure on MVS 4.

mbt deps

```
mbt deps [--update]
```

Stages the dependencies, tools and plugins in **.mbt/**, at the versions **mbt.lock** pins; a dependency not yet in the lock is resolved and added. A locked version outside its range, or an archive whose checksum differs from the lock, is an error (return code 3).

--update	resolve every range again and rewrite **mbt.lock**.

mbt dist

```
mbt dist
```

Writes the SMP installation package again from the load TRANSMIT file already in **dist/**, without building. It refuses the same versions as **mbt package** (“mbt package” on page 3). A project without **[distribution]** has nothing to build and ends with return code 0.

mbt doctor

```
mbt doctor [--offline]
```

Checks that the commands **cc370**, **as370**, **ld370**, **ar370** and **xmit370** are found, the toolchain's directory tree with LIBC/370, their versions against **[toolchain]**, and the project file, and logs on to the target, read-only; the target is chosen as for every command, **MBT_TARGET** included. A project file that does not load, an unreachable mvsMF and a failed logon are errors and end the command with return code 2; a CC/370 older than **[toolchain] cc370** is a warning.

--offline check the workstation only: no connection, no logon.

mbt migrate

```
mbt migrate [--dry-run]
```

Reads **project.toml** of MBT 2, writes **mbt.toml**, and removes **project.toml** and, if there is one, **VERSION**. With an **mbt.toml** present it does nothing and ends with return code 0. It writes nothing when the new file would describe a different project than the old one: it says **nothing written** and ends with return code 99. Notes go to standard error.

--dry-run print the new file to standard output instead.

mbt module-data

```
mbt module-data [--all] [--raw]
```

Checks the sources of the load modules for writable data: definitions outside functions and **static** definitions inside them that are not **const**. Writable data in a **rent = true** module is an error, in an **ac = 1** module, or a module without a **rent** declaration (an MBT 2 project), a warning. Modules with **rent = false** and no **ac** are not checked. A test is checked only if it declares **rent = true** (an error) or **ac = 1** (a warning). Writable data in the **[internal]** sources is a warning too, as long as a module may be RENT. An error ends the command with return code 1; the report goes to standard error.

--all list every warning, not three per module. Errors are always listed in full.

--raw scan the sources as written, without running the preprocessor first.

mbt package

```
mbt package [-j n]
```

Builds the load modules and the library and writes the release files into **dist/**: *name*-*version* **-load.xmit**, **-lib.tar.gz**, and, with **[distribution]**, the installation package **-dist.zip** and **-dist.tar.gz** (Chapter 9, "The SMP Package" on page 37). A version from which no FMID can be derived stops **mbt** at the start with return code 2 ("[smp]" on page 13). Without an explicit **[smp] fmid**, a prerelease and a patch release are refused ("Prerelease Builds" on page 47), and so is a *x*.*y*.**0** when a patch of the previous minor was released and **[smp] delete** is not explicit; these end with return code 2 and no installation package, the other files are written.

mbt prerelease

```
mbt prerelease
```

Tags the current commit **v***version* with the version in **mbt.toml**, such as **v1.4.0-dev**, moving the tag if it exists, and pushes it. The working tree must be clean, and the project must own the repository's tags: its project file is at the root, or it is the only one in the repository. A version without **-dev** or **-rc**

n is refused. The release workflow that the tag starts runs **mbt package**, which refuses the prerelease of a product with an SMP package and no explicit **[smp] fmid** (“Prerelease Builds” on page 47).

mbt release

```
mbt release version [--next version]
```

Releases the tree’s *version***-dev** as *version*: sets it, commits **release: v***version*, tags, pushes, then sets the next development version, commits **chore: bump to** *next* and pushes. Refused, with return code 2, unless the tree is clean and at a prerelease of *version* (**-dev** or **-rc***n*), when the tag exists locally or on **origin** or **origin** cannot be asked (no such remote, no network, no access), for a patch release with an SMP package but no explicit **fmid**, and when the project does not own the repository’s tags (“mbt prerelease” on page 3), and, for a product with an SMP package, when the next version has a number above 9. A step that fails is not undone, and the command ends with return code 2.

--next *version* — the next development version, a prerelease (**-dev** or **-rc***n*). Default: the patch number plus one, with **-dev**.

mbt run

```
mbt run [name] [--target name] [-v] [--dry-run] [-- arguments...]
```

Runs a command or a task of the Lua of the project and of **~/.mbt/init.lua** (Chapter 5, “The Lua API” on page 23). Without *name*, lists them. A task runs even when it is up to date. The arguments after **--** reach a command as **ctx.args**; a task gets none.

--target *name* — the target a task uses (“Selecting a Target” on page 19).

-v — show each program run and its output.

--dry-run — run the Lua, but start no program and write nothing.

mbt target

```
mbt target list
mbt target ping [name] [--wait seconds] [--tso [-v]]
mbt target info [name] [--wait seconds]
mbt target console [name] -- command
mbt target import file --name name
```

Works on the targets of **~/.mbt/targets.toml** (Chapter 3, “targets.toml” on page 17). Without *name*, on the selected target. **mbt target** alone is **list**, which also shows the target **env** when **MBT_TARGET_*** defines one. **ping** ends with return code 4 only when mvsMF does not answer or, with **--tso**, the TSO logon fails. **--tso** on a target without **[tn3270]** ends with return code 2.

list — the targets, the default marked *****, with where each password comes from.

ping — asks each way of reaching the target whether it answers, without logging on unless **--tso** is given.

info — logs on to mvsMF and shows what runs there.

--wait *s* — wait up to *s* seconds for mvsMF to answer; with **--tso**, for the TSO logon too, within the same *s*. If mvsMF does not answer in time, return code 4.

--tso — after the other probes, log on to TSO over TN3270 as the **[tn3270]** user up to **READY**, log off and show how long the logon took. mvsMF answering does not mean TSO takes logons: HTTPD starts before VTAM and TCAS are done. With

	--wait, a refused connection or a logon that does not reach **READY** is tried again; a userid in use (**IKJ56425I**) or a rejected logon fails at once.
-v	with **--tso**, trace the 3270 data stream, as **mbt test --tso -v** does.
console	issues an operator command and shows the reply.
import	turns an MBT 2 **.env** file into the target *name*, filling what it leaves out from the environment and **~/.mbt/config.toml**.

mbt test

```
mbt test [--only NAME]... [-v]
mbt test --mvs [--only NAME]... [--no-deploy] [--target name] [--linklib dsn] [-v]
mbt test --tso [--only NAME]... [--target name] [-v]
```

Without **--mvs** or **--tso**, builds the tests with the workstation's C compiler and runs them there.

--mvs	build the test load modules, deploy them to the test library and run each in batch and under TSO on the target.
--tso	run **test/tso/*.lua** in TSO sessions over TN3270. Wins over **--mvs**; **--no-deploy** and **--linklib** are ignored.
--only *NAME*	only this test. Repeatable.
--no-deploy	with **--mvs**: use the test library already on MVS.
--target *name*	the target.
--linklib *dsn*	with **--mvs**: the library the tests run against, after the test library in the **STEPLIB**. Default: **[deploy] target**. If it does not exist on MVS, the tests run from the test library alone.
-v	on the workstation, the compile commands and the compiler's errors; with **--mvs**, the pack commands and the deploy's details; with **--tso**, a timed trace of the 3270 data stream on standard error.

A test, or with **--mvs** a step, that ends with 0 but printed no **PASS:** and no **FAIL:** line is shown as **no assertions** on the workstation and as **ok CC 0 (none)** on MVS, and is named in a warning:

```
[mbt] WARNING: 2 test step(s) passed without a single assertion: TSTGATE (batch), TSTGATE (tso)
[mbt] all test steps passed, 2 without assertions
```

The return code is not affected. With **--mvs**, a step part of whose output could not be read is not marked. **--tso** counts no assertions.

A test that fails ends the command with return code 1. On the workstation, a test with **host = false** or a source that is not C is skipped and named as **skipped (MVS-only)**. With **--mvs**, the return codes are those of Appendix A, "Messages and Return Codes" on page 51: 4 when mvsMF fails, 5 for a data set.

mbt version

```
mbt version
```

Prints the version of MBT, the commit it was built from and the date. **mbt --version** is the same.

The Launcher

Every **mbt** checks **[toolchain] mbt** of the project before it runs a command. **"3.0"** accepts any 3.0.*x*, **"3.0.2"** only that version; **"3.0"** also accepts prereleases of 3.0. When the running **mbt** does not match, the highest matching one under **~/.mbt/versions/** runs instead. Only when none is there is

one downloaded from the releases of **mvslovers/mbt**, a release preferred to a prerelease, and checked against their SHA-256 list. **MBT_NO_SWITCH=1** keeps the running one. A pin that is no version is return code 2, a version that cannot be fetched 3. **help** is never switched.

mbt ci-info

```
mbt ci-info
```

For workflows: prints **PROJECT_NAME**, **PROJECT_VERSION**, **PROJECT_FILE**, **CC370_REF**, **LIBC370_REF** and **MBT_PIN** from the project file, one **KEY=VALUE** per line. **CC370_REF** and **LIBC370_REF** are **v***version* for a version in **[toolchain]**, the entry itself for anything else (a branch, say), and **main** for none. **MBT_PIN** is empty when the project pins no version of MBT.

Chapter 2. mbt.toml

mbt.toml describes a project, in TOML, schema 3. This chapter lists every table and key, with its default. *NAME* below is the project name in upper case.

A key not listed here is an error that names it, with return code 2, in every table except **[dependencies]** and **[plugins]**, whose keys are repository names, and in **[tools]** the keys of an entry are checked the same way. So is a table in a form other than the one shown here: **[[distribution.library]]** where **[distribution.library.DIR]** is meant, **[[build]]** or **[[smp]]** for a table of its own, a key that is a string where a table belongs, a **[dependencies]** entry that is not a string. The message names the key and the form to write. Every command that reads **mbt.toml** stops with it, **mbt build** and **mbt deps** included; nothing is left out of a package without a word. The reserved tables of "Reserved" on page 15 are errors too. The types of most other values are not checked: give them as shown. A few keys are checked where they are used, for example **schema**, **version**, **max_rc**, **size** and the fixtures. A directory that holds both **mbt.toml** and an MBT 2 **project.toml** is an error, and so are the MBT 2 forms **[[module]]** and **[[test]]**.

Top Level

Key	Default	Meaning
schema	required	the number **3**, not the string.

[project]

Key	Default	Meaning
name	required	the project's name.
version	required	*major*.*minor*.*patch*, optionally **-dev** or **-rc***n*. No number above 9 in a project whose FMID is derived.
kind	**"application"**	**application**, **library** or **module**, not checked. Only **library** changes anything: **mbt build** then builds the library.

[toolchain]

Key	Default	Meaning
mbt	none	the MBT that runs the project: **"3.0"** for any 3.0.*x*, **"3.0.2"** or **"3.0.2-dev"** for that one ("The Launcher" on page 5).
cc370	none	the CC/370 a release is built with: a version, which means its tag, or any Git reference. Only **mbt doctor** compares it with the installed one, and warns.
libc370	none	the LIBC/370 a release is built with, a version or a Git reference. A version is also the oldest LIBC/370 a build accepts. A LIBC/370 before 2.3.0 has no C start-up in **libc.a**:

	MBT warns and links **crt0.o** and **crt1.o** instead. With 2.3.0 or later, CC/370 1.2.0 or later is needed.

[dependencies]

One key per dependency: "*owner*/*repo*" ="*range*". A range is one or more conditions joined by commas:

Condition	Allows
>=*v*	*v* and later.
<*v*	earlier than *v*.
=*v*	exactly *v*.
^*v*	*v* and later, below the next major version, or for **0.***x* below the next minor. *v* may have one, two or three numbers; **^0** means below 1.0.0.

Versions are ordered by their numbers, then **-dev** < **-rc***n* < the release. A prerelease is chosen only for a range that names one.

[build]

Key	Default	Meaning
include	**[]**	directories for **-I**, after **include/**, which is on the path when the project has one, unless the list names it.
cflags	**[]**	options for **cc370**, after MBT's **-O1 -std=gnu99 -Wall -Wextra -Werror**.
asflags	**[]**	options for **as370**.
max_rc	**4**	the highest return code of **as370** that still passes, an integer from 0 to 16. A higher one fails the step and removes its object module. An assembly that passes with a return code above 0 is shown as a warning with **as370**'s messages, and the build closes with their count. **0** fails any warning.
cobflags	**[]**	options for **cobc370**, for every COBOL source: **-I** for copybook directories, **-s**. The **include/** of every dependency follows, so **COPY** finds the copybooks a dependency ships.
pliflags	**[]**	options for **pl1c370**, for every PL/I source. The **include/** of every dependency follows.

A source ending in **.cbl** or **.cob** is COBOL. **cobc370** translates it into **build/cobol/***stem***.asm** with **-MMD**, so a changed copybook translates it again, and **as370** assembles that into **build/***stem***.o**. **cobc370** is found in **MBT_COBC370**, else a **[tools]** entry named **cobc370**, else on **PATH**. A unit with COBOL sources needs **entry**, the PROGRAM-ID that starts it, unless one of its C sources defines **main()** at the start of a line; then the default **@@CRT0** stays. It cannot be **rent = true**: **cobc370**'s code keeps WORKING-STORAGE in the module.

A source ending in **.pli** or **.pl1** is PL/I. **pl1c370** translates it into **build/pli/***stem***.asm** with **-MD**, so a changed **%INCLUDE** member translates it again, and **as370** assembles that into **build/***stem***.o**. **pl1c370**

is found in **MBT_PL1C370**, else beside **cc370** in the **bin** directory of the sysroot, else on **PATH**. If it is nowhere, the build stops with a configuration error.

The compile line is MBT's options, the **-I** options, **cflags**, then **-I** for the dependencies, MBT's headers and **.mbt**. Two sources with the same file name stem in one project are an error, as both would compile to the same object module.

[build.host]

For **mbt test** on the workstation only. The host compile line is **-std=gnu99 -Wall -Wextra -Werror**, the options of **[build] cflags**, the **-I** options for the dependencies, **include/** and **.mbt**, then **[build.host] cflags**.

Key	Default	Meaning
cc	**CC**, else **cc**	the C compiler of the workstation.
cflags	**[]**	options after **[build] cflags**.
sources	**[]**	further sources linked into every test.
replace	**{}**	**{ "***source***" = "***other***" }**: use *other* instead of *source*.

[lib]

Key	Default	Meaning
name	the project name	the library: **build/***name***.a**.
sources	none	its sources. Patterns such as **"src/*.c"** are allowed here and wherever sources are listed; a pattern that matches nothing is a warning. Without sources, only the headers are published.
headers	**[]**	the headers published with it.

[internal]

Key	Default	Meaning
sources	none	sources of a private archive every module and test links with.
exclude	**[]**	files the patterns match that are left out.

[module.NAME]

One table per load module. *NAME* is the member name in upper case, one to eight characters, **A**-**Z**, **0**-**9**, **@**, **#**, **$**, not starting with a digit; quoted when it holds **#**, **@** or **$**. Aliases follow the same rule and must be unique across all modules and aliases; a test has none.

Key	Default	Meaning

sources	required	the module's own sources. Not checked: a module without any fails at the link.
exclude	**[]**	files the patterns match that are left out.
rent	required	reentrant.
reus	required	serially reusable.
refr	**false**	refreshable.
entry	**"@@CRT0"**	the entry point.
startup	a C program	**false**: no C run time; **"crtm"**: a C module run by another C program; it needs **crtm.o**, which newer LIBC/370 releases no longer ship. Any other value, including **"crt0"** and **"crt1"**, is an error.
dep_startup	none	**true**: **@@START** comes from a dependency; **false**: from LIBC/370. If a dependency defines **@@START**, every module that has a C start-up must set this key, else the build stops with return code 2.
ac	**0**	the authorization code.
aliases	**[]**	further names of the module.
ship	**true**	**false**: the module is built and deployed, but **mbt package** and **mbt dist** leave it out of the load TRANSMIT file and the SYSMOD, for a test tool that must not reach the product library.
smp	none	a table: the module goes into a system library of the SMP procedures, not into the product's target ("[module.NAME.smp]").

[module.NAME.smp]

A module that belongs in a library of the system, such as an SVC routine in **SYS1.LPALIB**, is linked there by SMP and travels as its assembler source, assembled at installation by the target system's assembler ("A Module in a System Library" on page 44):

```
[module.IGX00011]
entry   = "IGX00011"
startup = false
sources = ["asm/igx00011.asm"]
rent    = true
reus    = true
refr    = true

[module.IGX00011.smp]
syslib  = "LPALIB"
distlib = "AOST4"
source  = true
```

Key	Default	Meaning
syslib	required	the DD name, in the SMP procedures, of the system library the module is linked into: **LPALIB**, **LINKLIB** or **SVCLIB**. The data set is **SYS1.** and the DD name.
distlib	required	the DD name of the distribution library SMP keeps the module in: one of load modules of the MVS/CE procedures, **AOS***xx*, **ALPALIB** or **ACMDLIB**, as **SYS1.** and the DD name.

source	required	**true**: the module is shipped as its source. It is the only form supported: a load module linked from the staging library into a system library has not been measured.
parm	derived	the **PARM** of the link-edit step, without quotes. Derived: **MAP,LIST,NCAL**, then **RENT**, **REUS**, **REFR** as the module declares them, and **AC=***n* for **ac**.

The module has exactly one source, a **.asm** file, and no aliases. At most one module of a project has this table, and at least one module goes into the product's target as before. Other keys and other values are errors (return code 2). **mbt build** and **mbt deploy** treat the module like any other. A **project.toml** (mbt 2) with **smp** on a **[[module]]** is refused: migrate it first.

Tests

Every **test/**/*.c** and **test/**/*.asm** is a test, named after its file in upper case, with that file as its only source, unless it is in **[tests] exclude** or the **sources** of a **[test.***NAME***]**. A file name that is no member name is an error that names the file: exclude it or give it a **[test.***NAME***]**.

[tests]

Key	Default	Meaning
rent, **reus**	required once there is a test	for every test without its own.
refr	**false**	as for a module.
exclude	**[]**	paths under **test/** that are not tests, each written in full: **"test/helper.c"**. No pattern. Hidden files and directories are never tests.

[test.NAME]

Key	Default	Meaning
sources	the found file	the test's sources.
host	**true**	**false**: not on the workstation.
mvs	**true**	**false**: not on MVS.
parm	none	the program argument: a string for both MVS steps of **mbt test --mvs** (batch and TSO), or **{ batch = ..., tso = ... }**. A host test gets none. Without it the test gets none; one that then skips its work passes with no assertions. At most 49 characters for the batch step and 69 for the TSO step. A plain string goes to both steps, so it is limited to 49; only the form **{ batch = ..., tso = ... }** can use the 69 of the TSO step.
fixtures	none	**{** *DD* **= [***files***] }**: members loaded into a data set per DD, *HLQ*.*PROJECT*.**FIX.***TEST*.*DD*, FB 80. A member is the file's name without its extension, in upper case; its lines are the records, columns 73 to 80 included, nothing is numbered. A line longer than 80 columns is an error (return code 2). Lines starting **//** or **/***, JCL included, arrive as data. A

	member name is cut to eight characters; of two files with the same member name, the first counts. A file that has lines starting with each of **$A** to **$Z** is an error.

The keys of a load module, **exclude**, **entry**, **startup**, **dep_startup**, **ac**, **rent**, **reus** and **refr**, are allowed too. A test that must run APF-authorized needs **ac = 1** as well as APF-authorized libraries: without it **__isauth()** returns 0, even from a library on the APF list.

[tso.NAME]

The data sets the interactive test **test/tso/***name***.lua** needs (“Fixtures” on page 29), and its screen size. *NAME* is the file's name in any case, one to eight characters, a letter first; a table that names no such file is an error when **mbt test --tso** runs.

Key	Default	Meaning
fixtures	none	**{** *KEY* **= [***files***] }**: a data set per *KEY*, the last qualifier of its name, loaded as the fixtures of **[test.NAME]** are.
size	the target's	the terminal for this test, *COLS***x***ROWS* such as **"80x43"** (“The Screen Size” on page 28). It wins over **size** in the target's **[tn3270]**.

[deploy]

Key	Default	Meaning
target	*NAME* **.DEV.LINKLIB**	the development library.
test_target	*HLQ***.***NAME***.***VRM* **.TESTLIB**	the test library. The name is made upper case, characters other than **A**-**Z**, **0**-**9**, **@**, **#** and **$** are dropped, and it is cut to eight characters.

target is derived only for a project with load modules. A default **target** that would not be a valid data set name (a qualifier of more than eight characters, a character a data set name does not allow, or a digit first) is refused, with the key to set. The same holds for the defaults of **[smp]** and **[distribution.library]**.

[distribution]

Key	Default	Meaning
readme	none	shipped as **README.md**. The file must exist.
extra	**[]**	further files shipped under their base names; a missing file or a directory is an error.

[distribution.library.DIR]

A text library made from the directory *DIR* and received with TSO **RECEIVE**.

Key	Default	Meaning

target	*NAME*.*DIR*, the last part of *DIR* in upper case	the library it is received into.
element	none	**"MAC"** or **"SRC"**: SMP installs the library member by member, every member an element of that type, instead of the library being received as a plain data set ("Libraries as SMP Elements" on page 46).
distlib	**A** before the last qualifier of **target**	the distribution library SMP keeps the elements in. Only with **element**; without it the key is ignored.
stage	**T** before the last qualifier of **target**	the library the shipped members are received into and SMP reads them from. Only with **element**; without it the key is an error.
target_dd	the last qualifier of **target**	the DD name of the target library in the SYSMOD and the jobs ("DD Names" on page 40). Only with **element**; without it the key is an error.
distlib_dd	the last qualifier of **distlib**	the DD name of the distribution library. Only with **element**; without it the key is an error.

With **element** a member is a file of the directory, the member name its base name in upper case. The records are fixed 80 (**FB**, **LRECL=80**, **BLKSIZE=3120**); there is no key for another format. All 80 columns reach the library as they are – columns 73 to 80 may hold sequence numbers, and mbt neither renumbers nor strips them. **mbt package** refuses a member with a line past column 80 (tabs are expanded to every eighth column and trailing blanks do not count, as **xmit370** does), a control character, or a character without a CP037 equivalent, and names the file, line and column. A file is read as UTF-8 when it is valid UTF-8, else as Latin-1.

An element name is one for the whole SYSMOD: the same member name in two libraries of the same **element** type is an error, the same name as **MAC** in one library and **SRC** in another is not. The **target**, **distlib** and **stage** of the libraries with **element** must differ from each other and from **lklib**, **target** and **distlib** of **[smp]**. A library without **element** is not compared.

[smp]

Needed with **[distribution]**, and only with it; a **[distribution]** needs at least one load module to ship, and a module with **ship = false** is not one, nor is one with **[module.***NAME***.smp]**. *P4* is the first four characters of *NAME*.

Key	Default	Meaning
prefix	required unless **fmid**	**T** and three letters, by convention. Not checked itself, but *prefix* + major + minor + **0** must be a valid FMID of seven characters.
fmid	*prefix* + major + minor + **0**	the FMID, seven characters, a letter and six letters or digits. Required for a version whose patch number is not 0, and for a prerelease ("Prerelease Builds" on page 47).
delete	with a derived **fmid**: *prefix* + major + (minor – 1) + **0**; with an	the FMIDs this release replaces. Required with an explicit **fmid**, and with a derived one for *x*.**0.0**; **[]** for a first level. It must not name the release's own FMID. Each id has seven characters, a letter first.

	explicit one: none	
system	**"Z038"**	the SREL of the **++VER**.
prereq	**[]**	SYSMODs that must be applied before this one. Each id has seven characters, a letter first. The **ACCEPT** of the accept job then carries **BYPASS(REQ)** ("A Prerequisite That Cannot Be Accepted" on page 43).
accept_fmid	**true**	write the accept job, with the **ACCEPT** of the FMID and the **CLEANUP**. The installation job ends after the **APPLY** ("The Accept Job" on page 43). With **false** there is no accept job, and the **CLEANUP** stays behind the **APPLY**.
procedure	none	**"mvsce"** or **"tk5"**: the system the package installs on. The two **SMPAPP** procedures list different DDs (MVS/CE has **ASAMPLIB**, TK5 has not), and the install job must name an overridden DD in the procedure's order. Needed only if the order is not the same for both, and then mbt refuses to guess.
lklib	*NAME*.*P4*LOAD	the staging library the load modules are received into. Not the same as **target**.
target	*NAME*.LINKLIB	the target library.
distlib	*NAME*.A*P4*LOD	the distribution library.
lklib_dd	the last qualifier of **lklib**	the DD name of the staging library ("DD Names" on page 40).
target_dd	the last qualifier of **target**	the DD name of the target library.
distlib_dd	the last qualifier of **distlib**	the DD name of the distribution library.
[smp.predecessor]	none	a table of *DDNAME* = **"data.set.name"**: the libraries the level **delete** removes was installed through. For the one release that moves a library to another DD name ("Moving a Library to Another DD Name" on page 41). Each name is a DD name; each value is a full data set name (no **@VRM@**). A DD that the new level also uses must name the same data set. Needs a non-empty **delete**.
target_volume	**"MVSRES"**	the volume of the target libraries: the default of **TGTVOL** in the allocation job ("The Volumes of the Allocation Job" on page 38). One to six characters, **A**-**Z**, **0**-**9**, **@**, **#** and **$** lower case is made upper case.
dlib_volume	**"SMP000"**	the volume of the distribution libraries: the default of **DLBVOL**. The same characters.

A DD name is one to eight characters. A name that SMP or the procedures use for their own files is refused.

The defaults are the volumes of an MVS/CE system. TK5 is deliberately not the default: on TK5, set **TGTVOL=TK5RES** and **DLBVOL=TK5DLB** in the allocation job.

For a release version, **mbt package** refuses a version whose patch number is not 0 without an explicit **fmid**, and a derived **delete** for *x*.*y*.**0** when the repository has a release tag *x*.*y*–*1*.*p* with $p > 0$. It refuses

a **-dev** or **-rc***n* version without an explicit **fmid**, and writes no installation package (“Prerelease Builds” on page 47). The tag check needs the repository’s tags, which a shallow clone does not have.

[release]

Key	Default	Meaning
version_files	**[]**	further files in which **mbt release** replaces the version. A missing file, or one without the version, is an error.

When **[distribution]** is present, **mbt release** also refuses a patch release without an explicit **fmid**, and a next version with a component above 9 (give **--next**).

[tools]

One key per tool: *name* **= { ... }**.

Key	Default	Meaning
repo	required	*owner*/*repo* on GitHub.
version	required	the release **v***version*.
asset	**{name}-{os}-{arch}.tar.gz**	the release asset for this platform.
bin	**{name}-{os}-{arch}**	the member of a **.tar.gz** or **.zip** asset that is the program.

In **asset** and **bin**, **{name}**, **{version}**, **{os}** (**darwin**, **linux**, **windows**) and **{arch}** (**amd64**, **arm64**) are replaced.

[plugins]

One key per plugin: **"***owner*/*repo*" = "*range*"**, with the ranges of **[dependencies]**.

Reserved

[rule."*.*ext*"], **[files."***glob***"]**, **[lang.*]** and **[usermod.*]** are reserved for later versions.

Chapter 3. targets.toml

~/.mbt/targets.toml, or **targets.toml** in the directory **MBT_HOME** names, describes the MVS systems of a workstation, one table **[target.***name***]** each. A key not listed here is an error.

[target.NAME]

Key	Default	Meaning
default	**false**	the target used without a name. At most one.
hlq	the mvsMF user in upper case	the high-level qualifier of the data sets MBT allocates.
volume	none	the volume for the libraries MBT allocates and receives. Without it, the system chooses.
jobclass	**"A"**	the class of the jobs MBT submits.
msgclass	**"H"**	their output class.

[target.NAME.mvsmf]

Required.

Key	Default	Meaning
url	required	**http://***host***:***port*, the port 80 if left out. Only **http** is supported; another scheme fails at the connection.
user	required	the user ID.
password	none	a password ("Passwords" on page 19).
auth	**"session"**	how MBT logs on. **"session"** logs on once (**POST /zosmf/services/authenticate**) and sends the session's token with every request, and logs off at the end. **"basic"** opens no session: the user and password go with every request, and a **GET /zosmf/info** checks them first. For an mvsMF whose token logon does not work, such as an mvsMF 1.0.x on an HTTPD before 4.2.0, whose first request with the token abends.

[target.NAME.mvsmf.users.ALIAS]

Further mvsMF users, for Lua only: **ctx.mvs.session** and **ctx.mvs.request** with **auth = "basic"** act as one of them by its alias ("The Table ctx" on page 24). The alias is lower case letters, digits, **-** and **_**, and starts with a letter or digit. MBT itself (deploy, tests) always uses the user of **[target.NAME.mvsmf]**, and Lua never sees a password.

Key	Default	Meaning
user	required	the user ID.
password	none	its password (“Passwords” on page 19).

```
[target.lab.mvsmf.users.test2]
user     = "TEST2"
password = { keychain = "mbt/lab-test2" }
```

[target.NAME.hercules]

The web interface of Hercules, for operator commands when mvsMF does not take them.

Key	Default	Meaning
url	required	its URL.
user	none	a user, if it asks for one.
password	none	its password.

[target.NAME.tn3270]

For **mbt test --tso**.

Key	Default	Meaning
host	required	the host.
port	**3270**	the port.
tls	**false**	not supported yet; **tls = true** is rejected when the file is loaded, because connections are plain TCP and the TSO logon would go out in clear. **tls = false** is accepted.
user	none	the TSO user ID the tests log on with.
password	none	its password.
size	**"80x24"**	the terminal’s screen, *COLS*x*ROWS*: **"80x43"** for a model 4, **"160x62"** for an oversize screen (“The Screen Size” on page 28). A test’s **[tso.NAME] size** wins.

[target.NAME.ssh]

Key	Default	Meaning
host	required	the host that runs Hercules. **mbt target ping** tries a TCP connection to this host and port; MBT starts no SSH session.
user	none	the user, which Lua sees in **ctx.target**.
identity	none	a key file. MBT does not use it.
port	**22**	the port.

[target.NAME.console]

Key	Default	Meaning
order	**["mvsmf"]**, and **["mvsmf", "hercules"]** when the target has **[hercules]**	the channels for operator commands, in the order they are tried. A channel is tried only when the one before certainly did not deliver. Naming **hercules** without its section, or an unknown channel, is an error.

Passwords

Value	The password is
"*text*"	*text*. The file must not give group or others any permission (**chmod 600**) if any password in it is plain text. MBT checks this when it reads the file.
{ env = "*VAR***" }**	the value of the environment variable *VAR*.
{ keychain = "*name***" }**	the entry with the service name *name*, on macOS from the Keychain (**security find-generic-password -s**), elsewhere from the Secret Service (**secret-tool lookup service**).
{ cmd = ["*prog***", ...] }**	the first line *prog* writes, started without a shell.

A table names exactly one source, and **cmd** a non-empty list. An **env**, **keychain** or **cmd** source that yields an empty password, or an unset variable, is an error that names it. The password is read only when MBT logs on, so a dry run never asks for it.

A key not listed in this chapter is an error, reported by its table. The types of **tls**, **size**, **users**, **password** and **order** are checked. For the other keys a value of the wrong type counts as not given: give them as shown.

Selecting a Target

A command that needs MVS takes the first of:

1. **--target** *name*, else the variable **MBT_TARGET**: the target of that name. If there is none, an error lists the defined ones, or says that the file defines none. The name **env** selects the target from the environment; a target called **env** in the file is never chosen by name.
2. The target from the environment (“The Target” on page 32), when **MBT_TARGET_MVSMF_URL** is set.
3. The target with **default = true**.
4. The only target in the file.

Two or more targets and none the default is an error. Only when no name is given and the file is missing or defines no target does MBT use the MBT 2 settings, **MBT_MVS_*** and **.env**, with a warning.

Chapter 4. mbt.lock and deps.local.toml

mbt.lock

mbt.lock is a JSON object in the project directory, written by **mbt deps** only when it changes, with its keys sorted. It is committed. A project that never had dependencies, tools or plugins has none. MBT does not remove one, also when the last dependency is dropped.

```
{
  "mvslovers/crypto370": {
    "sha256": "f5686240265baf69773e887bb7abd2c3f8da84d77bd377df38f95f4aa879fb6f",
    "version": "1.0.1"
  }
}
```

Key	Entry
owner/*repo*	a dependency: the version chosen and the SHA-256 checksum of *repo*-*version*-**lib.tar.gz**.
tool:*name*@*os*-*arch*	a tool of **[tools]**, one entry for the platform MBT runs on, and one for each other of **darwin-arm64**, **darwin-amd64**, **linux-amd64**, **linux-arm64** and **windows-amd64** for which GitHub publishes a checksum of the asset. The checksum is of the asset, not of the program in it.
plugin:*owner*/*repo*	a plugin: the version and the checksum of *repo*-*version* **-plugin.tar.gz**.

Each entry has the two members **version** and **sha256**. **mbt deps** writes the entries of tools and plugins. A tool's entry is also written when Lua first asks for it (**ctx.tool**), or when a build with COBOL sources fetches a **cobc370** from **[tools]**. **mbt deps** fails when a locked dependency or plugin version is outside its range, or when a download has another checksum than the lock. A tool pinned at another version than **[tools]** names is pinned again. **mbt deps --update** rewrites the entries. An entry **tool:***name* without a platform, written by an earlier MBT, is replaced.

.mbt/deps.local.toml

.mbt/deps.local.toml is not committed. Its table **[override]** points dependencies and plugins at working copies:

```
[override]
"mvslovers/crypto370" = { path = "../crypto370" }
```

path is a directory with an **mbt.toml** or **project.toml** that has a **[lib]**. **mbt deps** copies its **build/***name* **.a** to **.mbt/deps/***repo***/lib/** and the files of its **[lib] headers** to **.mbt/deps/***repo***/include/**, and fails with **not built -- run 'mbt build' in** *path* when the archive is missing. **[lib] name** defaults to the project name; headers are copied without their directories, and a missing one is skipped. **mbt deps** and **mbt deps --update** leave the dependency's entry in **mbt.lock** as it is. A file that is not valid TOML, or an entry without **path**, is ignored without a message. For a plugin, **path** is the plugin's directory, with **plugin.toml** and **init.lua**.

Chapter 5. The Lua API

MBT runs Lua 5.5 from **mbt/init.lua** of an **mbt.toml** project, from **~/.mbt/init.lua**, and from staged plugins, in this order: plugins, the project, the user. **require("***name***")** loads **mbt/lua/***name***.lua** (or **~/.mbt/lua/**), **require("***owner***/***repo***")** a plugin. **dofile**, **loadfile** and **package** are not available, nor are **os**, **io** and **debug**. **print** writes to MBT's output. Each call into Lua runs under a limit of computation steps and of 256 MiB of memory; there is no limit of wall-clock time, and programs started with **ctx.exec** are not limited. The interface has a version, **mbt.api**; this chapter describes version 1.

The Table mbt

mbt.hook(*point***, *fn*)**	runs *fn(ctx)* at *point*: **before_** or **after_** followed by **build**, **test**, **package**, **dist**, **deploy** or **release**, or **on_failure**.
mbt.task{…}	declares a task ("Tasks").
mbt.command(*name***, *fn*)**	makes *fn(ctx)* the command **mbt run** *name*.
mbt.command{…}	the same as a table: **name**, **run**, and a **description** that **mbt run** lists beside the name. Only this form carries one.
mbt.api	the interface version, **1**.
mbt.version	the version of MBT.

mbt.hook with an unknown point is an error that lists the points; a command needs a name and a function, and a name may be used once. **require** takes names in lower case, dots for directories (**a.b** is **mbt/lua/a/b.lua**).

Errors. An error in a **before_** hook or in a task stops the command before it runs. An error in an **after_** hook fails the command and undoes nothing. An **on_failure** hook gets **ctx.result** with **command** and **error**; its own errors are only warnings.

Tasks

```
mbt.task {
  name        = "webroot",
  description = "the web root image",
  before      = { "package", "dist" },
  inputs      = { "static/**" },
  outputs     = { "build/webroot/httpd-webroot.img" },
  run         = function(ctx) ... end,
}
```

name	required; **mbt run** *name* runs it.
before	the commands it runs ahead of. Without it, only **mbt run** runs it.
inputs	files, directories and patterns (*****, **?**, ******).
outputs	the files it writes.
run	the function.

A task's **description** is shown by **mbt run**. Errors in a task declaration: **before** naming no command of the list above, a path outside the project, a name used twice or by a command. Under **--dry-run** a task runs, but its outputs and its record of the last run are left in place, also when it fails.

Tasks run before the **before_** hooks of a command, each at most once per run. A task with inputs but no outputs, or an input pattern that matches nothing, is an error. A task runs on the first run and, without outputs, every time. Otherwise it is skipped while its outputs are newer than every input and nothing that made them changed: the task itself, the Lua files loaded, the **[tools]**. Its outputs are removed before it runs and after it fails.

The Table ctx

Member	Meaning
ctx.project	a copy: **name**, **version**, **modules**, **tests**, **deploy_target**: **[deploy] target**; in an **mbt.toml** with load modules and no target, *NAME*.**DEV.LINKLIB**; **nil** when neither applies.
ctx.args	the arguments after **--** of **mbt run**.
ctx.dry_run	**true** under **mbt run --dry-run** and **mbt deploy --dry-run**.
ctx.kind	in **before_test** and **after_test**: **host**, **mvs** or **tso**.
ctx.result	in **after_** hooks: what the command produced – **build**: **modules**, **tests**; **package**, **dist**: **artifacts**; **release**: **version**, **tag**, **prerelease**; **test**: **kind**, **passed**; **deploy**: **library**, **modules**, **dry_run**. In **on_failure**: **command** and **error**.
ctx.out, **ctx.inputs**	in a task: its outputs, and the files its inputs matched.
ctx.log(*s***)**, **ctx.warn(***s***)**	a line of output, or a warning.
ctx.exec{*argv…* **, env = {}, check = true}**	runs a program without a shell. **check** defaults to **true**: a non-zero exit code raises an error; with **check = false** it is returned. Returns the output and the exit code; the output is standard output and standard error together. Under **mbt run** the program uses the terminal and the output returned is empty. The program runs in the project directory. **env** adds environment variables. Under **--dry-run** nothing runs. In a plugin, only the programs its **plugin.toml** names, by the base name of the first argument.
ctx.tool(*name***)**	the path of a **[tools]** program, fetched and checked if need be. Under **--dry-run**, only the path: nothing is fetched and **mbt.lock** is not written.
ctx.fs.read(*p***)**, **write(***p, s***)**, **exists**, **mkdir**, **remove**, **list**	files, inside the project only: an absolute path, **..** or a link leading outside is an error. **list(***pattern***)** returns files and directories, **exists** a boolean. **remove** does not remove the project itself. Under **--dry-run**, **write**, **mkdir** and **remove** do nothing.
ctx.target()	the selected target, defaults applied, never a password: **name**, **hlq**, **volume**, **jobclass**, **msgclass**, **console**, **mvsmf = {url, user}** (with **users = {***alias* **= user}** where the target has further users), and where the target has them **hercules = {url, user}**, **ssh = {host, user, port}**, **tn3270 = {host, port, tls,**. An error when the command has no target. **user}**

ctx.console(*cmd***)**	an operator command through the target's console chain. Returns the reply lines and the channel that delivered it; under **--dry-run**, an empty list.
ctx.mvs.request{ method, path, body, content_type }	a request to mvsMF, in MBT's session unless **auth** says otherwise. **method** defaults to **GET** **path** starts with **/** and lies below **/zosmf** **content_type** defaults to JSON when there is a body. Returns the HTTP status as the server sent it (**201**, **204**, **401** ...), the body, and the response headers: a table keyed by the name in lower case, several values joined by **,** , those of **set-cookie** by a newline. An error status comes with the body the server sent. Only a failed connection raises an error.
auth = "basic"	HTTP Basic with the stored password of the main user, or of **user = "***alias***"** ("[target.NAME.mvsmf.users.ALIAS]" on page 17). **password = "..."** sends a literal of the caller's own instead, for a test of a wrong one.
auth = "none"	no credentials at all.
auth = "token"	**token = "..."** as the session cookie, and no logon if it is refused.
ctx.mvs.token()	the session token, for another program's environment; under **--dry-run**, an empty string. A program that gets it must never log it off: the session is MBT's.
ctx.mvs.session{ user }	a session of its own, logged on as the main user or as **user = "***alias***"**: every logon is a session of its own. **s:request{ method, path, body,** as above; **s:token()**; **content_type }** **s:logout()** returns the status of the logoff (**204**). After a 401 it does not log on again, so a test sees it. A session still open is logged off when the command ends.

Dry run. Under **--dry-run**, **ctx.exec** runs nothing and **ctx.mvs.request** sends only **GET** requests; anything else answers **0**, **""** and no headers. **ctx.tool** fetches nothing and pins nothing, and **ctx.mvs.token** does not log on: both serve a program that a dry run does not start. **ctx.mvs.session** logs on to nothing: its token is empty, and its requests and its logoff answer **0**.

Plugins

A plugin is the release asset *repo*-*version*-**plugin.tar.gz** of its repository, holding:

plugin.toml	**api = 1**, the interface version it is written for, and **exec = [***programs***]**, the programs it may start.
init.lua	run when the plugin is loaded; it returns the plugin's module.
lua/	further modules: **require("***owner***/***repo***/***mod***")**.

A plugin without **api**, without **init.lua**, or for an interface version MBT does not offer is a configuration error.

Chapter 6. The TSO Test API

`mbt test --tso` runs each file **`test/tso/*.lua`** in a TSO session of its own over TN3270, as the user of the target's **`[tn3270]`**; a target without one is an error (return code 2). The test is named after its file in upper case; **`--only`** *NAME*, repeatable, in any case, runs a subset. The hooks and tasks of **`before_test`** run first, those of **`after_test`** afterwards; the hooks of both see **`ctx.kind "tso"`**.

The file must return a function, which is called with the session *t*. The test passes when the function returns without an error. The Lua environment is that of Chapter 5, "The Lua API" on page 23.

```
return function(t)
  t:logon()
  t:type("TIME"):enter()
  t:expect("TIME-")
  t:logoff()
end
```

The Session

Member	Meaning
`t:logon()`	logs on with the target's TN3270 user, in upper case, and password, and waits for **`READY`**, at most two minutes. When the user's logon procedure starts ISPF, it leaves the ISPF primary option menu (**`=X`**, then PF3) to reach **`READY`**. When the terminal answers its first Enter with **`INPUT NOT RECOGNIZED`**, it types the user there. A logon that fails after the password logs off again, so the user is not left in use.
`t:logoff()`	presses Clear, types **`LOGOFF`**, presses Enter and waits up to 30 seconds for **`LOGGED OFF`**, then up to three seconds for the logon screen VTAM writes back, so that the connection is not closed under it (which leaves an I/O error on the terminal). It leaves the ISPF primary option menu first (**`=X`**, then PF3); it does not back out of a deeper full-screen application.
`t:type(`*s***`)`**	types *s* at the cursor. An error, with the screen, when the keyboard is locked or the cursor is in a protected field.
`t:enter()`, **`t:clear()`**	press Enter, Clear.
`t:pf(`*n***`)`**, **`t:pa(`***n***`)`**	press PF1 to PF24, PA1 to PA3. Another number is an error.
`t:tab()`	moves the cursor to the next input field; sends nothing.
`t:expect(`*text*** **`[, { timeout =`** ***s*** **`}])`**	waits until *text*, case-sensitive, is on the screen, or was on it since the last key that went to the host. *s* is whole seconds, default 10. Otherwise an error with the lines of the screen.
`t:screen()`	the screen: one line per row, trailing blanks removed – 24 lines, or the rows of the alternate size once the host has switched to it ("The Screen Size" on page 28). Fields that are not displayed, such as a password, are blank.

t:text()	the same as one string, lines joined by newlines.
t:cursor()	the cursor's row and column, counted from 1.
t:color(*row***, *col***)**	the colour a 3279 shows at *row*, *col* (from 1): **"blue"**, **"red"**, **"pink"**, **"green"**, **"turquoise"**, **"yellow"** or **"white"**. The colour the host gave the character (Set Attribute) or its field (Start Field Extended, Modify Field), else the default of the field: protected blue, protected and intensified white, unprotected green, unprotected and intensified red. On a screen without fields the colour is green. A blank cell has a colour too, although it shows none. A position off the screen is an error.
t.user	the TN3270 user.
t.testlib	the name of the test library of **mbt test . --mvs**
t.fixtures	the data sets of **[tso.NAME] fixtures**: *KEY* **=** *name* ("Fixtures" on page 29). Empty for a test without any.
t.size	the terminal's alternate screen size, *COLS***x***ROWS*, such as **"80x24"** ("The Screen Size").

All methods except **screen**, **text**, **cursor** and **color** return the session, so calls can be chained: **t:type("TIME"):enter()**. Enter, Clear, PF and PA wait for the keyboard to be unlocked, at most 60 seconds, and then for 300 milliseconds without output. With a table **{ wait = false }** as the last argument – **t:enter{ wait = false }**, **t:pa(1, { wait = false })** – they send the key and return at once. That is for a program that keeps the keyboard locked while it runs: an attention test presses Enter on its **CALL** without waiting, waits with **t:expect** for the program's prompt, and then presses PA1, which reaches TSO while the program still runs.

The Screen Size

The terminal is a 3278 model 2, 24 rows of 80 columns, unless a size is given: **size** in the target's **[tn3270]** ("[target.NAME.tn3270]" on page 18) or, for one test, in **[tso.NAME]** ("[tso.NAME]" on page 12), which wins. A size is written *COLS***x***ROWS*, columns first, as x3270's **-oversize** takes it:

Size	The terminal
80x24	model 2 (the default), **IBM-3278-2-E**.
80x32	model 3, **IBM-3278-3-E**.
80x43	model 4, **IBM-3278-4-E**.
132x27	model 5, **IBM-3278-5-E**.
any other	an oversize screen of at least 80x24 and at most 16383 positions, such as **160x62**. It goes as the largest model it covers in both directions (**160x62**: model 5), as with x3270; its size travels in the reply to the host's Read Partition Query.

The size is the terminal's alternate screen. Every model starts on the default screen of 24x80; the host switches to the alternate one with Erase/Write Alternate and back with Erase/Write, and the Clear key returns to 24x80. **t:screen()** always shows the screen in use. TSO line mode, and the logon, stay on 24x80; a full-screen application, such as ISPF, decides itself whether it uses the alternate size.

Fixtures

A test that works on data sets of its own – an exec, a CLIST, a panel, input – declares them in **mbt.toml**:

```
[tso.FSSMENU.fixtures]
TSOTEST = ["test/tso/fssmenu.rexx"]
RXLIB   = ["rxlib/FSSMENU.rexx", "rxlib/FSSAPI.rexx"]
```

The table form, not an inline table across several lines: TOML 1.0 keeps an inline table on one line, and a tool of the project's own that reads **mbt.toml** with a TOML 1.0 parser (Python's **tomllib**) refuses the other.

Before the sessions, and after the **before_test** hooks, MBT deletes and allocates one PDS per key, *HLQ* **.***PROJECT***.FIX.***TEST***.***KEY*, FB 80, and loads the files into it with one job, **MBTTSOFX**, whose number it prints. Member names and the load are those of the **mbt test --mvs** fixtures ("[test.NAME]" on page 11): each line becomes a record, columns 73 to 80 included, nothing is numbered. A member that did not load fails the command with return code 5 before any session. Only the tests selected with **--only** are loaded. The test allocates what it needs itself:

```
t:type("ALLOC F(RXLIB) DA('" .. t.fixtures.RXLIB .. "') SHR"):enter()
```

Logging Off

When the function ends, after a successful **logon** and without a **logoff** of its own, MBT logs the session off, whether the test passed or failed. When that fails, the test fails, even if it had passed, and MBT says so beside the test's own error: the user is most likely still logged on, and every later logon fails until it is cancelled (operator command **C U=***user*). The tests after it are therefore not run and are shown as **not run**.

A TSO user can be logged on only once. The tests run one after the other, and the target's TN3270 user should not be one you are logged on with yourself. **mbt test --tso** does not deploy the test modules; run **mbt test --mvs** first when a session calls one. **-v** traces the 3270 data stream, never a password. Each line starts with the seconds since the connection was made, so the trace also shows how long the host took for a record; the logon notes there when it waits for the host on a first screen without a logon prompt, and what came of it.

Chapter 7. Environment Variables

MBT reads these environment variables. None is needed for an ordinary build on a workstation with a **`targets.toml`**.

MBT Itself

Variable	Meaning
`MBT_HOME`	the directory used instead of **`~/.mbt`** for **`targets.toml`** (also where **`mbt target import`** writes it), **`versions/`**, **`init.lua`** and **`lua/`**. The download caches and MBT 2′s **`config.toml`** stay in **`~/.mbt`**.
`MBT_NO_SWITCH`	**`1`**: run the started **`mbt`** even when the project pins another version.
`MBT_VERSION_SWITCHED`	set by MBT for the pinned **`mbt`** it starts, to that version. Any value stops MBT from switching again. Do not set it yourself; use **`MBT_NO_SWITCH`**.
`GITHUB_TOKEN`	a token for the GitHub API, which lifts the limit of 60 requests an hour. For dependencies, **`MBT_GITHUB_TOKEN`** is read when **`GITHUB_TOKEN`** is not set.
`SOURCE_DATE_EPOCH`	seconds since 1970: the time stamp of the files in the archives **`mbt package`** and **`mbt dist`** write, for reproducible archives.

The Toolchain

Variable	Meaning
`MBT_CC`, **`MBT_AS`**, **`MBT_LD`**, **`MBT_AR`**	the commands used instead of **`cc370`**, **`as370`**, **`ld370`** and **`ar370`** by **`build`**, **`test --mvs`**, **`package`**, **`deploy`**, **`module-data`** and **`doctor`**. **`mbt doctor`** also looks for **`cc370`**, **`as370`**, **`ld370`**, **`ar370`** and **`xmit370`** by these names on **`PATH`**.
`MBT_COBC370`	the **`cobc370`** for COBOL sources, a path or a name on **`PATH`**. Ahead of a **`[tools]`** entry and of **`PATH`**.
`MBT_PL1C370`	the **`pl1c370`** for PL/I sources, a path or a name on **`PATH`**. Ahead of the **`bin`** directory that holds **`cc370`** and of **`PATH`**.
`CC`	the C compiler of the workstation for **`mbt test`**, unless **`[build.host] cc`** names one. Default **`cc`**.
`ASMDATE`, **`ASMTIME`**	read by **`as370`**: the date and time written into object modules.
`LDDATE`, **`LDTIME`**	read by **`ld370`**: the date and time written into load modules.

`ASMDATE` and the others belong to CC/370; set them to make two builds byte-identical (*MBT User's Guide*, "Migrating from MBT 2").

The Target

MBT reads and checks **targets.toml** first, also when the environment target is used; a broken file stops MBT even when **MBT_TARGET_MVSMF_URL** is set.

Variable	Meaning
MBT_TARGET	the name of the target, as **--target**.
MBT_TARGET_MVSMF_URL	defines a target from the environment, named **env**, with this mvsMF URL. It is chosen ahead of the default of **targets.toml** unless **--target** or **MBT_TARGET** names another.
MBT_TARGET_MVSMF_USER, **MBT_TARGET_MVSMF_PASSWORD**	its user and password. The password variable must be set and not empty when MBT logs on.
MBT_TARGET_MVSMF_AUTH	**basic** for a logon without a session, as **auth** in **[target.NAME.mvsmf]** ("[target.NAME.mvsmf]" on page 17).
MBT_TARGET_MVSMF_USERS	further users by alias, comma separated (**test2,ro-user**), as **[target.NAME.mvsmf.users.ALIAS]**. Each one's user and password are **MBT_TARGET_MVSMF_USER_***ALIAS* and **MBT_TARGET_MVSMF_PASSWORD_***ALIAS*, the alias in upper case with **-** as **_**.
MBT_TARGET_HLQ, **MBT_TARGET_VOLUME**	**hlq** and **volume**.
MBT_TARGET_JOBCLASS, **MBT_TARGET_MSGCLASS**	**jobclass** and **msgclass**.
MBT_TARGET_HERCULES_URL, **_USER**, **_PASSWORD**	the Hercules web interface; with the URL it joins the console chain.
MBT_TARGET_TN3270_HOST, **_PORT**, **_USER**, **_PASSWORD**, **_SIZE**	the TN3270 access, port 3270 by default; **_SIZE** is the screen, *COLS***x***ROWS* ("The Screen Size" on page 28).

Waiting for MVS

Variable	Meaning
MBT_DEPLOY_TIMEOUT	seconds to wait for the deploy job. Default: 300, plus 60 for each whole megabyte of the TRANSMIT file. It also covers the receive of the test library.
MBT_TEST_TIMEOUT	seconds to wait for the job of **mbt test --mvs**. Default: 10 per step, fixture loads included, at least 120.

MBT 2 Settings

Without a **targets.toml** that defines a target, and without **MBT_TARGET_MVSMF_URL**, MBT uses the settings of MBT 2, with a warning: **MBT_MVS_HOST**, **MBT_MVS_PORT**, **MBT_MVS_USER**, **MBT_MVS_PASS**, **MBT_MVS_HLQ**, **MBT_MVS_DEPS_VOLUME**, **MBT_JES_JOBCLASS** and **MBT_JES_MSGCLASS**, from the environment, a **.env** file of the project, or **~/.mbt/config.toml**. **MBT_MVS_DEPS_HLQ** and **MBT_BUILD_ID** are no longer read. **mbt target import** turns them into a target.

The order is the environment, the **`.env`**, **`~/.mbt/config.toml`** (tables **`[mvs]`** and **`[jes]`**, keys **`host`**, **`port`**, **`user`**, **`pass`**, **`hlq`**, **`deps_volume`**, **`jobclass`**, **`msgclass`**), then the defaults: host **`localhost`**, port **`1080`**, user and **`hlq`** **`IBMUSER`**, classes **`A`** and **`H`**. No volume.

Chapter 8. Files and Directories

In the Project

Path	Commit	Contents
`mbt.toml`	yes	the project file (Chapter 2, “mbt.toml” on page 7).
`mbt.lock`	yes	the pinned dependencies, tools and plugins (Chapter 4, “mbt.lock and deps.local.toml” on page 21).
`project.toml`, **`VERSION`**, **`Makefile`**	yes	MBT 2′s files. **`project.toml`** is read when there is no **`mbt.toml`**. **`mbt migrate`** reads all three, writes **`mbt.toml`**, and removes **`project.toml`** and **`VERSION`**.
the files of **`[release] version_files`**	yes	**`mbt release`** writes the new version into them and into **`mbt.toml`**.
`include/`	yes	public headers; on the include path.
`test/`	yes	the tests: every **`.c`** and **`.asm`** file below it, named by its stem in upper case, unless **`[tests] exclude`** lists it; **`test/tso/`***`name`***`.lua`** the interactive ones.
`mbt/init.lua`, **`mbt/lua/`**	yes	the project’s Lua.
`build/`	no	object modules (***`stem`***`.o`, ***`stem`***`.d`), archives, load modules (***`NAME`***, ***`NAME`***`.iebcopy`), **`host/`** with the workstation’s test programs, **`cobol/`** and **`pli/`** (the generated assembler and its **`.d`** files), **`dist-stage/`**, the TRANSMIT files ***`project`***`.deploy.xmit` and ***`project`*** `.test.xmit`, the generated jobs and spool files: **`test-runner.jcl`**, **`test-runner.spool`**, **`testlib-receive.spool`**, **`deploy.spool`**.
`dist/`	no	the release files of **`mbt package`**, and the SMP package of **`mbt package`** and **`mbt dist`**.
`.mbt/`	no	MBT’s state: **`deps/`**, **`tools/`**, **`plugins/`**, **`include/`** (**`mbtcheck.h`**), **`buildstamp.h`**, **`cmd/`** (the command line of each step and the signature of each task), **`deps.local.toml`**, **`libc370-checked`** (the last libc370 check), **`migrate-check.toml`** and **`migrate-failed.toml`** (the files of **`mbt migrate`**’s check). **`mbt distclean`** removes the whole directory, **`deps.local.toml`** too.
`compile_commands.json`	no	written by **`mbt compiledb`**.

In the Home Directory

Path	Contents

~/.mbt/targets.toml	the targets (Chapter 3, “targets.toml” on page 17). **MBT_HOME** moves it. **mbt target import** creates or extends it, readable by you only.
~/.mbt/init.lua, **lua/**	your own Lua. **MBT_HOME** moves them.
~/.mbt/versions/	the versions of MBT that projects pinned. **MBT_HOME** moves them.
~/.mbt/v3/cache/	downloaded archives by checksum: dependencies directly below it, tools in **tools/**, plugins in **plugins/**. Always under the home directory.
~/.mbt/config.toml	MBT 2′s settings, read for the fallback when there is no target.

A project’s **.env** is read for the same fallback.

Chapter 9. The SMP Package

With **[distribution]** and **[smp]** (“[smp]” on page 13), **mbt package** writes an installation package. *n* is the project name, *v* the version, *NAME* the name in upper case, *P4* its first four characters.

Contents

n-*v*-**dist.zip** and **-dist.tar.gz** hold one directory *n*-*v*/ with:

File	Contents
README.md	**[distribution] readme**.
n-*v*-**load.xmit**	the load modules: a TSO TRANSMIT file of a load library. A module shipped as source (“A Module in a System Library” on page 44) is not in it.
n-*v*-*q*.**xmit**	one per **[distribution.library.***dir***]**: a TRANSMIT file of a text library, *q* the DD name of its target (its last qualifier, “DD Names” on page 40) in lower case.
n-*v*-**alloc.jcl**	the allocation job.
n-*v*-**inst.jcl**	the installation job, with the SYSMOD inline, and the source of a module shipped as source. It ends after the **APPLY**.
n-*v*-**accept.jcl**	the accept job: **GUARD**, **ACCEPT** and **CLEANUP** (“The Accept Job” on page 43). Not written with **accept_fmid = false**.
extra files	under their own names.

Data Sets

Data set	Default	Role
staging	*NAME*.*P4***LOAD** (**lklib**)	the load modules are received into it; the **LKLIB** of the **++MOD**s. Deleted by the accept job, or by hand (“The Accept Job” on page 43).
target	*NAME*.**LINKLIB** (**target**)	the installed modules.
distribution	*NAME*.**A***P4***LOD** (**distlib**)	the accepted modules.
text library	*NAME*.*DIR*	received with TSO **RECEIVE**, outside SMP.
element library	*NAME*.*DIR* (**target**), **A** before its last qualifier (**distlib**), **T** before it (**stage**)	a library with **element**: the members are received into the staging library, SMP installs each one into the target and distribution libraries (“Libraries as SMP Elements” on page 46).

The DD name of each data set in the jobs and the SYSMOD is its last qualifier, unless a key gives it another (“DD Names” on page 40). No two of them may share one, and none may be **SMPTLIB**. Data set names are at most 44 characters. **@VRM@** in a name is replaced by the version qualifier.

Text libraries are made from the regular files directly in *DIR*, not from subdirectories or files whose name begins with a period. Each file is a member and needs a valid member name; an empty directory is an error. In their text, **@LINKLIB@**, **@VRM@**, **@VERSION@**, **@FMID@** and **@***dd***@** (the DD name of each library's target, “DD Names” on page 40) are replaced by the values of the release.

The allocation job allocates the target and distribution libraries, **RECFM=U**, **BLKSIZE=15040**, in cylinders, and never deletes them. A first step, **CHECK**, looks each of them up in the catalog and ends with return code 8 if one is cataloged: the allocation step does not run then (“The Allocation Job, Submitted Again” on page 39). The target and distribution library of an element library are allocated by it as well, fixed 80, **BLKSIZE=3120**, sized for the first version twice over, with a directory of at least 20 blocks. Every one of them is allocated on a volume of the job's choosing (“The Volumes of the Allocation Job”).

The Volumes of the Allocation Job

The allocation job is an in-stream procedure, **SMPALLOC**, called by one step. Its two symbolic parameters name the volumes, and each library is allocated with **UNIT=SYSDA,VOL=SER=&TGTVOL** or **UNIT=SYSDA,VOL=SER=&DLBVOL**. No library of the package is allocated without a volume.

Symbolic	Default	Used for
TGTVOL	**MVSRES**	every target library: the one the load modules are installed into, and those of an element library.
DLBVOL	**SMP000**	every distribution library.

The defaults come from **target_volume** and **dlib_volume** of **[smp]** (“[smp]” on page 13), and are the volumes of an MVS/CE system. The operator changes them in the job, either on the **PROC** statement or on the **EXEC** statement:

```
//ALLOC   EXEC SMPALLOC,TGTVOL=TK5RES,DLBVOL=TK5DLB
```

On TK5, set TGTVOL=TK5RES and DLBVOL=TK5DLB. The defaults are those of MVS/CE, and a job that keeps them on TK5 is held (see the last row of the table below).

What the system does was measured on MVS/CE and on TK5, each in a container of its own, with a job that allocates four libraries (two target, two distribution):

Case	Result
MVS/CE, defaults	Return code 0. **IEF285I** shows **VOL SER NOS= MVSRES** for the two target libraries and **SMP000** for the two distribution libraries. The VTOCs of the two volumes list each library once, on that volume, and not on the other.
TK5, **TGTVOL=TK5RES**, **DLBVOL=TK5DLB**	Return code 0, the same check with **TK5RES** and **TK5DLB**.
A volume the system does not have (MVS/CE: **TGTVOL=NOVOL1** TK5: the defaults)	The job does not end, and no return code comes. It is held while the system allocates the data sets of the step, and the console shows **IEF244I ... UNABLE TO ALLOCATE 1 UNIT(S)** (**2 UNIT(S)** on TK5 with the defaults, where both volumes are missing), **IEF489I ... 1 UNIT(S) NEEDED FOR LINKLIB** and ***00 IEF238D ... REPLY DEVICE NAME OR 'CANCEL'**. Reply **R 00,CANCEL**: the job ends with **IEF251I ... JOB CANCELLED**, the step is not executed, and no

	library of the job remains (a library already allocated on the other volume is deleted, **`IEF285I ... DELETED`**).

The held job is what a wrong volume looks like, so the operator who sees an allocation job that does not end looks at the console first. Submit the job again with the volumes corrected.

The job has no step that deletes a library (see above), so it does not move one that exists to another volume.

The Allocation Job, Submitted Again

The **`CHECK`** step is IDCAMS with one **`LISTCAT ENTRIES(`***`dsn`***`)`** for each library the job allocates, each followed by **`IF LASTCC = 0 THEN SET MAXCC = 8`**. A name that is not in the catalog ends the **`LISTCAT`** with 4, which the last statement of the step turns into 0; a name that is cataloged, on any volume, makes the step end with 8. The allocation step carries **`COND=(0,NE,CHECK)`** and is not run behind it. The job ends with return code 8, and the listing shows the libraries found. Nothing is allocated, and no **`IEF287I ... NOT CATLGD 2`** is left behind.

The catalog is asked, not the volume, so the check holds when the second run names other volumes than the first: that is the case in which the allocation alone ended with return code 0 and an uncataloged duplicate. The check works the same on MVS/CE and on TK5 (measured with the libraries of a BREXX/370 3.0.0 package: first run **`CHECK`** 0 and **`ALLOC`** 0; second run on the same volumes **`CHECK`** 8, **`ALLOC`** not run, job 8). With **`TGTVOL`** changed for the second run it was measured on TK5: the same result, and no **`IEF287I`**. To allocate again, delete the libraries first, or take other names.

Free space, measured on a fresh MVS/CE container, before the job: **`MVSRES`** 276 cylinders, **`SMP000`** 158 cylinders and 26 tracks. A target or distribution library of the load modules takes 5 cylinders as its primary space (secondary 2), so the job above takes 10 cylinders of each volume. The load library of a package of the size of BREXX/370 3.0.0 needs 7 to 8 cylinders on each side, which fits.

The Volume and the APF List

An entry in **`SYS1.PARMLIB(IEAAPF00)`** of MVS 3.8j is **`dsname volser`**: it names a data set together with its volume, and authorizes it there. A library of the same name on another volume is not authorized. The allocation job does not read the list, does not compare the volume with it and does not say if they differ: the volume of a library is whatever **`TGTVOL`** (or the volume written into its **`DD`**) says. Where the library is APF-listed, the volume of the entry decides:

Table 1. Setting the volume of a library with an APF entry

Where	Effect
`target_volume` in **`[smp]`**	The default of **`TGTVOL`**, written into the **`PROC`** statement when **`mbt package`** or **`mbt dist`** renders the job. Every package of the project gets it.
`PROC` statement of the job	Changes the default for this run of the job, if the package is already written.
`EXEC SMPALLOC,TGTVOL=`*`volser`*	The same, on the **`EXEC`** statement.
`VOL=SER=`*`volser`* in a **`DD`**	For one library, when the target libraries of the product have entries on different volumes. If no **`DD`** uses **`TGTVOL`** then, it leaves the **`PROC`** statement as well (see below).

The list is read at IPL. A library that has no entry yet gets one that names the volume **`TGTVOL`** puts it on, and counts as authorized after the next IPL.

A symbol that no DD uses. Taking a **DD** statement out of the job, for a library that exists, can leave **TGTVOL** or **DLBVOL** on the **PROC** statement with nothing that uses it. MVS 3.8j does not ignore the symbol: the job does not run, with **IEF452I JOB NOT RUN - JCL ERROR** and **IEF657I SYMBOL NOT DEFINED IN PROCEDURE** (measured on MVS/CE with **DLBVOL**, and with **TGTVOL** left behind a job of one distribution library). The symbol goes from the **PROC** statement with its **DD** (*MBT User's Guide*, "Upgrading an Installation").

DD Names

SMP addresses a library by DD name, and stores that name with every element it installs. Left alone, mbt uses the last qualifier of the data set name, and for *NAME*.**LINKLIB** or *NAME*.**SAMPLIB** that is **LINKLIB** or **SAMPLIB**: the DD names of **SYS1.LINKLIB** and **SYS1.SAMPLIB** in the SMP procedures. The installation job overrides the procedure's DD with the product's library, so the installation works; but the element is recorded under **LINKLIB**, and a later SYSMOD or restore run with the standard procedure resolves that name to the system library and writes there.

Each library SMP installs into can have a DD name of its own:

Key	Library
[smp] lklib_dd, **target_dd**, **distlib_dd**	the staging, target and distribution library of the load modules.
[distribution.library.*DIR***] target_dd**, **distlib_dd**	the target and distribution library of an element library ("[distribution.library.DIR]" on page 12). A text library is not touched by SMP and has no DD name to set.

The name is 1 to 8 characters, **A**-**Z**, **0**-**9**, **@**, **#** and **$**, not starting with a digit; lower case is made upper case. It is used wherever the last qualifier was: the **DISTLIB**, **SYSLIB** and **LKLIB** operands of the **++MOD**, **++MAC** and **++SRC** statements, the **COPYLOAD** step of the JCLIN, the DD cards of the APPLY CHECK, APPLY and ACCEPT steps, the **INDSN** of each **RECEIVE** (its data set name ends in the DD name), the allocation job, the name of the TRANSMIT file of an element library (*q* in "Contents" on page 37), and the **@***dd***@** placeholder of the text in a library. A library without a key keeps its last qualifier.

mbt package checks the names against the DDs of the SMPAPP procedures of MVS/CE and TK5 (measured with HMASMP level 04.48), the DDs SMP reads by name (**SMP...**) and those JCL gives a meaning (**JOBLIB**, **STEPLIB**, **JOBCAT**, **STEPCAT**, **SYS...**):

- a name given with a key that is one of them is an error. The library would replace the system library of that name in the step, and SMP would store the name with every element.
- the last qualifier of a load library or an element library, when no key was given, that is one of them is a warning (**[mbt] WARNING:**) and the package is built as before. This is the case for the default target library *NAME*.**LINKLIB**. The default distribution library of an element library does it as well when its target is, for example, **MACLIB** or **SAMPLIB**: **AMACLIB** and **ASAMPLIB** are distribution libraries of MVS/CE. **SMPTLIB** is an error either way.

On purpose: a module in a system library. The one place a procedure's DD name is right is a module that belongs in the system library itself: **[module.***NAME***.smp] syslib** and **distlib** ("[module.NAME.smp]" on page 10) name **LPALIB** and **AOST4**, for example, and the module is recorded under them, as IBM's own modules are ("A Module in a System Library" on page 44). They are checked against a list of their own, and a product library may still not use them.

Changing the DD name of a library that is already installed does not move the elements SMP has recorded: they stay under the old name. Choose the names before the first install. A release that has to move a library anyway carries the old name for its **DELETE** ("Moving a Library to Another DD Name" on page 41).

Moving a Library to Another DD Name

The **++VER ... DELETE(***old***)** of a release asks SMP to delete the elements of the level it replaces. SMP finds them by the DD name recorded with each element, and resolves that name in the step of the job that runs the **APPLY**, not in the data set the element was written to. If the new level's library has another DD name than the old one, the job needs a card for the old name, or the delete looks in the wrong place:

- The old name is not defined in the step: **APPLY CHECK** ends with return code 12 and **HMA2832** *dd* **DDCARD MISSING**, or the **APPLY** passes and the **ACCEPT** fails with **HMA2833** and **HMA3020 ... DELETE PROCESSING FAILURE**.
- The old name is a DD of the SMP procedures, in practice **LINKLIB**: nothing fails. The procedure resolves **LINKLIB** to **SYS1.LINKLIB**, **HMA2240** reports the delete as successful, and it removes a module of that name from the system library. The old product library keeps its modules.

[smp.predecessor] gives the job those cards. One line for each library whose DD name changes, written as the card it becomes. The old library is a data set of its own; the new level gets another one:

```
[smp]
prefix    = "TABC"
target    = "ABC.ABCLINK"
target_dd = "ABCLINK"

[smp.predecessor]
LINKLIB = "ABC.LINKLIB"
```

The jobs then carry **//HMASMP.LINKLIB DD DISP=SHR,DSN=ABC.LINKLIB** in the DD block of the **APPLYCHK**, **APPLY** and **ACCEPT** steps (the last is in the accept job), in the order the SMPAPP procedure requires: a DD of the procedure, such as **LINKLIB**, comes before every other card, or the procedure's own wins without a message. The cards of the new level follow it. A library whose DD name and data set stay the same needs no line; a line for it is accepted and the card is written once. List the target library and the distribution library of the load modules, and the target and distribution library of each element library, whenever the old level used another DD name for it.

The same data set name under another DD name is supported. A new level that installs into **ABC.LINKLIB** through **ABCLINK**, while the table names **LINKLIB = "ABC.LINKLIB"**, gets both cards, for one data set. Measured on TK5 with the cards in the job: SMP runs the delete of the predecessor before it copies the new modules, so a module the new level writes stays, and a module the new level no longer has is deleted. The card for the old name is required here too. Without it the delete does not reach the library: for **LINKLIB** it removes a module from **SYS1.LINKLIB** and leaves the old copy in the product library, and a module the new level no longer has stays behind as an orphan.

When the new level gets a data set of its own, as in the example, that library does not exist when the upgrade starts; the guide says how to allocate it (*MBT User's Guide*, "Upgrading an Installation").

mbt package checks the table:

- the DD name is valid, as for **target_dd**, and not one SMP or JCL reads by name (**SMP...**, **SYS...**, **...PRINT**, **JOBLIB**, **STEPLIB**, **JOBCAT**, **STEPCAT**). A DD of the procedures that is a library, **LINKLIB** for one, is allowed here: it is the case the table is for.
- the data set name is valid and written out. **@VRM@** stands for the version of the level being packaged, not of its predecessor, and is refused.
- **delete** is not empty. Without a deleted level nothing reads the cards.
- a DD name that the new level uses as well must name the same data set. The card is then written once. Another data set is refused: one DD name cannot name two data sets in a step, and the delete would land in the new library.
- a TXLIB card of an element never takes a name of the table.

Only the release that moves the library carries the table. The release after it deletes a level that was already installed through the new DD name, and the table is removed again.

The Installation Job

Step	Action
DELOLD	IDCAMS: deletes the staging library and every text library; **SET MAXCC=0**.
RECV*n*	IKJEFT01: **RECEIVE INDSN(...)** for each TRANSMIT file. The names to edit are **DATASET(...)** marked **CHANGE.ME**. Each **RECEIVE** also carries a commented **VOLUME('CHANGE.ME')** line: harmless as shipped (TK5), to be uncommented and given a volume where **RECEIVE** needs one (usually MVS/CE, **PUB000** on a stock system).
GENER1	only with a module shipped as source ("A Module in a System Library" on page 44): IEBGENER writes the SYSMOD, inline in the job (**DD DATA,DLM=@@**), to the temporary data set **&&MCS**, if the last **RECV***n* ended with 0.
ASSEM	only with a module shipped as source: IFOX00 assembles its source, inline in the job, against **SYS1.MACLIB** and **SYS1.AMODGEN**, and appends the object deck to **&&MCS** (**SYSGO DISP=(MOD,PASS)**), if **GENER1** ended with 0.
RECV	SMP **RESETRC** and **RECEIVE SELECT(***fmid***)** of the SYSMOD, inline in the job (**DD DATA,DLM=@@**), if the last **RECV***n* ended with 0. With a module shipped as source the SYSMOD is read from **&&MCS**, and the step also needs **GENER1** and **ASSEM** to have ended with 0.
APPLYCHK	**APPLY CHECK**, if the last **RECV***n*, **GENER1** and **ASSEM** (where the job has them) and **RECV** ended with 0.
APPLY	**APPLY**, only after a clean check: if the same steps and **APPLYCHK** ended with 0.
CLEANUP	only with **accept_fmid = false**: IDCAMS deletes the staging library, if the same steps, **APPLYCHK** and the **APPLY** ended with 0 (with 4 for the **APPLYCHK** and the **APPLY** of a package with a library of **SRC** elements, and for the **APPLY** of a package with a module in a system library).

JCL does not test a **COND** against a step that did not run. So each SMP step names every SMP step before it, not only the one before, and also the steps that can bypass the **RECV** (the last **RECV***n*, with a module shipped as source also **GENER1** and **ASSEM**): behind a **RECV** that ended with 8, the **APPLYCHK** is bypassed, and an **APPLY** that named only the **APPLYCHK** would run without the check; behind an **ASSEM** that ended with 8, the **RECV** is bypassed, and an **APPLYCHK** that named only the **RECV** would go on to apply a SYSMOD that an earlier run had received. The earlier **RECV***n* are not named (JCL takes eight tests on a **COND** at most): **DELOLD** scratches every staging library first, so after a failed **RECEIVE** the **APPLYCHK** has no data set to allocate under that library's DD, and the job ends there.

With **accept_fmid** (the default) the job has no **ACCEPT** and no **CLEANUP**. It ends after the **APPLY**, and its closing comment says what to do next. The staging libraries stay, because SMP reads the elements from them again at the **ACCEPT**. The comment also holds the **DELETE** statements of the staging libraries, as comment cards, for a site that takes the function out with a **RESTORE** or never accepts it.

Return codes. A clean run ends with return code 0, also with **prereq**. The **APPLY CHECK** and the **APPLY** stay strict with **prereq**: a prerequisite that is not applied ends them with return code 12 and **HMA3590 ---** followed by the identifier and **REQ** ("A Prerequisite That Cannot Be Accepted" on page 43), and the job ends with 12. With **delete** the **APPLY** ends with 4 (**HMA2461**, "The SYSMOD" on page 44), and with a module in a system library as well ("A Module in a System Library" on page 44).

The Accept Job

The **ACCEPT** is the point after which a **RESTORE** cannot take the function out again, so it is not a step of the installation job but a job of its own, which the user submits after trying the product. The job needs no editing: the staging libraries have fixed names, the same as in the installation job.

Step	Action
GUARD	SMP: **LIST CDS SYSMOD(***fmid***)**. Ends with 0 while the SYSMOD is in the target zone, and with 4 when it is not.
NOTAPPL	IDCAMS, with **COND=(4,NE,GUARD.HMASMP)**: runs only when the **GUARD** ended with 4, and ends the job with return code 8.
ACCEPT	**ACCEPT S(***fmid***) DIS(WRITE)**, with **BYPASS(REQ)** if **prereq** is not empty ("A Prerequisite That Cannot Be Accepted"). Runs only if the **GUARD** ended with 0.
CLEANUP	IDCAMS: deletes the staging library, and those of the element libraries, if the **GUARD** ended with 0 and the **ACCEPT** ended with 0. With **prereq**, or with a module in a system library ("A Module in a System Library" on page 44), the **ACCEPT** ends with 4 by design, and the step runs at 4 as well, but not above.

The guard. The **APPLY** enters the SYSMOD in the target zone (CDS) and a **RESTORE** takes it out again, so **LIST CDS** ends with 0 exactly while the SYSMOD is applied. A SYSMOD that was received and never applied has no entry there, which is the case after an **APPLY CHECK** that ended with 12, and after an installation job that never ran. Then the job ends with return code 8 in the step **NOTAPPL**, the **ACCEPT** and the **CLEANUP** do not run, and nothing is accepted or scratched. A job step whose **COND** names only a step that did not run is not held back, so the **ACCEPT** and the **CLEANUP** name the **GUARD** themselves.

The guard tests for the entry, not for the status text. A **RESTORE** that fails for lack of the DD cards of the **APPLY** leaves the entry, with the status **REC APP RES ERR**, and **LIST CDS** still ends with 0: the job goes on. After a **RESTORE** that ended with return code 0 the entry is gone and the job stops at the guard.

A clean run ends with return code 0, or with 4 when **prereq** is set. A failed **ACCEPT** (return code 8 or above, for example a staging library that was deleted) leaves the staging libraries alone, and the job can be submitted again once the cause is fixed. Until the **ACCEPT** ran, a **RESTORE** takes the function out again. A **RESTORE** run with the same DD cards as the **APPLY** ends with return code 0.

With **accept_fmid = false** there is no accept job: the level is never accepted, and the **CLEANUP** step stays in the installation job, behind the **APPLY**.

A Prerequisite That Cannot Be Accepted

With **[smp] prereq**, the SYSMOD carries the identifiers in **++VER(...) REQ(...)**, and the **APPLY CHECK** and the **APPLY** refuse to go on while one of them is not applied: they end with return code 12 and **HMA3590 ---** followed by the identifier and **REQ**. SMP checks a requisite against the target zone at **APPLY** and against the distribution zone at **ACCEPT**. A prerequisite that is applied but cannot be accepted, such as a usermod whose own requisite is not accepted either, would let the **ACCEPT** end with 12 on every system.

So when **prereq** is not empty, the ACCEPT of the accept job is

```
ACCEPT S(TABC120) BYPASS(REQ) DIS(WRITE) .
```

for the product's own SYSMOD alone, and the header of the job says why. One **BYPASS(REQ)** covers every identifier of the list. The **APPLY CHECK** and the **APPLY** of the installation job stay strict, and the requisite is still written as **REQ**. The accept job ends with return code 4 and **HMA3590 ---** *id* **REQ BYPASSED** for each prerequisite: that is expected. Without **prereq**, the step has no **BYPASS**.

The installation job does not submit the accept job. The accept job checks what the installation job did (the guard, "The Accept Job" on page 43): after an **APPLY CHECK** that failed on a missing prerequisite the SYSMOD is received and not applied, so the job ends in **NOTAPPL** and does not take the **ACCEPT** with the requisite bypassed. Submit it after an installation job whose **APPLY CHECK** and **APPLY** ended with return code 0, or with 4 for **HMA2461** (a predecessor in **delete**, the normal end of an upgrade), **HMA4441** (a library of **SRC** elements) or **HMA2391** with **RETURN CODE=08** (a module in a system library, "A Module in a System Library") alone, and not after 8 or 12, as the header of the job says. The guard looks for the SYSMOD in the target zone, not at the return code of the installation job.

The SYSMOD

For a release 1.2.0 of a product with two modules, one with aliases, that requires **TXYZ100** and replaces **TABC110**:

```
++FUNCTION(TABC120) .
++VER(Z038) REQ(TXYZ100) DELETE(TABC110)
   /* prod 1.2.0                                              */
   /* Load modules are copied from the LKLIB, not re-bound.   */ .
++JCLIN .
//TABC120 JOB 1,'PROD JCLIN',MSGLEVEL=1,CLASS=A
//COPYLOAD EXEC PGM=IEBCOPY
//AABCLOD  DD  DISP=SHR,DSN=ABC.AABCLOD
//LINKLIB  DD  DISP=SHR,DSN=ABC.LINKLIB
//SYSIN    DD  *
  COPY INDD=AABCLOD,OUTDD=LINKLIB
  SELECT MEMBER=(...)
/*
++MOD(ABCSRV) LKLIB(LKLIB) DISTLIB(AABCLOD)
      TALIAS(ABC,ABCX) .
++MOD(ABCUTIL1) LKLIB(LKLIB) DISTLIB(AABCLOD) .
```

REQ and **DELETE** appear only when **prereq** and **delete** are not empty, the identifiers separated by commas. The first comment card names the product and the version. **LKLIB**, **DISTLIB**, **INDD** and **OUTDD** are the DD names of the data sets ("DD Names" on page 40). The modules follow in name order. A **TALIAS** card that would reach beyond column 71 is an error.

The **++JCLIN** is a copy, so SMP copies each module as it was linked and keeps its attributes, entry point and authorization code. Aliases travel as **TALIAS**.

A Module in a System Library

A module with **[module.*NAME*.smp]** ("[module.NAME.smp]" on page 10) is not copied into the product's target. The JCLIN link-edits it into the system library, and it travels as its assembler source. For **syslib = "LPALIB"**, **distlib = "AOST4"**, the SYSMOD of "The SYSMOD" gets a third comment card, a step of its own in the JCLIN behind the **COPYLOAD**, and the module's **++MOD** as its last statement:

```
   /* Load modules are copied from the LKLIB, not re-bound.   */
   /* IGX00011 is assembled at install, linked into LPALIB.   */ .
++JCLIN .
...
//IGX00011 EXEC PGM=IEWL,PARM='MAP,LIST,NCAL,RENT,REUS,REFR'
//AOST4    DD  DISP=SHR,DSN=SYS1.AOST4
//SYSLMOD  DD  DISP=SHR,DSN=SYS1.LPALIB
//SYSLIN   DD  *
  INCLUDE AOST4(IGX00011)
  NAME IGX00011(R)
/*
++MOD(ABCSRV) LKLIB(LKLIB) DISTLIB(AABCLOD) .
++MOD(IGX00011) DISTLIB(AOST4) .
```

The module is in neither the load TRANSMIT file nor the **COPYLOAD**. The installation job carries its source, and before the **RECEIVE** it writes the SYSMOD to **&&MCS** (**GENER1**) and assembles the source with IFOX00 against the system's own **SYS1.MACLIB** and **SYS1.AMODGEN** (**ASSEM**). The object deck lands behind

the last card of the SYSMOD, the module's **++MOD**, and SMP receives a plain object deck. The **PARM** of the link-edit gives SMP the attributes of the module: SMP links it with them and records them for the load module (**LIST CDS LMOD**, **LKED ATTRIBUTES**). The source has at most 80 columns, plain ASCII, no tabs and no line beginning with **@@** the job's own cards are still held to column 71.

The DD cards. **SYS1.AOST4** is a DD of the SMPAPP procedure of MVS/CE but not of TK5, and the **ACCEPT** writes the module there. The **APPLY CHECK**, **APPLY** and **ACCEPT** steps therefore carry a card **//HMASMP.AOST4** ahead of the product's own: an override on MVS/CE, an addition on TK5, the same text on both. A library every procedure has (**LPALIB**) gets no card. With **procedure** set, the card is written only where that procedure lacks the library. When the product's own libraries override procedure DDs as well, the order differs between the systems, and **mbt package** asks for **procedure**.

Return codes. The **APPLY** link-edits the module with an **INCLUDE** of the module from the system library, where it is not: new, or taken out by the **DELETE** of the level before. The link edit ends with 8 on **IEW0342**, SMP reports **HMA2391 LINK SUCCESSFUL** with **RETURN CODE=08**, and the **APPLY** ends with 4. The **ACCEPT** ends with 4 for the same reason, with **HMA2471 BLDL FAILED IN LIBRARY AOST4** before **HMA2390 LINK SUCCESSFUL**. Both are the normal end, and the **CLEANUP** steps run at 4. The **APPLY CHECK** ends with 0.

The upgrade. A **DELETE** takes the module of the level before out of the library its LMOD entry names. Measured on TK5, with the module shipped before through **COPYLOAD** into the product's target: the **APPLY** deleted it there (**HMA2240** on the target's DD) and linked the new one into **SYS1.LPALIB**, the **ACCEPT** deleted it from the product's distribution library and linked it into **SYS1.AOST4**. From one level in **LPALIB** to the next, the **APPLY** deletes it from **LPALIB** and links it again, the **ACCEPT** does the same in **AOST4**. Both zones then record the new level as the module's **FMID** and **RMID** with **DISTLIB=AOST4**, and the load module with **SYSTEM LIBRARY LPALIB**.

A copy SMP did not put there. The **INCLUDE** of the module from the system library finds a member if one is there, and the **APPLY** takes it into the link. Measured on TK5, with a package with the attributes of the example above (**syslib**, **distlib**, **PARM RENT,REUS,REFR**) installed over a system where an earlier release had copied the module into **SYS1.LPALIB** by hand, not refreshable:

- The **APPLY** link-edit has **INCLUDE SMPWRK3(IGX00011)**, then **INCLUDE LPALIB(IGX00011)**. SMP reports **HMA2390** with **RETURN CODE=00**, not **HMA2391** with 8, and the listing says **MODULE HAS BEEN MARKED REENTERABLE, REUSABLE, AND NOT REFRESHABLE**. **IEHLIST** shows the attribute bytes **C3F2** for the member in **LPALIB**.
- The **ACCEPT** link-edit includes only the new object. The member in **AOST4** is refreshable (**C373**).
- The **ACDS** (the distribution zone) still records **LKED ATTRIBUTES = RENT REUS REFR DC**. The text of the module is byte-identical in all three copies.

The link takes the attributes from the old member, not from the **PARM**, and the **ACDS** records the **PARM**, so it does not show the difference. A first installation, where there is nothing to include, is not affected. The user takes the foreign copy out of the library before the first **APPLY** (not measured as a sequence; derived from the first installation) and reads the attributes of the result from **IEHLIST** or from the listing of the link edit. Not measured: what the missing **REFR** means for the LPA at an IPL with **CLPA**, what the **CDS** records, and a foreign copy that is refreshable.

A module in **SYS1.LPALIB** or **SYS1.SVCLIB** takes effect at the next IPL with **CLPA**, one in **SYS1.LINKLIB** at the next IPL. The installation job says so in its header; the IPL is the operator's. Measured is **LPALIB** with **AOST4**, on TK5 and MVS/CE; **LINKLIB** and **SVCLIB** are accepted by the same rules and not measured.

Libraries as SMP Elements

A library with **`element = "MAC"`** or **`"SRC"`** (“[distribution.library.DIR]” on page 12) is installed by SMP, member by member, not received into its target as a plain data set. Every member becomes an element of the SYSMOD, after the **`++VER`** and before the **`++JCLIN`**:

```
++MAC(DAYSBETW) DISTLIB(ARXLIB) SYSLIB(RXLIB) TXLIB(DAYSBETW) .
++SRC(XMAS) DISTLIB(ASAMPLES) SYSLIB(SAMPLES) TXLIB(XMAS) .
```

`DISTLIB` and **`SYSLIB`** are the DD names of the distribution and the target library, **`TXLIB`** the DD name of the card that points to the member. The members are in name order, the libraries in the order of the configuration in a `project.toml` and sorted by `dir` in an `mbt.toml`.

The staging library and the TXLIB cards. The library travels as a TRANSMIT file of its own and is received into the staging library, by default *`NAME`***`.T`***`DIR`*. The installation job and the accept job have one DD card per element, in the **`RECEIVE`**, **`APPLY CHECK`**, **`APPLY`** and **`ACCEPT`** steps, each pointing to the member of the same name:

```
//DAYSBETW DD  DISP=SHR,DSN=ABC.TRXLIB(DAYSBETW)
```

SMP reads the element again in each of the four steps (the last one is the **`ACCEPT`** of the accept job) and fails with **`HMA2832`** (DDCARD MISSING) when the card is not there. The DD name is the member name, except when that name is taken: a name of the SMP procedures or beginning with **`SMP`** or **`SYS`**, the DD name of one of the libraries of the job, or a name already used by the other type. Those members get **`TX000001`**, **`TX000002`**, and so on. A card that would pass column 71 continues on a second card. The target and distribution libraries are DDs of the **`APPLY`** and **`ACCEPT`** steps and are allocated by the allocation job, like the load libraries.

A library of 116 members and one of 122 make about 240 DD cards in each of the four steps. They fit: the TIOT limit was not reached on MVS/CE nor on TK5 with 235 cards in a step at the job card’s **`REGION=4096K`**.

What a member must be. Fixed 80, no other format. All 80 columns reach the library unchanged, columns 73 to 80 included. **`mbt package`** checks every file before it makes the TRANSMIT file and stops, naming file, line and column, for a line past column 80 (tabs are expanded to every eighth column and trailing blanks do not count), a control character, and a character CP037 has no equivalent for. A file that is valid UTF-8 is read as UTF-8 and a character up to **`U+00FF`** is passed on as Latin-1, which **`xmit370`** converts through CP037; any other file is read as Latin-1, as **`xmit370`** reads it. Nothing is renumbered, cut or stripped.

Return codes. With a library of **`SRC`** elements the **`APPLY`** and the **`APPLY CHECK`** end at return code 4 (**`HMA4441`**, twice per element: SMP has no assembly step for the source and says so), so the job allows 4 there and goes on. The guard of the accept job lists the SYSMOD and ends at 0, so the accept job ends at 0 as well. The installation job’s header says so. A project without such a library keeps the check on 0.

A name SMP already knows. An element name belongs to one SYSMOD. A **`MAC`** whose name is in the target system’s CDS already, delivered by another SYSMOD, is not installed: the element shows as **`NOT SEL`** in the **`APPLY`** and the return code stays 0. Nothing is wrong with the member in the library; the system’s own element stays. A **`SRC`** is worse when its name is that of a load module (**`MOD`**) in the CDS: SMP takes the member for the source of that module and assembles it. When the assembly fails (it did on MVS/CE for a REXX member named like a TSO command) the **`APPLY`** ends at return code 12 with **`HMA2402`** and **`HMA3020`** and the SYSMOD is not applied. If the assembly worked, the module would be replaced.

`mbt package` therefore prints the names of every element library, with the statements to run on the target system before the first installation (a **`LIST`** statement takes one element type, so a library of sources has two):

```
120 SRC elements for ABC.SAMPLES (library samples) -- ...:
    LIST CDS MOD(ALPHA,BETA,...) .
    LIST CDS SRC(ALPHA,BETA,...) .
```

A name listed under **`MODULE ENTRIES`** or under another SYSMOD's entries is a collision: rename the member's file.

Dropping a member. The next minor release names its predecessor in **`delete`**. SMP deletes every element of the predecessor from the target and distribution libraries, and installs those the new SYSMOD has. A member the later version no longer has is gone from both libraries afterwards (measured with 114 and 121 members: the dropped ones were absent from the target library and the distribution library, a new one was present, a changed one had its new content). The **`APPLY`** ends at 4 with **`HMA2461`** and the **`ACCEPT`** at 0, as for a library of load modules only ("The SYSMOD" on page 44).

Staging libraries. The **`CLEANUP`** step of the accept job deletes the staging library of every element library together with that of the load modules, once the **`ACCEPT`** has ended, since SMP has copied what it needs. Until then they stay, because SMP reads the elements from them again at the **`ACCEPT`**. The closing comment of the installation job lists the **`DELETE`** statements for all of them.

Prerelease Builds

A version with **`-dev`** or **`-rc`***n* gets no derived FMID. The FMID derived from **`1.4.0-dev`** is the one of the release 1.4.0, and an installation spends an identifier for good (*MBT User's Guide*, "The Function Identifier"). An installation of the prerelease build would therefore:

- **spend the release's identifier**, so that the release itself can no longer be installed under it on that system;
- **take the modules**: SMP gives a module to the SYSMOD that owns its name, so a later patch release of the previous minor, shipped under an identifier of its own, installs nothing. Its elements are **`NOT SEL`**, and every step still ends with return code 0 (*MBT User's Guide*, "Checking an Installation").

`mbt package` and **`mbt dist`** refuse such a version, with return code 2, unless **`[smp] fmid`** is set:

```
[mbt] ERROR: version 1.0.0-dev is a prerelease: the derived FMID TSUM100 is the release's, and an install of
this build would spend it and block the release and patch releases of the previous minor (NOT SEL) -- set [smp]
fmid and delete explicitly, to a throwaway id (Reference, SMP chapter)
```

`mbt package` builds and writes the load TRANSMIT file and the library archive before it refuses; the installation package is not written. A project without **`[distribution]`** is not affected.

A throwaway installation. To get an installable package from a prerelease, set both keys:

```
[smp]
fmid   = "TTST100"
delete = []
```

The identifier seven characters, a letter and six letters or digits; MBT checks nothing more. It is not checked against the derived one either: **`TSUM100`** for **`1.0.0-dev`** is accepted and has all the consequences above. Take one that no release will carry, for example under a prefix that no product uses. Once installed anywhere, it is spent like any other.

delete is required with an explicit **`fmid`**. **`[]`** means the build replaces nothing, which is right on a system where the product is not installed. A throwaway SYSMOD that replaces nothing does not own the modules the product already has: on a system that carries the product, its elements come back **`NOT SEL`** with return code 0. Give the modules of the throwaway build names the product does not use, or install on a system without the product. Naming the product's identifier in **`delete`** instead deletes the installed level's modules.

Checking the target library's member list shows whether anything was installed, not the return codes (*MBT User's Guide*, "Checking an Installation").

MBT does not notice a throwaway identifier at a release. The version **`1.4.0`** with an explicit **`fmid`** is packaged under that **`fmid`**, without a warning. Remove **`fmid`** from **`[smp]`** again, put **`prefix`** back, and set **`delete`** as "Replacing a Throwaway Installation" describes, before **`mbt release`** (*MBT User's Guide*, "Releasing").

`mbt prerelease` tags the commit and does not package; the release workflow does, on the tag (*MBT User's Guide*, "The Workflows"). For a product with **`[distribution]`** and a derived FMID, the **`Package`** step of **`release3.yml`** fails with the message above and nothing is published, until the project sets a throwaway **`fmid`** and **`delete`**.

Replacing a Throwaway Installation

The release replaces a throwaway installation the way it replaces its predecessor: it names the throwaway identifier in **`delete`**. MBT derives only the identifier of the previous minor. A **`delete`** that is set takes the place of the derived one, it does not add to it. After a throwaway **`TTST1D0`** was installed, the release 1.1.0 therefore names both:

```
[smp]
prefix = "TSUM"
delete = ["TSUM100", "TTST1D0"]
```

For a release *x***`.0.0`** there is no derived **`delete`** at all; set it to the throwaway identifier next to whatever the release replaces, or to **`[]`** for a first level that no throwaway build was installed for. Take **`fmid`** out and put **`prefix`** back, as above.

The following was observed on a scratch MVS/CE system with SMP4, with small C modules that SMP copies from a **`++JCLIN`**, as MBT writes them ("The SYSMOD" on page 44). It was not observed on TK5, nor with other element types such as macros, source or modules shared by several load modules.

The throwaway is named The release is received, applied and accepted. It becomes the owner of the modules in the target zone and in the distribution zone. **`LIST`** shows the throwaway identifier and the previous minor as deleted, with **`DELBY`** naming the release. The target and the distribution library hold exactly the modules of the release, and the modules report the release's version when they run. The **`APPLY`** ends with return code 4 and **`HMA2461`** for every identifier in **`delete`** that is installed, as in an upgrade (*MBT User's Guide*, "Upgrading an Installation").

The throwaway has modules of its own names They are deleted from both libraries, with **`HMA2240`** for each, and the release installs its own. A throwaway build whose modules carry names the product does not use leaves nothing behind once it is named in **`delete`**.

The throwaway was never installed Naming it costs nothing. SMP gives no message and no return code for it. It does record the identifier as deleted by the release, in both zones, so it is spent. List every throwaway identifier that may be installed on any system the release goes to.

Several throwaway levels A second throwaway build can name the first in its own **`delete`**, and the release can name the whole chain. The release then owns the modules, all identifiers are recorded as deleted, and the libraries hold only the release's modules. Installing the same throwaway identifier again, while it is installed, ends with return code 0.

The throwaway is not named. Two things can happen. Neither is an error for SMP:

- **The release has the module names of the throwaway build.** **`RECEIVE`**, **`APPLY`** and **`ACCEPT`** all end with return code 0, and no **`HMA2380`** message appears for any module. The elements of the release

come back **NOT SEL** against the release's own SYSMOD. The release is recorded as applied and accepted, but it owns no module. The throwaway build stays the owner, and it is what runs.
- **The release has other module names.** It installs, and the throwaway build stays installed beside it, with its modules and its identifier not deleted.

Note: In the element summary of a correct upgrade, a module the release applied also shows a line **NOT SEL** that ends in **DELETED**: that is the deleted SYSMOD, which SMP does not select, and it is normal. The failure is a module that is **NOT SEL** against the **new** SYSMOD, with no **HMA2380**.

Checking. Do not rely on the return code to tell these cases from a replacement. What does (*MBT User's Guide*, "Checking an Installation"):

1. **HMA2380 COPY SUCCESSFUL** for each module and each library, with the release's identifier as **SYSMOD**, in the **APPLY** and in the **ACCEPT**;
2. **LIST** of the release and of the throwaway identifier in both zones: the release applied and accepted, the other with **DELBY** naming the release;
3. **LIST** of the modules, with the release as the last updater;
4. the member lists of the target and the distribution library.

A library of its own. A **DELETE** acts on the library that the DD name of the deleting job resolves to. A throwaway installed into a target library of its own, set with **[smp] target**, is not cleaned by the release, whose job points at the product's library. Its modules stay and run from that library, and a library that comes earlier in the STEPLIB than the product's hides the release's modules. Take such a library out of the STEPLIB, or delete it, after the release is installed. For the same reason a throwaway build that names the product's identifier in **delete** deletes the modules of the installed level only in its own target library.

Appendix A. Messages and Return Codes

Return Codes

Code	Kind	When
0	success	
1	build or test	a compile, assembly or link failed; LIBC/370 is older than **`[toolchain]`** asks; packing for a deploy failed; a test failed on the workstation, on MVS or under TSO; an extension failed for a reason other than its configuration.
2	configuration	an error in **`mbt.toml`** or **`targets.toml`**, a wrong argument, an unknown **`--module`** or build name, two sources of one name, a fixture that is wrong, a plugin for another interface, a failed **`mbt doctor`**, a launcher pin that is no version, any failed **`mbt release`** or **`prerelease`**.
3	dependencies	a dependency, tool or plugin could not be resolved, downloaded or checked; dependencies not staged; a **`[tools]`** table that is wrong; a pinned MBT that cannot be fetched.
4	MVS	mvsMF could not be reached, a job failed or was not finished in time, a deploy failed on MVS or stopped at the space check, a test job that could not run or whose output could not be read.
5	data set	**`mbt test --mvs`**: a fixture could not be loaded. Ahead of 1.
99	internal	an error of no known kind; **`mbt migrate`** found a difference it did not intend.
130	interrupted	Ctrl-C or a termination signal, after logging off mvsMF.

Messages

MBT writes its messages with the prefix **`[mbt]`**: progress lines, **`WARNING:`** for what does not stop it, and **`ERROR:`** for what does, followed by indented lines with the details and, where there is one, the way out.

Table 2. Some messages and what to do

Message	Do
`dependencies not staged: ... -- run 'mbt deps'`	run **`mbt deps`**.
`locked '...' no longer fits '...' -- run 'mbt deps --update'`	run **`mbt deps --update`** if the new range is meant.
`GitHub API rate limit reached`	set **`GITHUB_TOKEN`**.

ERROR: writable data in ...*file* **(rent = true): :***line*	make the data **const**, move it, or declare **rent = false**. The same text with **(ac = 1 ...)** or **(RENT by ld370's default)** is a warning.
mbt.toml: unknown key '...' in [...]	the key is misspelt or not one MBT knows: correct it, or take it out (Chapter 2, “mbt.toml” on page 7).
mbt.toml: [[...]] is the v2 spelling -- write one table per entry, [....NAME]	write one table per entry, for a library **[distribution.library.DIR]** (“[distribution.library.DIR]” on page 12).
mbt.toml: [[...]] is an array of tables -- write one table, [...]	the table is written with double brackets, or as an inline array (**x = [...]**, which gives the same message): write it with single brackets, as the message shows.
mbt.toml: ... is a string, not a table -- write [...]	the key is a value where MBT expects a table (the type is a string, an integer, a float or a boolean, or, for a date-time, “a value of another type”): write it as a table, as the message shows.
mbt.toml: [...] holds one table per entry, [....NAME]; '...' is ...	an entry of **[module]** or **[test]** is a value, not a table: write **[module.NAME]** or **[test.NAME]**.
mbt.toml: [distribution.library] holds one table per directory, [distribution.library.DIR]; '...' is ...	a directory under **[distribution.library]** is a value, not a table: write **[distribution.library.DIR]** (“[distribution.library.DIR]” on page 12).
mbt.toml: [tso.NAME] must be a table	an entry of **[tso]** is a value, not a table: write **[tso.NAME]**.
mbt.toml: [tso.NAME] fixtures must be a table: { KEY = ["file", ...] }	write **fixtures** as an inline table that maps each key to a list of files.
mbt.toml: [dependencies] "..." is ..., not a version range -- write "..." = "<range>"	give the dependency as a version range string (“[dependencies]” on page 8).
test NAME: parm is N chars; M is the most the batch EXEC card holds (JCL ends at column 71)	**mbt test --mvs** (exit 2, before anything is uploaded): the **parm** does not fit the batch step even on a continuation card (49 characters). Shorten it.
test NAME: parm is N chars; M is the most the TSO CALL holds (TSO reads columns 1-72)	**mbt test --mvs** (exit 2, before anything is uploaded): the TSO argument does not fit the CALL command (69 characters). Shorten it, or give the batch step its own value with **{ batch = ..., tso = ... }**.
mbt.toml: test.NAME.parm is ... -- write a string, or a table [test.NAME.parm] with batch and tso	write **parm** as one string, or as a table with the keys **batch** and **tso**.
mbt.toml: [smp] fmid is explicit, so delete must be too: ...	name the FMID this release replaces, or write **delete = []** for a product’s first level.
version ... is a prerelease: the derived FMID ... is the release's, ...	set **[smp] fmid** to a throwaway id, and **delete** (“Prerelease Builds” on page 47).
sources sharing a name: ... -- rename one	rename one of the two files.

`... defines N targets and none is default`	give **`--target`**, or **`default = true`** to one target.
`password: keychain ... is empty`	store the password again, in an interactive terminal.
`... has about N of M tracks free`	compress the library, or **`--reallocate`** while nothing holds it, or **`--ignore-space`**.
`deploy job ... ran out of space (SD37 in step MERGE, ...)`	the data set named is the library: compress it while nothing holds it, or **`--reallocate`**, then deploy again. The library may hold some of the new members until a deploy completes. If the data set named is the staging copy, the library was not written; make room for the staging copy and deploy again.
`deploy failed: --reallocate: cannot delete ..., most likely a running server holds it -- stop it first`	stop the server first.
`--reallocate: ... has no room for N+M tracks ... -- nothing was deleted`	free space on the volume, or compress the library while nothing holds it.
`deploy job ... -- ... does not exist: the job did not allocate it`	read the job's messages, then deploy again: a plain **`mbt deploy`** allocates the library.
`the deploy job ended clean, but ... lacks ...`	read **`build/deploy.spool`**.
`cannot release ...: the tree is at ..., expected ...-dev`	release the version the tree is at.
`tag v... already exists locally`	remove the tag an aborted run left, as the message shows.

Index

A

accept job 43
allocation job
  second run 39
APF 39

B

BYPASS(REQ) 43

C

CHECK 39
ctx 24

D

DD name 40
DELETE 41
  throwaway 48
deps.local.toml 21
distlib_dd 40
DLBVOL 38

E

environment variables 31

F

FMID
  throwaway 47
  throwaway, replacing 48

I

IEAAPF00 39
IFOX00 44

L

launcher 5
lklib_dd 40
LPALIB 10, 44
Lua
  API 23

M

mbt build 1
mbt command 1
mbt deploy 2
mbt deps 2
mbt doctor 2
mbt package 3
mbt release 4
mbt target 4
mbt test 5
  --tso 27
  --tso fixtures 29
  --tso screen size 28
mbt.lock 21
mbt.toml 7
MBT_TARGET 32

P

password
  sources 19
plugin 25
predecessor 41
prerelease
  SMP package 47

R

REFR 45
refreshable 45
return code 51

S

SMP
  ACCEPT 43
  package 37
  prereq 43
system library 10, 44

T

target
  selection 19
target_dd 40
targets.toml 17
  further users 17
task 23
test
  no assertions 5
TGTVOL 38