



Technical Guide for Engineers

UDE Memtool

FLASH/OTP Programming Tool: Features, Automation and AURIX TC3xx UCB Programming

Positive One Corporation

This document is based on product documentation and application notes from PLS Development Tools.

Structure of This Guide



01

Product Overview

Positioning, key functions, supported devices, communication interfaces, GangProgrammer, licensing



02

Automation (Batch / COM)

Batch commands and example, COM automation flow, API overview and language samples



03

AURIX TC3xx UCB Programming

UCB overview and warnings, UCB Configuration Function (GUI / API), Direct UCB Programming (DF_UCBS)



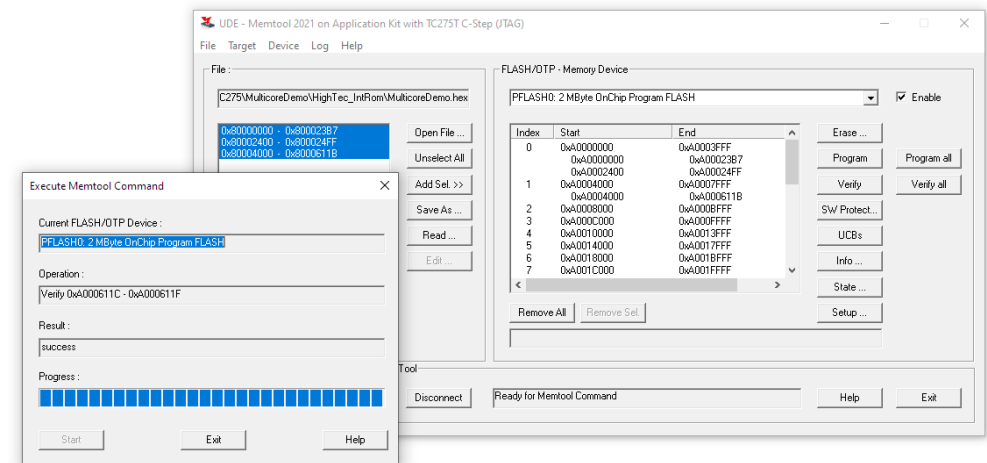
04

Operational Notes

Programming checklist, reference documents, contact

What Is UDE Memtool?

- Tool for programming on-chip FLASH/OTP as well as external FLASH and EPROM
- Runs as an add-in for UDE (Universal Debug Engine): program inside the IDE during development; memory-map aware, runs the programming algorithm when required
- Also usable stand-alone via a separate front end (UDEMemtool.exe)
- All programming functions exposed via standard COM automation for integration into production/test systems and scripts
- Batch mode and command line interface available
- Uses the target's on-chip RAM for programming; handles multiple memory modules



UDE Memtool main window and command execution dialog (source: PLS)

UDE add-in

Stand-alone

COM API

Batch / CLI

Four usage modes to match your workflow

Key Functions (depending on memory type)



Erase

Erase the entire memory module or selected sectors



Load File

Supports Intel HEX and Motorola S-Record



Program

Program all or selected portions of a file



Verify

Compare all or selected portions with memory contents



Read & Save

Read back current memory content and save to disk



Protection

Set / reset chip and sector protection (on-chip only)



UCB / ABM / BMI

Handling of UCB, ABM (Alternate Boot Mode) and BMI (Boot Mode Index) headers (some derivatives)



Multi-Module

Handles more than one memory module per target

Supported Devices



Automotive MCUs (on-chip)

- AURIX / TriCore
- Stellar SR6P / SR6G / SR5E
- S32 Automotive Platform
- RH850
- Power Architecture / PowerPC
- TRAVEO T2G



General-purpose MCUs / Arm (on-chip)

- STM32xx
- Arm Cortex-M0 / M3 / M4 / M7
- Cortex-R4
- Cortex-A8 / A9 / A53
- XMC7000 / XMC1000 / XMC4500
- XC2000 / XE166 / XC800



External FLASH

- Parallel NOR-FLASH
- Serial EEPROM (I2C / SPI / SPIFI)
- NAND-FLASH (on demand)

See <https://www.pls-mc.com> for the complete, up-to-date device list. Further devices are under development or available on request.

Communication Interfaces and Product Variants

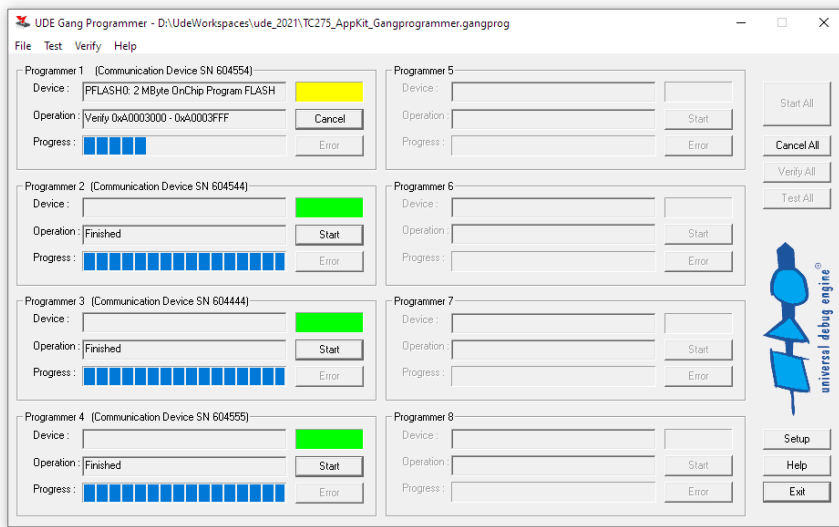
Interface	Target MCUs	Memtool/UAD2 pro	Memtool/UAD2 next	Memtool/UAD3 +	Memtool/ASC
ASC bootstrap loader / ASC	AURIX, TriCore, PowerPC, XMC1000/4500, STM32, XE166, XC2000	✓	✓	—	✓ *1
CAN bootstrap loader	AURIX, TriCore, PowerPC, XMC1000/4500, STM32, XE166, XC2000	✓	✓	—	—
K-Line	AURIX, TriCore, PowerPC, XE166, XC2000	✓ *2	✓ *2	—	✓ *2
JTAG / DAP / SWD	AURIX, TriCore, S32, Stellar, PowerPC, Arm, Cortex, RH850, STM32, XE166, XC2000	✓ *3	✓ *3	✓ *3	—
DXCPL (DAP over CAN physical layer)	AURIX, TriCore	✓ *3	✓ *3	—	—

Access devices: UAD2pro / UAD2next / UAD3+ = Universal Access Device; ASC = host serial port (RS232)

*1 No additional hardware required *2 External K-Line converter required *3 Ask for supported derivatives (<https://www.pls-mc.com>)

Access devices connect to the host PC via USB or Ethernet. Channels include ASC / CAN bootstrap loaders, CAN combined with ASC, and JTAG, plus host RS232 (COMx) and K-Line.

GangProgrammer and Licensing



GangProgrammer window (source: PLS)

GangProgrammer (parallel production programming)

- Manages up to 8 programming stations simultaneously to save time
- Access devices connect via USB, IEEE1394 or Ethernet using hubs
- Each target managed by its own target configuration

Licensing

One license per seat

More than one MCU architecture requires an architecture upgrade

1-year maintenance

Support contract included; renewable every year

One adaptation included

Defined via the Memtool Configuration Form; customers can create further adaptations

Other models

Department / company, OEM license and full-custom versions on request

Three Ways to Automate



Batch Mode

- Runs a text file containing a list of commands
- GUI menu: File > Run Batch
- Or as a start parameter: Memtool[.exe] file
- Ideal for repeatable programming procedures



Command Line (MemtoolCli.exe)

- Programming from the console
- Supports Direct UCB Programming
- Call from production-line scripts



COM Automation

- COM object with ProgID "UDE.Memtool"
- Control from JScript, VBScript, Perl, Python, C# and more
- Returns results such as verify difference count; UCB configuration API available
- For integration into production and test systems

* For MemtoolCli.exe options, refer to the manual of your UDE version.

Batch Mode: Command Reference

Command	Description
<code>connect</code>	Establish connection to the target system. If a connection is already alive, it must be disconnected first
<code>disconnect</code>	Shut down the connection to the target system
<code>open_file file [, [+ -]<offset>]</code>	Open a HEX file and show all sections in the left list box. file must be an absolute path. The optional offset is added to all addresses in the HEX file
<code>select_all_sections</code>	Select all sections in the left list box
<code>add_selected_sections</code>	Add the selected sections from the left list box to the right one (programming targets)
<code>program</code>	Start programming the FLASH device



How to run: select File > Run Batch in the GUI and choose the batch file, or pass the batch file as a command line parameter when starting Memtool[.exe]

Batch Mode: Example (programming four files at once)

```
connect
open_file c:¥Data¥starter.hex
select_all_sections
add_selected_sections
open_file c:¥Data¥firmware1.hex
select_all_sections
add_selected_sections
open_file c:¥Data¥firmware2.hex , +40000h
select_all_sections
add_selected_sections
open_file c:¥Data¥sernum.hex
select_all_sections
add_selected_sections
program
```

1

Connect

connect establishes the target connection

2

Collect sections

Repeat open_file > select_all_sections > add_selected_sections to build up the right-hand list

3

Apply offset

All addresses in firmware2.hex are shifted by +40000h

4

Program all

A single program command at the end writes all collected sections

Tip: always specify files with absolute paths.

COM Automation: Basic Flow

UDE is based on Microsoft COM technology. Any client or scripting host that can use COM (ActiveX) components can remote-control UDE Memtool.



VerifySections returns the number of differences. Treat any non-zero value as a programming failure.

Key API (UDE.Memtool object)

Method / Property	Arguments	Function
Version	— (property)	Returns the Memtool version
CreateLogFile(path, level)	Log file path, log level (50 in samples)	Enables logging to file for diagnosing problems
Init(cfgFile)	Target configuration file (.cfg)	Initializes the component with a target configuration
ConnectTarget()	—	Connects the target system
OpenFile(hexFile, flag)	HEX file path, boolean (true / 1 in samples)	Loads the target application file
ProgramSections()	—	Programs all loaded sections to all enabled FLASH devices
VerifySections(arg)	String ("" in samples)	Verifies programmed sections; returns the difference count
DisconnectTarget()	—	Disconnects the target
Cleanup()	—	Cleans up the component (releases resources)
GetFlashMod(name)	Module name (e.g. "PFLASH0")	Returns a FLASH module object (used e.g. for UCB configuration)

* This table summarizes the APIs used in the PLS application note samples. See the UDE manual for the full API specification.

Sample: Programming and Verifying with Python

```
import os
import win32com.client

WorkDir = os.path.abspath(".")
CfgFile = WorkDir + "¥¥TriBoard_TC27x_dap.cfg"
HexFile = WorkDir + "¥¥MulticoreDemo.hex"
LogFile = WorkDir + "¥¥Memtool.log"

Memtool = win32com.client.Dispatch("UDE.Memtool")
print("Version: " + Memtool.Version)

Memtool.CreateLogFile(LogFile, 50)
Memtool.Init(CfgFile)
Memtool.ConnectTarget()
Memtool.OpenFile(HexFile, True)
Memtool.ProgramSections()
Diffs = Memtool.VerifySections("")
print(str(Diffs) + " differences found")

Memtool.DisconnectTarget()
Memtool.Cleanup()
```

Requirements and Usage

- Windows with UDE Memtool installed
- Uses Python win32com (pywin32 package)
- Run: python MemtoolAutomationDemo.py
- The .cfg is a target configuration created with UDE; replace it with your own board's configuration
- Writes a log file at level 50 for troubleshooting

Tip for production use

Return a non-zero exit code when Diffs is not 0 so the production line can flag the unit as failed.

Differences Between Language Samples

Language	How to run	Object creation	OpenFile 2nd arg
JScript	<code>cscript MemtoolAutomationDemo.js</code>	<code>new ActiveXObject("UDE.Memtool")</code>	<code>true</code>
VBScript	<code>cscript MemtoolAutomationDemo.vbs</code>	<code>CreateObject("UDE.Memtool")</code>	<code>true</code>
Perl	<code>perl MemtoolAutomationDemo.pl</code>	<code>new Win32::OLE(\$ProgId)</code>	<code>1</code>
Python	<code>python MemtoolAutomationDemo.py</code>	<code>win32com.client.Dispatch("UDE.Memtool")</code>	<code>True</code>
C#	Build and run	<code>new UDELib.UDEMemtool()</code>	<code>1</code>



C#

Add UDENetCOMLib.UDELib to the project references and declare using UDELib; VerifySections returns int



VBScript

Calls without return value are written without parentheses (e.g. `Memtool.CreateLogFile LogFile, 50`)



Perl

Uses `use Win32::OLE`; exit with `die` if object creation fails

What Are UCBs (User Configuration Blocks)?

- Special data FLASH sectors in the AURIX TC3xx family that store configuration information
- Configure read/write protection of on-chip FLASH, startup behavior, debug interface lock, etc.
- See the device User's Manual for which UCBs each derivative provides and their content

Example UCBs (tabs of the UCB Handler on TC38x)

UCB_BMHD0	UCB_BMHD1	UCB_BMHD2	UCB_BMHD3
UCB_PFLASH	UCB_DFLASH	UCB_DBG	UCB_HSM
UCB_HSMCOTP0	UCB_HSMCOTP1	UCB_OTP	UCB_SWAP



General Warning

Programming invalid information to a UCB or completely erasing a UCB FLASH sector may leave the controller unable to execute its on-chip startup software. It will then no longer be possible to connect with UDE to correct the UCB configuration. Always use proper UCB programming data.



Hint: When changes take effect

A changed UCB configuration only takes effect after a reset. In Memtool, disconnect and reconnect the target before checking the result.

Two Ways to Program UCBs



① UCB Configuration Function

For developers

- Set up and program UCB content via configuration dialogs
- Change or remove existing configurations where the UCB allows it
- Built-in measures against invalid configurations
- Export a tested configuration to a HEX file
- Also available through the COM object model (scriptable)



② Direct UCB Programming

For production

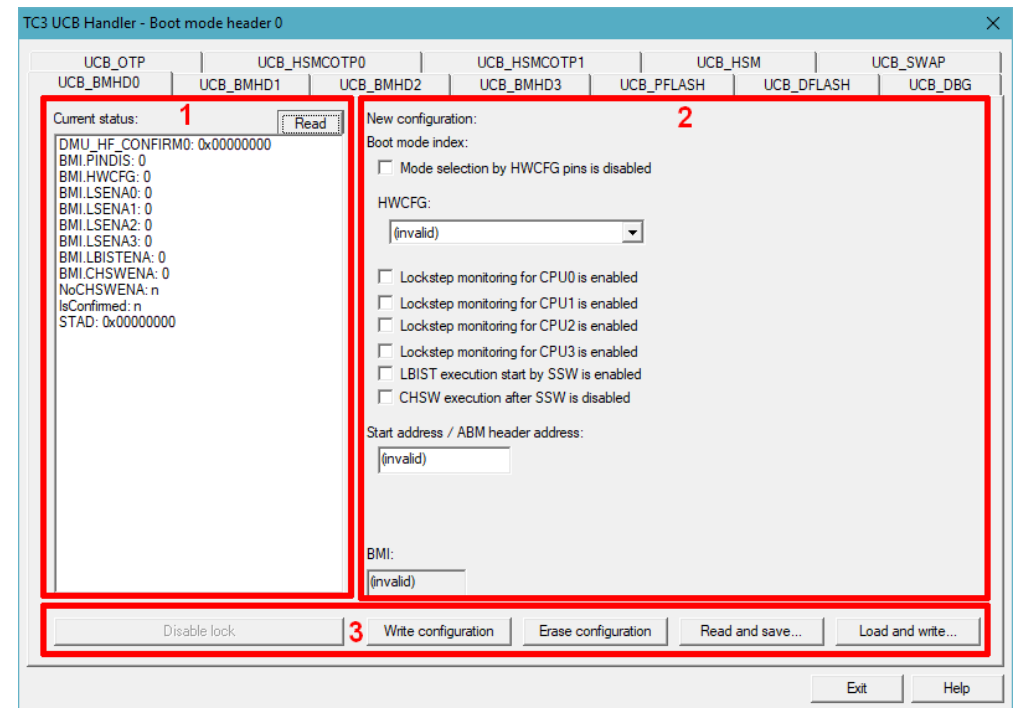
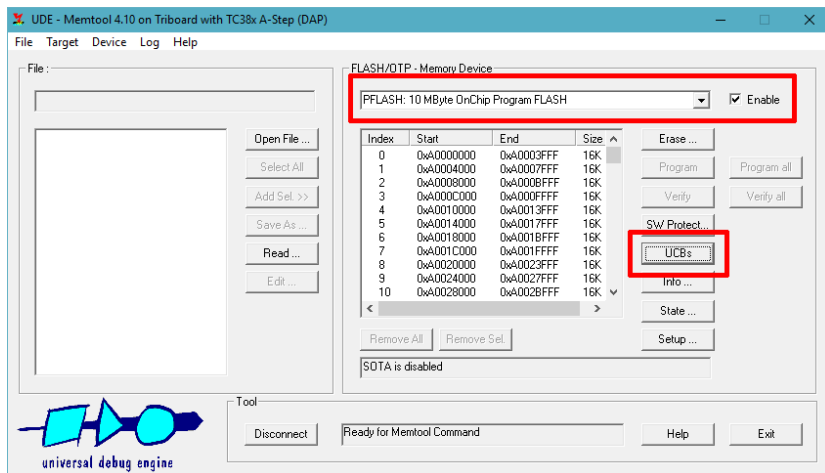
- Load and program validated HEX files like any other FLASH (e.g. PFLASH)
- Supported by the command line tool MemtoolCli.exe
- Memory device DF_UCBS is disabled by default and must be enabled first
- UCBs are programmed only completely; partial HEX data is rejected
- Additional checks block invalid data (e.g. invalid confirmation codes)

Recommended flow: create and test with ① in development > export to HEX > program the same HEX with ② in production

Using the UCB Configuration Function (GUI)

UCB programming is associated with the PFLASH memory device.

- 1 In FLASH/OTP - Memory Device, select PFLASH and check Enable
- 2 Click UCBs to open the TC3 UCB Handler dialog
- 3 Select the UCB tab, prepare the configuration and execute



1 Current status

Registers and bit-fields affected by the UCB

2 New config

Prepare the new UCB configuration

3 Buttons

Execute UCB handling functions

UCB Handler Functions



Disable lock

Unlocks a password-protected UCB. Without the 256-bit (8 x 32-bit) password, its content can be neither accessed nor changed



Write configuration

Writes the new configuration. For password-protected UCBs you are asked for a new 256-bit password (option to set the UCB to confirmed state)



Erase configuration

Does not actually erase; overwrites the UCB with an empty configuration. Not available for OTP UCBs



Read and save...

Reads the current UCB content and saves it to a HEX file



Load and write...

Loads a HEX file and programs its content to the UCB

Password format: enter eight comma-separated 32-bit values, e.g. "1,2,3,4,5,6,7,8" (PW0=0x00000001 ... PW7=0x00000008). Use Check to confirm the expanded result.

Automating UCB Configuration (C# API example)

```
// access FlashModExt object
UDELib.DIUDEFlashMod FlashMod =
    m_Memtool.GetFlashMod("PFLASH0");
UDELib.DIUDEFlashModExt FlashModExt =
    FlashMod.GetExtension("UCB_DFLASH");

if (!FlashModExt.ReadStatus()) return -1;

if (FlashModExt.Locked) {
    if (!FlashModExt.DisableLock("1,2,3,4,5,6,7,8"))
        return -1;
}

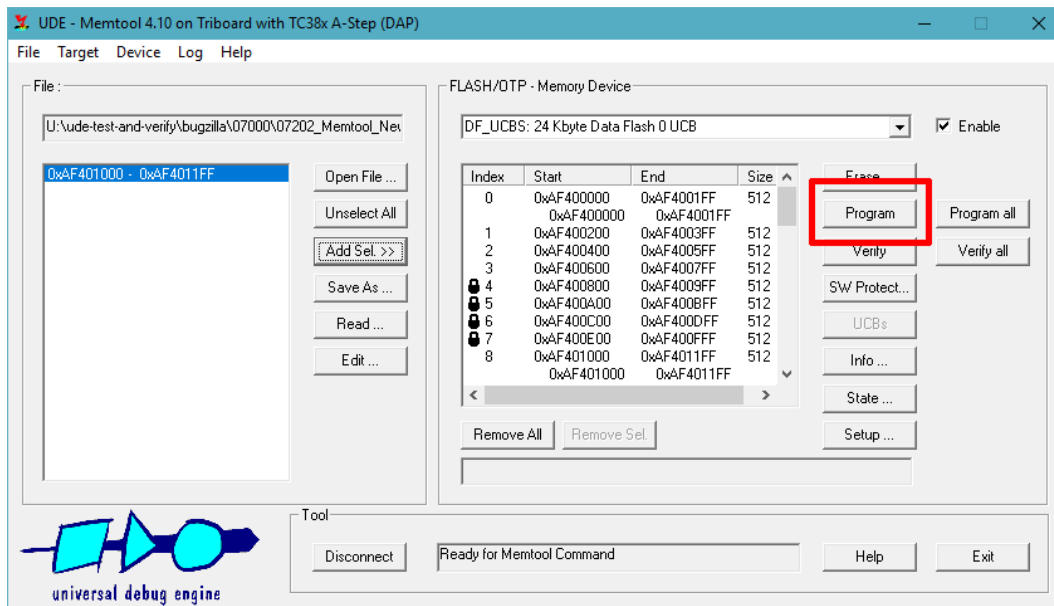
FlashModExt.PrepareNewConfig();
var Config = FlashModExt.Config;
Config.SetItem("PROCONRAM.RAMINSEL1", 1);
Config.SetItem("PROCONRAM.RAMINSEL2", 1);

if (!FlashModExt.WriteConfig()) return -1;
```

- 1 GetFlashMod / GetExtension**
Get the extension object for the UCB (e.g. UCB_DFLASH) from PFLASH0
- 2 ReadStatus()**
Read the current status from the target
- 3 Locked / DisableLock()**
If locked, unlock with the 256-bit password
- 4 PrepareNewConfig()**
Prepare a new configuration based on current status
- 5 Config.SetItem()**
Change items by "REGISTER.BITFIELD" name
- 6 WriteConfig()**
Write the new configuration to the target

ReadStatus, DisableLock and WriteConfig return success/failure; abort on failure.

Direct UCB Programming (DF_UCBS)



Enable DF_UCBS, add the HEX sections and click Program (source: PLS)

MemtoolCli.exe supports the same method.

1

Select memory device DF_UCBS (e.g. 24 Kbyte Data Flash 0 UCB) and check Enable

2

Each UCB appears as a separate FLASH sector (e.g. from 0xAF400000, 512 bytes each)

3

Open the HEX file(s) with Open File and add them with Add Sel.

4

Click Program, then Verify

Notes

- UCBs with a lock symbol cannot be programmed by the user
- No Erase function; program a HEX file containing an empty UCB configuration instead
- HEX files covering only part of a UCB are rejected

Programming Checklist



Before

- Target configuration (.cfg) matches the board and interface
- Address layout and offsets of HEX / S-Record files checked
- Batch file paths are absolute
- UCB data was created and tested with the UCB Configuration Function



During

- Logging to file (CreateLogFile) is enabled
- Do not issue connect while already connected
- Enable DF_UCBS only when needed
- Never fully erase or partially program a UCB



After

- Verify difference count is 0
- After UCB changes, disconnect/reconnect (reset) and check
- Always finish with DisconnectTarget() and Cleanup()
- Record and archive results and logs

References and Contact

PLS documents this guide is based on

- UDE FLASH Programming data sheet (2025_1126)
- Application Note: UDE Memtool Programming in Batch-Mode (2023_1114)
- Application Note: UDEMemtool Automation Basics (2023_1114)
- Application Note: Programming AURIX TC3xx UCBs with UDE Memtool (2025_0512)



Positive One Corporation

Please contact us for questions on introducing UDE Memtool, licensing, target configuration, automation or UCB programming.

Manufacturer: PLS Programmierbare Logik & Systeme GmbH (<https://www.pls-mc.com>)

UDE and Universal Debug Engine are registered trademarks of PLS. AURIX is a trademark of Infineon Technologies AG. Other product names are trademarks of their respective owners. Contents are subject to change without notice.