User Guide

CMSIS-DAP Debug Probe Software
for Chips with Cortex-M and RISC Cores

Version 1.2.2, 15.09.2026

Table of Contents

1. Key features of the CMSIS-DAP utility

The CMSIS-DAP is software for programming and debugging microcontrollers with Cortex-M and RISC cores. CMSIS-DAP is designed to work with the “Debug probe kit for chips with a Cortex-M core TSKYA.468998.109 (the original part number is ТСКЯ.468998.109)”, developed by JSC “ICC Milandr”.

The debug probe firmware supports two data exchange protocols with the PC - CMSIS-DAP v1 and CMSIS-DAP v2. Protocol v1 uses the USB HID class and is the stable, fully verified version (**STABLE**, 1.0.3). Protocol v2 uses Bulk data transfer, which provides higher programming and debugging speed compared to v1, and is currently under active development (**BETA**, 2.0.0). Version 2.0.0 fully retains backward compatibility with protocol v1. Switching between versions is described in the “The Debugger Firmware Update” section (Section 2.5).

Programming flash memory:

- Work with main and information (v1.5+) flash memory areas;
- Mass erase;
- Write firmware to the microcontroller using files in the HEX[1] and ELF[2] formats;
- Save firmware from the microcontroller to a file in the HEX[1] format;
- Compare firmware from the microcontroller with data in HEX[1] and ELF[2] files.

Debug features:

- Download a program to RAM with subsequent execution;
- SWO debug output with saving to a TXT file;
- Read/write core registers and memory variables;
- Hardware RESET signal generation.

Additional features:

- Debugger firmware updates;
- Ability to execute user scripts before reading, writing, or erasing memory;
- Support for the UART bootloader by JSC “ICC Milandr” (only for CMSIS-DAP v1).

> ⚠ **Warning**
>
> When using the debug probe (first revision), a hardware reset of the microcontroller must be performed exclusively via the debug probe. Other hardware reset methods may lead to debug probe failure.

[1] Requirements for HEX format files (not applicable to software version 2.0.0 and later):
- sorted addresses must be in ascending order;
- maximum line length is 16 bytes.
A file in the HEX format generated by the CodeMaster ARM development environment does not meet these requirements, so you must follow the instructions at https://support.milandr.ru/base/spravka/32-razryadnye-mikrokontrollery/sredy-otladki-i-programmatory/36885/.

[2] Only ELF format files generated by IAR EWARM and ARM Keil environments.

1.1. System Requirements

To run the CMSIS-DAP utility, the PC must meet the following requirements:

- operating system Windows 7 SP1 or later, 32-bit or 64-bit;
- a free USB port;
- administrator rights during software installation;
- at least 100 MB of free disk space.

2. Description of the CMSIS-DAP utility interface

The main window of the utility is shown in fig. 1. Starting from version 2.0.0, the utility supports light and dark UI themes, which can be switched using the slider in the bottom right corner of the window.

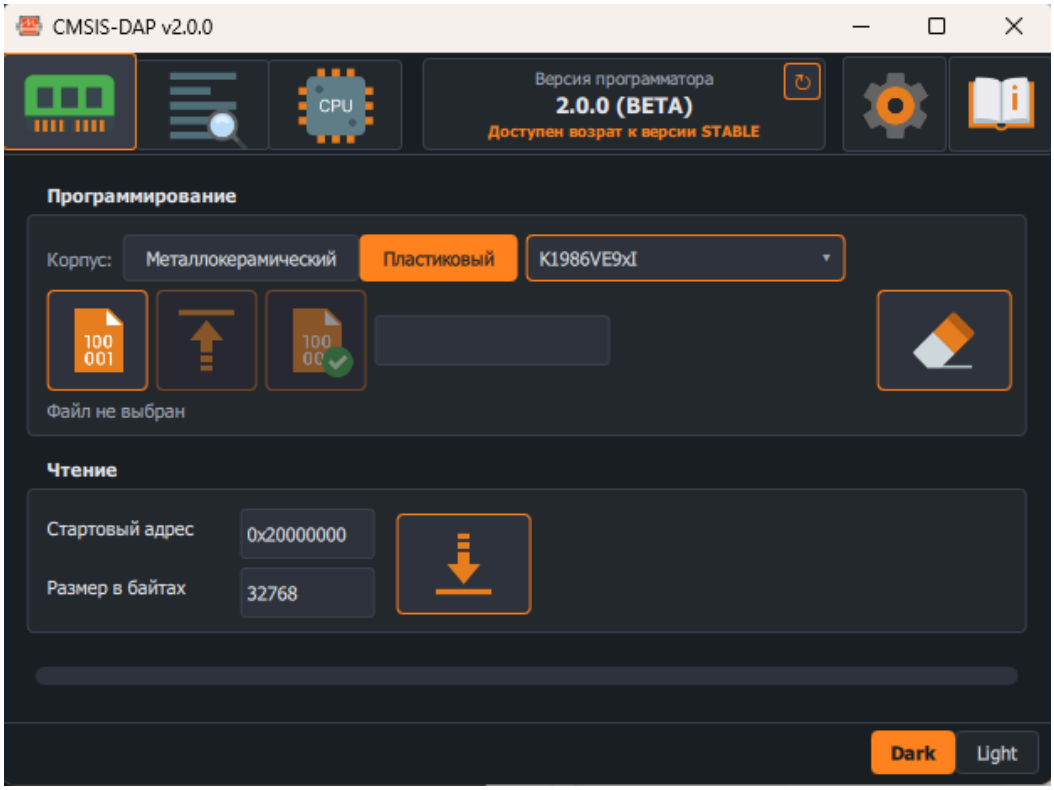

Figure 1. The general view of the utility dialog window (dark theme)[3]

The utility contains three dialog tabs:

1. Memory programming tab;
2. SWO output data display tab;
3. Core registers and memory variables access tab.

There are settings and help buttons in the right upper corner.

[3] The view of the main window of the “CMSIS-DAP utility” can differ depending on the utility version.

2.1. Memory Programming Tab

The first tab of the utility “CMSIS-DAP” – “Memory programming” (fig. 2) allows to work with the memory of the target. Allowed operations are read, write, verification of the flash content and erase. The memory erase operation executes the mass erase of the flash of the target device.

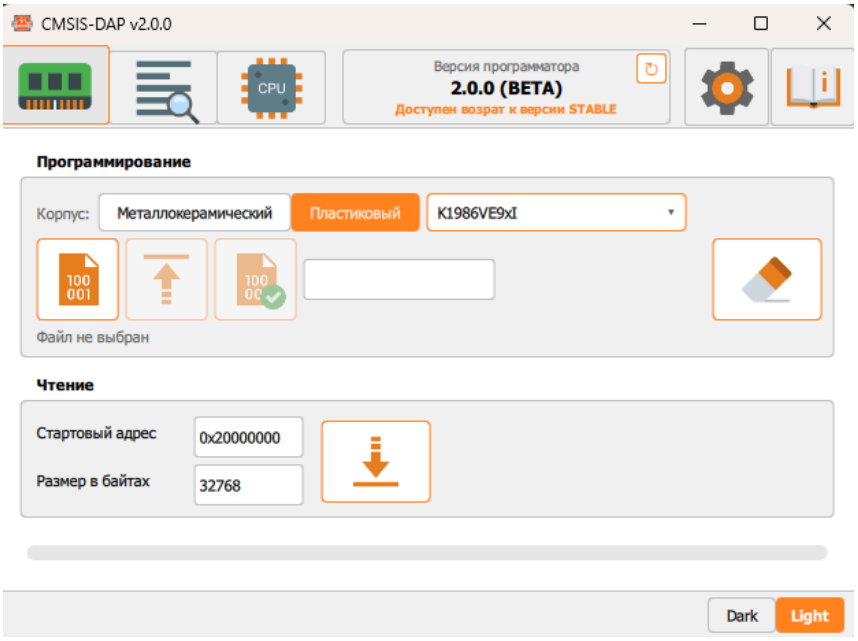

Figure 2. Memory programming tab

> **Note**
>
> The operation with the flash is disabled while SWO data display is running.

2.1.1. Writing to the Memory

To download the data you need to (fig. 3):

1. Use the switch to select the microcontroller package type - “Металлокерамический” or “Пластиковый” (added in version 2.0.0). The corresponding drop-down list of loaders is activated depending on the selection;
2. Select the required loader from the drop-down list. A list of available flash loaders is provided in the “Built-in Loaders” section (Section 5.2);
3. Select a file (HEX[1] or ELF[2]) for programming. After selecting, the full path to the file and its CRC value will appear below the write button;
4. Press the write button to start downloading and wait for the writing operation to complete.

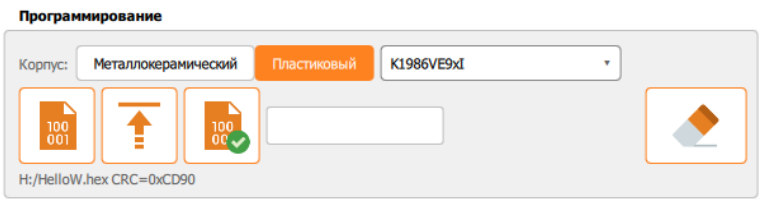

Figure 3. Selecting a file for programming

2.1.1.1. UART loader (only for CMSIS-DAP v1)

> **⚠ Warning**
>
> The UART bootloader functionality is available only for the debug probe with CMSIS-DAP v1 firmware. The UART loader allows access to RAM (read/write) and program launch, but Flash memory access is not implemented.

If the loader supports the ability to work via UART, then you can enable this mode by switching the slider. After that, you can set the required baud rate (fig. 4).

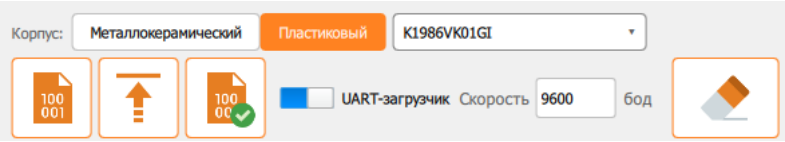

Figure 4. UART loader set up

Then you must set the UART loader mode for the microcontroller and set up the connection as in tab. 1.

Table 1. UART interface connections

Debug Probe Pin Number	Target Pin Name
17	TX
19	RX
15	Reset
1	Vcc
1. Connect the grounds. 2. The connection of the Reset pin is not obligatory but if not connected you must reset the target yourself before using the UART interface[4].	

2.1.1.2. Additional options

If the loader supports supplementary options, then you can enter them in the field next to the verification button (fig. 5).

[4] **Attention**: during the operation of the debug probe (first revision) with the microcontroller, the hardware reset of the latter is allowed to be performed exclusively by means of the debug probe; any other methods of hardware reset may lead to failure of the debug probe.

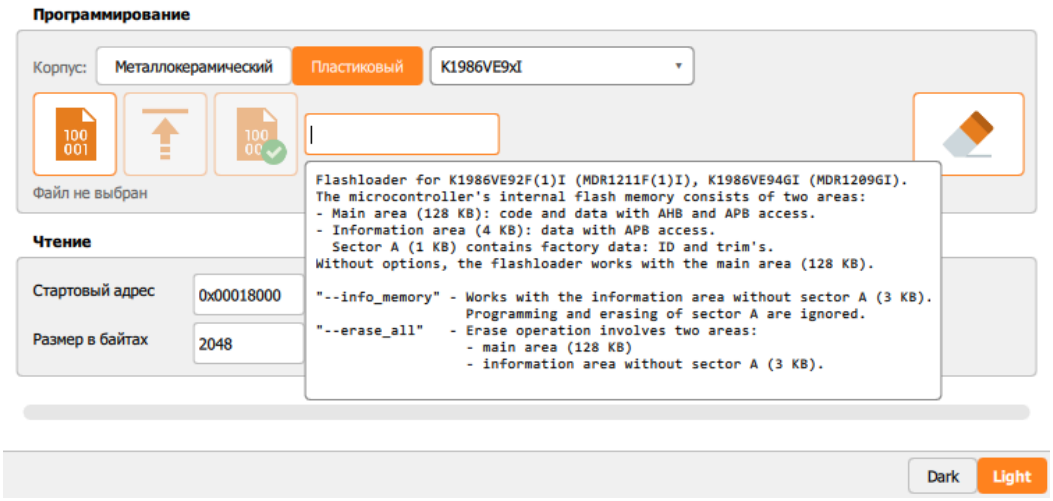

Figure 5. Popup window with additional options

When you hover over this field, a tooltip describing the available options will appear. The list of the options for each built-in loader is given in the Section 5.2.

2.1.2. Verification of the programmed data

To verify the data you need to:

1. Select a file (HEX[1] or ELF[2]) for programming. After selecting the file, its full path and the CRC value will appear below the button;
2. Press the verification button to start the verification and wait for the operation to complete;
3. The result will be displayed below the current progress status bar. The CRC of the memory data will be displayed too (fig. 6).

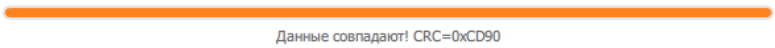

Figure 6. CRC of the memory data

2.1.3. Reading into a File

To read data from the memory to an Intel HEX file you need to (fig. 7):

1. Specify the start address and the size of the loaded data in bytes;
2. Press the read button to read the data into a file and specify a file name to save the data;
3. Wait for the process to complete. After that, the amount of data which has been read and CRC value will be displayed.

Figure 7. Reading data into a file

2.2. SWO Output Data Tab

The second dialog tab, labeled “SWO Output Data” (fig. 8), shows the data received via the debugger using the SWO protocol. All received data can be exported to a plain text file (.txt).

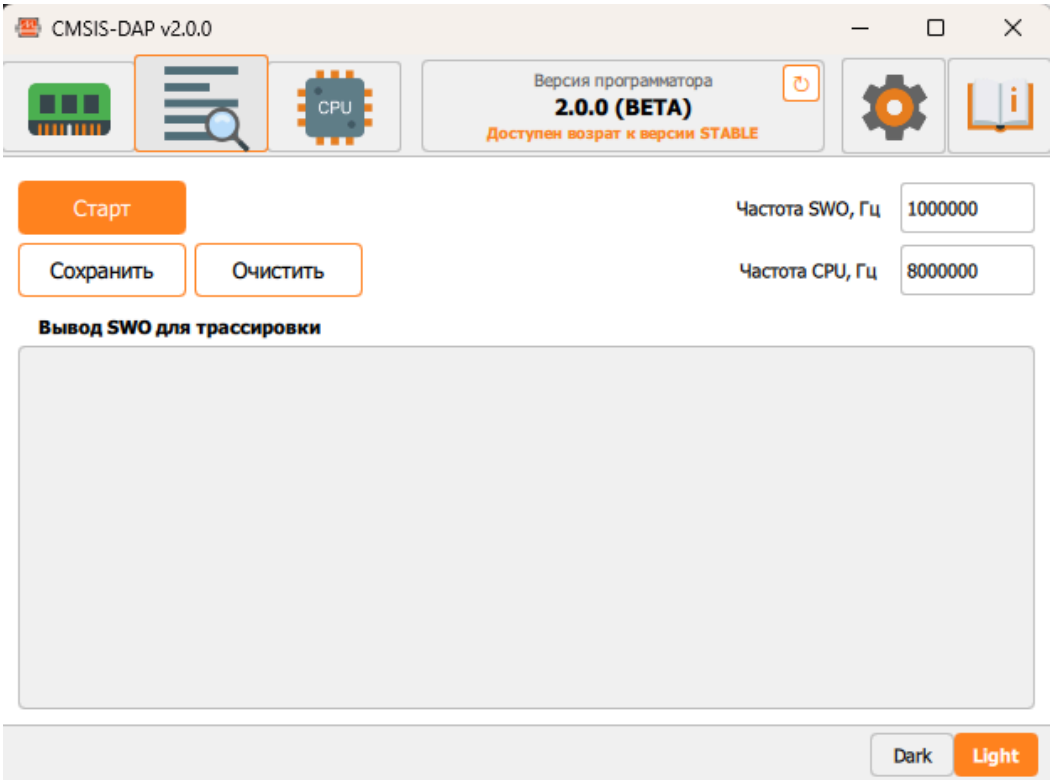

Figure 8. SWO output data tab

To receive the data you need to:

1. Specify the desired SWO interface exchange frequency (lower than 1000000) (the field “Частота SWO”) and set the current working frequency of the target (the field “Частота CPU”);
2. Press the button “Старт” (which means “to start”);

You need to press the button “Сохранить” (which means “to save”) to save the received data to a plain text file. The button “Очистить” (which means “to clear”) erases all data in the SWO output field.

> **Note**
>
> SWO is unavailable if the JTAG interface has been selected or any operation with the memory is being performed.

2.3. Core Registers and Memory Variables Access Tab

The third dialog tab, labeled “Core registers and memory variables access” (fig. 9), enables reading and writing of the core registers and memory variables. Press the button “Подключить” (“Connect”), to enable the data entry fields. When accessing memory, you can select the variable display format: binary, hexadecimal, or decimal.

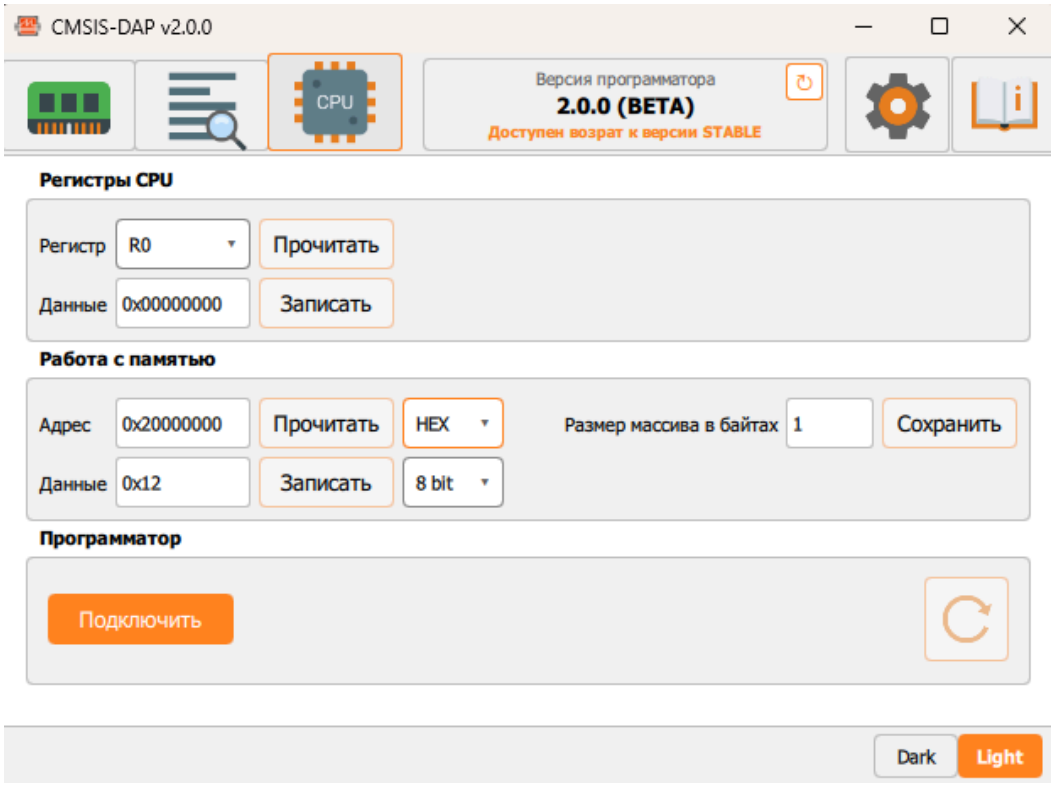

Figure 9. Core registers and memory variables access tab

While reading core registers, the core briefly stops for about 5 ms. It’s important if the program controls the equipment which is critical to the response time.

When the reset button is pressed, a hardware RESET signal is generated (the RESET pin is set to a low level for ~100 ms).

2.4. The Utility Settings

Click the “settings” button to open the utility settings dialogue window (fig. 10).

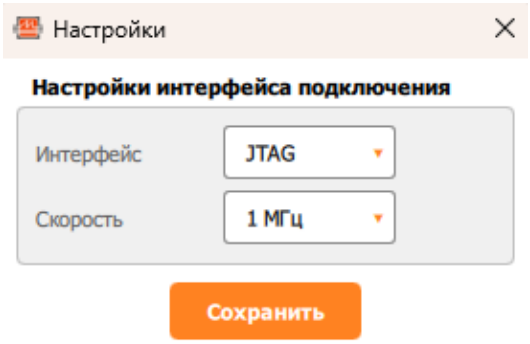

Figure 10. The dialog window "Настройки" ("Settings")

JTAG and SWD interfaces can be enabled in this dialogue. The interface speed can be defined - 100kHz, 500kHz or 1MHz. Press the “Сохранить” button to apply the changes.

2.5. The Debugger Firmware Update

Starting from version 2.0.0, the utility interface provides the ability to update the debug probe firmware without third-party tools. A button located at the top of the program window displays, in its default state, the connection status of the debug probe and the current version of its firmware (fig. 11).

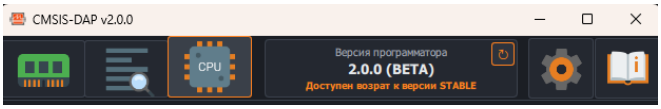

Figure 11. The debug probe firmware update button

Pressing the button allows you to choose the debug probe firmware version - **STABLE** (1.0.3) or **BETA** (2.0.0), see Section 1. During the update, the debug probe LED will turn blue (tab. 2), and a progress slider is displayed at the bottom of the program window.

> **Caution**
>
> Do not disconnect the debug probe from the PC during the update.

Upon completion, a message about the successful update will be displayed. When using version 2.0.0 (BETA), it is possible to roll back to version 1.0.3 (STABLE).

> **Note**
>
> If you have any comments regarding the operation of version 2.0.0 (BETA), please send them to the technical support department at support@milandr.ru.

3. Pin assignment of the Debug Probe

Different variants of pin assignment of the debug probe in different working modes are shown in fig. 12.

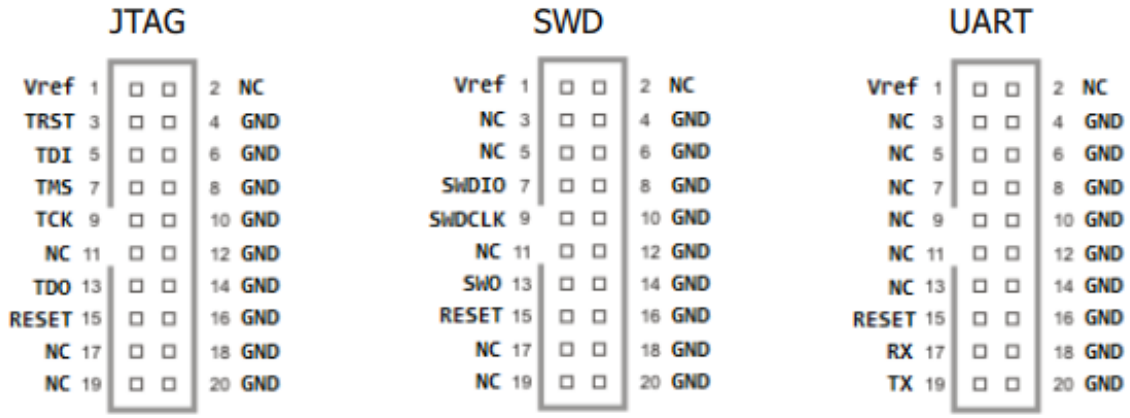

Figure 12. Variants of pin assignment

It is necessary to provide the voltage on the Vref pin. The permissible voltage range is (2.7 - 5.5) V.

Caution

This debug probe cannot supply power to the target board!

4. Description of the LED Indication of the Debugger

There is a full-color LED on the top of the case. It can show different indications depending on its current state (tab. 2, tab. 3).

Table 2. Description of the LED indication of the debugger with CMSIS-DAP v1 firmware

LED state	Description
Blinking red	No USB connection
Permanent red	The debugger is in the standby mode
Permanent blue	The debugger is in the software update mode
Single blue flash	The reset pin was set to low, and then to high level
Permanent green	The debugger is connected to the target
Blinking green	The debugger transmits data in the UART mode
Blinking red	The debugger receives data in the UART mode

Table 3. Description of the LED indication of the debugger with CMSIS-DAP v2 firmware

LED state	Description
Permanent red	The debugger is in the standby mode or no USB connection
Permanent blue	The debugger is in the software update mode
Permanent green	The debugger is connected to the target

If the debugger is used with the development environment, the state of the LED is defined by the IDE. For example, when working in the IAR EWARM, a permanent yellow light indicates the program is running, while a permanent green light indicates it is suspended.

5. Flash Memory Loaders

5.1. Loader Format

The utility uses the loader mechanism to write data to the target. The loader is a compiled program for the target. It contains program functions which provide erasing, writing and verification of the data. This program is loaded to the RAM of the target.

The program is compatible with bootloaders used in the IAR EWARM. If there is the need to add one's own bootloader, one has to create it in accordance with the guide https://netstorage.iar.com/FileStore/STANDARD/001/002/595/arm/doc/FlashLoaderGuide.ENU.pdf.

Then the loader is compiled and 3 files are added to the "flashloaders_ceramic" or "flashloaders_plastic" folder located in the CMSIS-DAP.exe program directory:

1. loader.flash
2. loader.out
3. script.js

***.flash**

The file contains the flash memory description. Tags "macro" and "aggregate" are not used. Special tags are added:

- uart – if set to 1, then there is a UART-loader in the target device. The UART-loader by JSC "ICC Milandr" is only supported;
- uart_baud – the UART speed to synchronize with the loader;
- js – JS code file.

***.out**

An ELF file with the loader for the flash.

***.js**

An optional file with the JS code. It is executed before reading, writing and erasing of the memory. One can disable specific protections, stop Watchdog and do other preparatory actions in it.

When the program is launched, it scans the "flashloaders_ceramic" and "flashloaders_plastic" directories for the *.flash files. When the file is found, the program attempts to open *.out file defined in the exe tag. The full path to the file is ignored, the file name is only used. When the file is successfully opened, an item with the same name as the name of the *.flash file is added to the list of the loaders.

The mass erase of the memory during the memory initialization is supported. The FLAG_ERASE_ONLY flag is set while executing FlashInitEntry to provide the mass erase. If this option is not supported by the loader, then the memory will be erased sector by sector.

The function FlashChecksumEntry is called to check the CRC of the written data. If this function is not available in the loader, then the written data will be read with subsequent CRC calculation.

5.2. Built-in Loaders

5.2.1. Ceramic Package Microcontrollers

5.2.1.1. K1986VE1x_002TU

Flashloader for K1986VE1T (12134), K1986VE1AT (12144). The microcontroller's internal flash memory consists of two areas:

- Main area (128 KB): code and data accessible via AHB and APB.
- Information area (4 KB): data accessible via APB. Sector A (1 KB) contains factory data: ID and trim's.

Without options, the flashloader works with the main area (128 KB).

Options:

- `"--info_memory"` - works with the information area without sector A (3 KB). Programming and erasing of sector A are ignored.
- `"--info_memory_factory"` - works with the entire information area (4 KB).
- `"--erase_all"` - erase operation involves two areas:
  - main area (128 KB)
  - information area without sector A (3 KB).
- `"--erase_all --info_memory_factory"` - full erase operation involves two areas:
  - main area (128 KB)
  - information area (4 KB).

5.2.1.2. K1986VE9x_002TU

Flashloader for K1986VE91T (12104), K1986VE92U(1) (12115(A)), K1986VE93U (12125), K1986VE94T (12094). The microcontroller's internal flash memory consists of two areas:

- Main area (128 KB): code and data accessible via AHB and APB.
- Information area (4 KB): data accessible via APB. Sector A (1 KB) contains factory data: ID and trim's.

Without options, the flashloader works with the main area (128 KB).

Options:

- `"--info_memory"` - works with the information area without sector A (3 KB). Programming and erasing of sector A are ignored.
- `"--info_memory_factory"` - works with the entire information area (4 KB).
- `"--erase_all"` - erase operation involves two areas:
  - main area (128 KB)
  - information area without sector A (3 KB).
- `"--erase_all --info_memory_factory"` - full erase operation involves two areas:
  - main area (128 KB)
  - information area (4 KB).

5.2.1.3. K1901VC1T

Flashloader for K1901VC1T (K1901VC1T). The microcontroller's internal flash memory consists of two areas:

- Main area (128 KB): code and data accessible via AHB and APB.
- Information area (4 KB): data accessible via APB.

Without options, the flashloader works with the main area (128 KB).

Options:

- `"--info_memory"` - works with the information area (4 KB).
- `"--erase_all"` - full erase operation involves two areas:
  - main area (128 KB)
  - information area (4 KB).

5.2.1.4. K1986VE1x

Flashloader for K1986VE1T (K1986VE1T), K1986VE1AT (K1986VE1AT). The microcontroller's internal flash memory consists of two areas:

- Main area (128 KB): code and data accessible via AHB and APB.
- Information area (4 KB): data accessible via APB.

Without options, the flashloader works with the main area (128 KB).

Options:

- `"--info_memory"` - works with the information area (4 KB).
- `"--erase_all"` - full erase operation involves two areas:
  - main area (128 KB)
  - information area (4 KB).

5.2.1.5. K1986VE3T

Flashloader for K1986VE3T (K1986VE3T). The microcontroller's internal flash memory consists of two areas:

- Main area (128 KB): code and data accessible via AHB and APB.
- Information area (4 KB): data accessible via APB.

Without options, the flashloader works with the main area (128 KB).

Options:

- `"--info_memory"` - works with the information area (4 KB).
- `"--erase_all"` - full erase operation involves two areas:
  - main area (128 KB)
  - information area (4 KB).

5.2.1.6. K1986VE4U

Flashloader for K1986VE4U (K1986VE4U). The microcontroller's internal flash memory consists of two areas:

- Main area (128 KB): code and data accessible via AHB and APB.
- Information area (8 KB): bootloader and data accessible via AHB and APB.

The default bootloader is located in block 0 (2 KB) of the information area. Without options, the flashloader works with the main area (128 KB).

Options:

- `"--info_memory"` - works with the information area without block 0 (6 KB). Programming and erasing of block 0 (bootloader) are ignored.
- `"--info_memory_boot"` - works with the entire information area (8 KB).
- `"--erase_all"` - erase operation involves two areas:
  - main area (128 KB)
  - information area without block 0 (6 KB).
- `"--restore_boot"` - restores the bootloader in block 0 when performing the erase-only operation.
- `"--erase_all --info_memory_boot"` - full erase operation involves two areas:
  - main area (128 KB)
  - information area (8 KB).

The main area, 4 blocks * 32 KB: 0x0000_0000-0x0001_FFFF.

The information area, 4 blocks * 2 KB (addressing with gaps):

- block 0: 0x0000_0000-0x0000_0800
- block 1: 0x0000_8000-0x0000_8800
- block 2: 0x0001_0000-0x0001_0800
- block 3: 0x0001_8000-0x0001_8800

5.2.1.7. K1986VE9x

Flashloader for K1986VE91T (K1986VE91T), K1986VE92U(1) (K1986VE92U(1)), K1986VE93U (K1986VE93U), K1986VE94T/F/YA (K1986VE94T/F/YA). The microcontroller’s internal flash memory consists of two areas:

- Main area (128 KB): code and data accessible via AHB and APB.
- Information area (4 KB): data accessible via APB.

Without options, the flashloader works with the main area (128 KB).

Options:

- `"--info_memory"` - works with the information area (4 KB).
- `"--erase_all"` - full erase operation involves two areas:
  - main area (128 KB)
  - information area (4 KB).

5.2.1.8. K1986VK01x

Flashloader for K1986VK018 (K1986VK018), K1986VK016 (K1986VK016). Additional options are not implemented.

5.2.2. Plastic Package Microcontrollers

5.2.2.1. K1986VE1xI

Flashloader for K1986VE1FI (MDR1213FI), K1986VE1GI (MDR1213GI). The microcontroller's internal flash memory consists of two areas:

- Main area (128 KB): code and data with AHB and APB access.
- Information area (4 KB): data with APB access. Sector A (1 KB) contains factory data: ID and trim's.

Without options, the flashloader works with the main area (128 KB).

Options:

- `"--info_memory"` - works with the information area without sector A (3 KB). Programming and erasing of sector A are ignored.
- `"--erase_all"` - erase operation involves two areas:
  - main area (128 KB)
  - information area without sector A (3 KB).

5.2.2.2. K1986VE9xI

Flashloader for K1986VE92F(1)I (MDR1211F(1)I), K1986VE94GI (MDR1209GI). The microcontroller's internal flash memory consists of two areas:

- Main area (128 KB): code and data with AHB and APB access.
- Information area (4 KB): data with APB access. Sector A (1 KB) contains factory data: ID and trim's.

Without options, the flashloader works with the main area (128 KB).

Options:

- `"--info_memory"` - works with the information area without sector A (3 KB). Programming and erasing of sector A are ignored.
- `"--erase_all"` - erase operation involves two areas:
  - main area (128 KB)
  - information area without sector A (3 KB).

5.2.2.3. K1901VC1QI

Flashloader for K1901VC1QI (MDR32FG16S1QI). The microcontroller's internal flash memory consists of two areas:

- Main area (128 KB): code and data accessible via AHB and APB.
- Information area (4 KB): data accessible via APB.

Without options, the flashloader works with the main area (128 KB).

Options:

- `"--info_memory"` - works with the information area (4 KB).
- `"--erase_all"` - full erase operation involves two areas:
  - main area (128 KB)
  - information area (4 KB).

5.2.2.4. K1986VE1QI

Flashloader for K1986VE1QI (MDR32F1QI). The microcontroller's internal flash memory consists of two areas:

- Main area (128 KB): code and data accessible via AHB and APB.
- Information area (4 KB): data accessible via APB.

Without options, the flashloader works with the main area (128 KB).

Options:

- `"--info_memory"` - works with the information area (4 KB).
- `"--erase_all"` - full erase operation involves two areas:
  - main area (128 KB)
  - information area (4 KB).

5.2.2.5. K1986VE92QI

Flashloader for K1986VE92QI (MDR32F9Q2I). The microcontroller's internal flash memory consists of two areas:

- Main area (128 KB): code and data accessible via AHB and APB.
- Information area (4 KB): data accessible via APB.

Without options, the flashloader works with the main area (128 KB).

Options:

- `"--info_memory"` - works with the information area (4 KB).
- `"--erase_all"` - full erase operation involves two areas:
  - main area (128 KB)
  - information area (4 KB).

5.2.2.6. K1986VK01GI

Flashloader for K1986VK01GI (MDR1201GI). Additional options are not implemented.

5.2.2.7. K1986VK214

Flashloader for K1986VK214 (MDR32F23QI). The microcontroller's internal flash memory consists of two areas:

- Main area (64 KB): code and data accessible via AHB and APB.
- Information area (4 KB): bootloader and data accessible via AHB and APB.

The default bootloader is located in block 0 (2 KB) of the information area. Without options, the flashloader works with the main area (64 KB).

Options:

- `"--info_memory"` - works with the information area without block 0 (2 KB). Programming and erasing of block 0 (bootloader) are ignored.
- `"--info_memory_boot"` - works with the entire information area (4 KB).
- `"--erase_all"` - erase operation involves two areas:
  - main area (64 KB)
  - information area without block 0 (2 KB).
- `"--restore_boot"` - restores the bootloader in block 0 when performing the erase-only operation.
- `"--erase_all --info_memory_boot"` - full erase operation involves two areas:
  - main area (64 KB)
  - information area (4 KB).

The main area, 2 blocks * 32 KB: 0x0000_0000-0x0000_FFFF.

The information area, 2 blocks * 2 KB (addressing with gaps):

- block 0: 0x0000_0000-0x0000_0800
- block 1: 0x0000_8000-0x0000_8800

5.2.2.8. K1986VK234

Flashloader for K1986VK234 (MDR32F21QI). The microcontroller's internal flash memory consists of two areas:

- Main area (128 KB): code and data accessible via AHB and APB.
- Information area (8 KB): bootloader and data accessible via AHB and APB.

The default bootloader is located in block 0 (2 KB) of the information area. Without options, the flashloader works with the main area (128 KB).

Options:

- `"--info_memory"` - works with the information area without block 0 (6 KB). Programming and erasing of block 0 (bootloader) are ignored.
- `"--info_memory_boot"` - works with the entire information area (8 KB).
- `"--erase_all"` - erase operation involves two areas:
  - main area (128 KB)
  - information area without block 0 (6 KB).
- `"--restore_boot"` - restores the bootloader in block 0 when performing the erase-only operation.
- `"--erase_all --info_memory_boot"` - full erase operation involves two areas:
  - main area (128 KB)
  - information area (8 KB).

The main area, 4 blocks * 32 KB: 0x0000_0000-0x0001_FFFF.

The information area, 4 blocks * 2 KB (addressing with gaps):

- block 0: 0x0000_0000-0x0000_0800
- block 1: 0x0000_8000-0x0000_8800
- block 2: 0x0001_0000-0x0001_0800
- block 3: 0x0001_8000-0x0001_8800

6. CRC Calculation Algorithm

When opening a file, verifying and reading the data, the CRC of data is displayed. The CRC calculation algorithm is shown in list. 1.

Listing 1. CRC calculation algorithm

```
uint16_t calcCrc16( uint8_t *data, uint32_t size, uint16_t crc )
{
    while (size--)
    {
        int i;
        uint8_t byte = *data++;

        for (i = 0; i < 8; ++i)
        {
            uint32_t osum = crc;
            crc <<= 1;
            if (byte & 0x80)
                crc |= 1;
            if (osum & 0x8000)
                crc ^= 0x1021;
            byte <<= 1;
        }
    }
    return crc;
}
```

The data CRC is calculated as presented in list. 2 to provide compatibility with the Crc16 function from the bootloader.

Listing 2. Data CRC calculation

```
uint8_t zero[2] = { 0, 0 };
uint16_t crc = 0;

crc = calcCrc16( someData, someDataSize, crc );
crc = calcCrc16( zero, 2, crc );
```

7. Working with Script Files

There is a possibility to perform the JS-code from the file before writing, erasing or reading of the memory. You need to add a tag to a *.flash file, as shown in list. 3.

Listing 3. Including a script in the .flash file

```
<flash_device>
  ...
  <js>script.js</js>
  ...
</flash_device>
```

The global variable “event” was added to provide the ability to determine the current event. An example which demonstrates possible states of the “event” variable is shown in list. 4.

Listing 4. Determining the event state

```
if( event == "save" )
{
    // script is called before read to file procedure
}
else if( event == "program" )
{
    // script is called before write to memory procedure
}
else if( event == "erase" )
{
    // script is called before erase procedure
}
```

Currently, the following functions are available:

- `void dap.showMessage("Text")` – shows a message in the program status field;
- `unsigned int dap.readMemory32( unsigned int adr )` – reads memory at a given address;
- `bool dap.writeMemory32( unsigned int adr, unsigned int val )` – writes data to the memory at a given address. Returns “true” if writing was successful;
- `unsigned int dap.readDpReg( unsigned int reg )` – reads the register Debug port;
- `bool dap.writeDpReg( unsigned int reg, unsigned int val )` – writes to the Debug port register. Returns “true” if writing was successful;
- `unsigned int dap.readApReg( unsigned int reg )` – reads the Access port register;
- `bool dap.writeApReg( unsigned int reg, unsigned int val )` – writes to the Access port register. Returns “true” if writing was successful.

An example of a script for a microcontroller with a Cortex-M3 core is shown in list. 5.

Listing 5. Example script for a microcontroller with a Cortex-M3 core

```
var idcode, cpuid, dhcsr, result;

idcode = dap.readDpReg( 0x00 ); // Read IDCODE (0x00 - DP_IDCODE)
dap.showMessage("IDCODE = 0x" + idcode.toString(16).toUpperCase());

cpuid = dap.readMemory32( 0xE000ED00 ); // Read CPUID (0xE000ED00)
dap.showMessage("CPUID = 0x" + cpuid.toString(16).toUpperCase());

dhcsr = dap.readMemory32( 0xE000EDF0 ); // Read DHCSR (0xE000EDF0)
if( (dhcsr & 0x20000) === 0 ) // Check the S_HALT bit
{
    dap.showMessage("Core is running");
}
else
{
    dap.showMessage("Core is halted");
}

result = "Success";
```

8. Troubleshooting

If there is an error notification or a failure of the utility, a log containing the diagnostic information can be created – launch the program with the “-d” option. A log file named logFile.txt will be created in the program directory.

If you find an error in this document or a problem in the CMSIS-DAP software, please report it to us (support@milandr.ru) and we will try to assist you as soon as possible.

9. Manual versions

№	Date	Version	Summary	Description
1.	30.03.2022	1.0	Document was created.	-
2.	14.07.2022	1.1	**Added:** • the links to the figures and tables; • the descriptions of the loaders. **Changed**: • figures.	Pages 14-16.
3.	07.08.2026	1.2.0	**Added:** • description of the CMSIS-DAP v2 protocol; • system requirements section; • description of the microcontroller package type switch when selecting a loader; • description of the light/dark UI theme. **Changed:** • the debugger firmware update section; • the utility settings section; • the loader descriptions.	Follow the text.
4.	04.09.2026	1.2.1	**Changed:** • the note on the HEX file requirements (not applicable to software version 2.0.0 and later); • the wording about the core stop when reading core registers.	Follow the text.
5.	15.09.2026	1.2.2	**Changed:** • title of the document; • headings for sections 5, 8 and item 5.1; • description of the LED indication of the debugger with v1 and v2 firmware; • description of the UART loader; • the reference to the “IAR FlashLoaderGuide” document; • removal of the CMSIS-DAP download link; • typos fixed.	Follow the text.