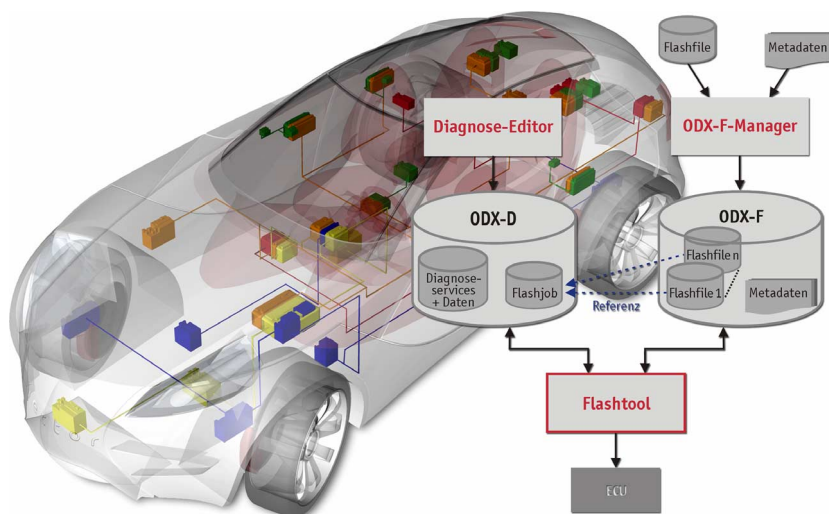


Flexible Flash Solution for Every Job

Open standards enable use of generic tool chains



Reprogramming of modern flash chips has become a commonplace process in development and production. In practice, the jobs are exceptionally diverse – depending on the ECU, department, manufacturer and supplier – so preparation, management and the flash process itself all involve considerable effort. Therefore, this article first surveys the purely technical terrain in which flash solutions occur. Afterwards, the perspective is expanded to cover process-oriented jobs and rational approaches to solutions.

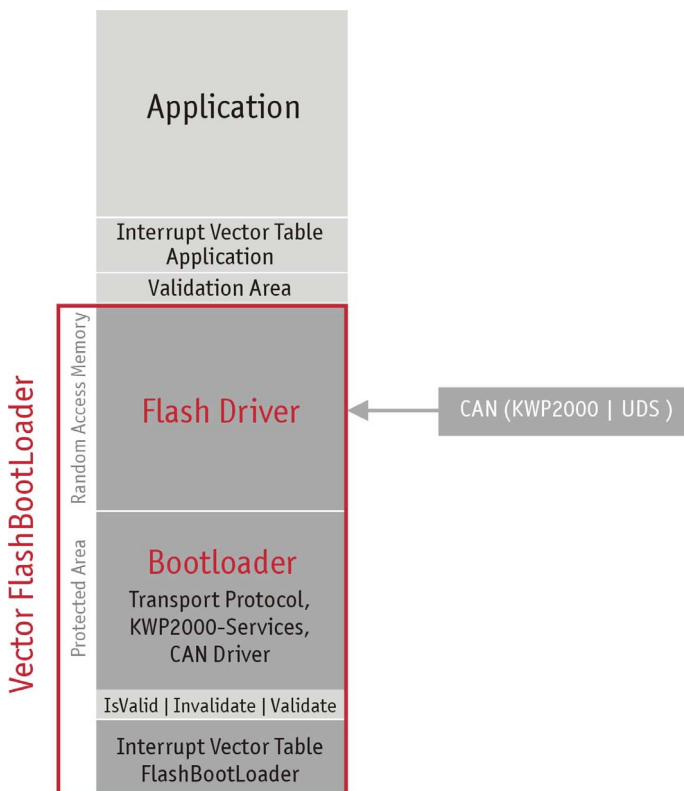
Memory fundamentals

Wherever information needs to be saved for a certain period of time, one finds rewritable flash memory in use today, e.g. in USB sticks, digital cameras and mobile telephones. Long-term storage of the firmware of microcontroller-based ECUs in the automobile is a frequent application area for flash memory. The market offers various flash architectures as NAND or NOR flash, which differ in terms of control logic, access speed, current consumption and life. Common to all types is that the memory needs to be erased before rewriting. Although data can be read-out byte-by-byte, the erasing process – in contrast to conventional

EEPROM memory – can only be performed block-wise. In ECUs, NOR flash is used almost exclusively.

Technical flashing

“Technical flashing” essentially refers to a process for updating program code and/or data in ECUs. This may be done, for example, using a PC or laptop with special software - the so-called flash tool - via the existing network infrastructure, e.g. the CAN serial bus system (Figure 1). Requirements differ, sometimes significantly, for flashing during the development phase, end-of-line programming or software updates at service centers. While software changes in the development phase always involve providing new code or new data to the same ECU, what matters in production-related flashing is to use – from a wide variety of released software levels – those that are correct for the specific vehicle and its features. In this case, it is essential to consider the vehicle variant, installed features, country code, serial number, safety aspects, etc.



[Figure 1: Memory structure in ECU with FlashBootLoader from Vector.]

At a minimum, the following components are needed for flashing:

- Flash data,
- Flash tool,
- Flash bootloader, which is integrated in the ECU and executes the actual erase/write processes
- The necessary meta information.

The flash data contain the program code and specific parameter sets intended for the ECU. The flash tool implements the flash routine on the PC side. If the tool only supports a special flash routine, then dedicated flash tools may be needed for different flash jobs. Data-driven, multipurpose flash tools are efficient and more modern. Flash job control enables their use for different ECUs and flash routines, e.g. if the same ECU is to be used at different OEMs. A single dedicated tool is all that is needed to manage flash jobs and associated flash data in containers, and it always has the same user interface.

Responsible for the routine on the ECU side is the flash bootloader (FBL) that is integrated in the ECU and is always active. The FBL is partitioned into the permanent bootloader existing in the ECU and the flash driver. For security reasons, the flash driver is only temporarily loaded in the ECU's RAM, since among other things it contains the service for initial erasure of the flash memory, acceptance of data from the flash tool and execution of the subsequent reset. The bootloader consists of the CAN driver, transport protocol and diagnostic layer and thereby ensures that the ECU always remains addressable, even if the flash process could not be completed properly.

Process-oriented flashing

A special challenge in flashing is to always load the right application in an ECU. That is the only way to ensure complete and error-free functionality in the vehicle. In production-related flashing, meta-data are required for this, such as

ECU and vehicle variant, software serial number, name of the flash job and much more. Data container files, which are referred to here as flash containers, are appropriate for consistent management of this information together with the flash data and flash jobs. ODX (Open Diagnostic data eXchange) is available as a standardized format, wherein a distinction is made between ODX-D for diagnostic-relevant information and ODX-F for flash-specific data.

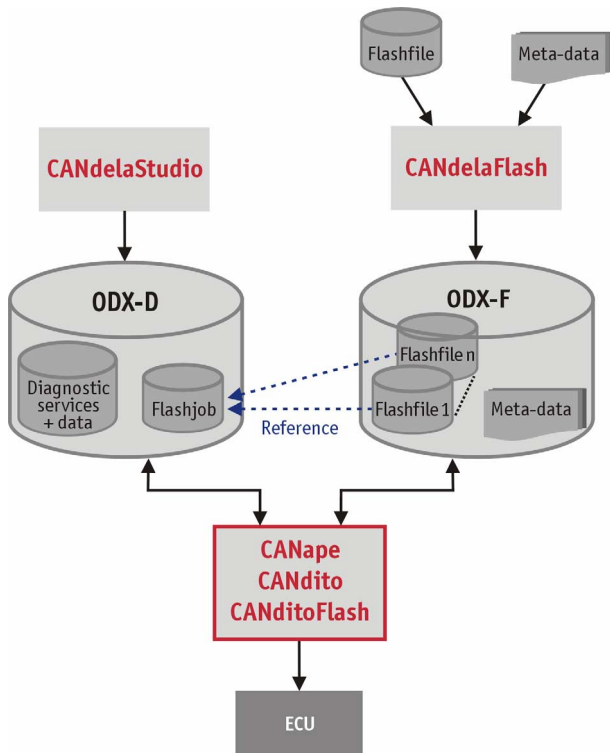
Since meta information is generally unavailable in development, flashing is often very different during development and during production. So ECU developers sometimes flash – not via the diagnostic protocol, but via CCP (CAN Calibration Protocol) or XCP (Universal Measurement and Calibration Protocol).

Rational generation and management by tools

To generate the executable flashable binary files via compiler/linker, one needs the source code for the application, diagnostic code and embedded code elements, which are supplemented by checksum information, fingerprints and other data. Use of a comprehensive product solution quickly leads to the desired results – e.g. one from Vector Informatik with source code for the embedded portion in the ECU, consisting of operating system, network management, CAN drivers, CCP/XCP protocols, etc. A generator tool is responsible for data configuration and generation of the header files.

If one utilizes a suitable authoring tool to conveniently generate diagnostic data and services, the tool can be used to directly parameterize the diagnostic tester and generate the relevant diagnostic code for the ECU. An ODX-D data container stores the diagnostic data and flash job together. There is an ODX-F management and authoring tool for linking flash data, jobs and meta-data. The associated communication parameters are not contained in the ODX-D

container, rather in a dedicated ODX-C container ('C' stands for "Communication"), Figure 2.



[Figure 2: Based on the ODX-F container, the flash routine is executed fully automatically with CANape, CANdito and CANditoFlash. The container contains all necessary data and information such as reference to the flash file, flash job and meta information.]

Getting “a grip” on references

Later in the flash process the system reads from the ODX-F file the flash files, meta information and reference to the flash job being used. The latter is found in the diagnostic description, from which it is loaded and executed. References are used to produce the relationships between the ODX-F, ODX-D and ODX-C containers.

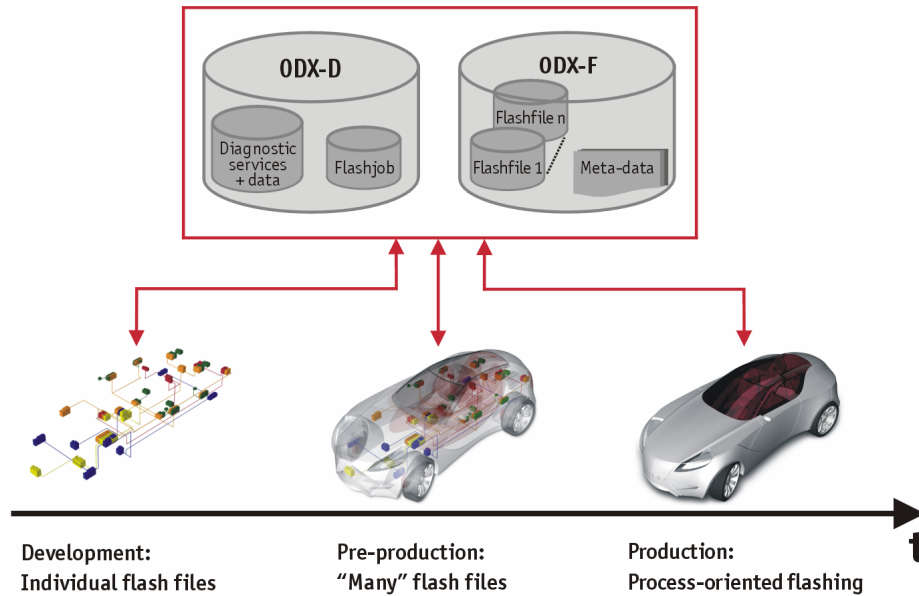
On the one hand, references to flash data and flash jobs are easy to replace without requiring regeneration of the ODX files, but they are also associated with the risk that the process will no longer function if a referenced document is missing. One solution to this problem was to create “meta” containers that save

all associated data in a single, compressed file. By definition, the containers – with the 'PDX' (Packaged ODX) extension - do not permit any references to files outside of the container.

This concept ensures that one always loads the right flash data with associated meta information into the ECU by the proper flash routine. Furthermore, it permits both fully-automatic and semi-automatic flashing. Fully automatic means that the user does not perform any configuration or selection activities. In semi-automatic flashing the user can influence the flash process.

Summary

The use of standards such as ODX enables the use of tool chains that ensure reliable and process-conformant flash processes in every development phase (Figure 3). Generic and data-driven tools benefit the user by offering enhanced flexibility with the goal of covering a wide variety of requirements and job tasks related to flashing. Flash routines cannot be implemented without suitable tool chains. From Vector the ECU developer gets universal tool support in all aspects of flashing: CANdelaStudio for specifying diagnostic requirements and generating the CDD, ODX-D and ODX-C files, CANdelaFlash for generating the ODX-F flash containers and the flash bootloader for a wide range of controllers. As authoring and development tools, CANape and CANDito are used to generate script-based flash routines, and they also form the actual runtime environment for the flash process. CANDitoFlash serves as a pure runtime environment for flash processes generated with CANape and CANDito.



[Figure 3: Use of the ODX-Standards enables a smooth transition in flash processes from development to production.]

Revised: 7/2007

Word count: 1,257

Character count: 8,130

Figures:

Opening graphic 0

Figure 1: Memory structure in ECU with flash bootloader from Vector.

Figure 2: Based on the ODX-F container, the flash routine is executed fully automatically with CANape, CANDito and CANDitoFlash. The container contains all necessary data and information such as reference to the flash file, flash job and meta information.

Figure 3: Use of the ODX-Standards enables a smooth transition in flash processes from development to production.

All figures: Vector Informatik GmbH

Author:



Andreas Patzer (Graduate engineer) is employed at Vector Informatik GmbH as Business Development Manager for the "Measurement and Calibration" product line.
Tel. +49-711/80670-562, Fax +49-711/80670-111,
E-mail: andreas.patzer@vector-informatik.de

Vector Informatik GmbH
Ingersheimer Str. 24
D-70499 Stuttgart Germany
www.vector-informatik.de

Editorial contact: Holger Heit
Tel. +49-711/80670-567, Fax +49-711/80670-555,
E-mail: holger.heit@vector-informatik.de