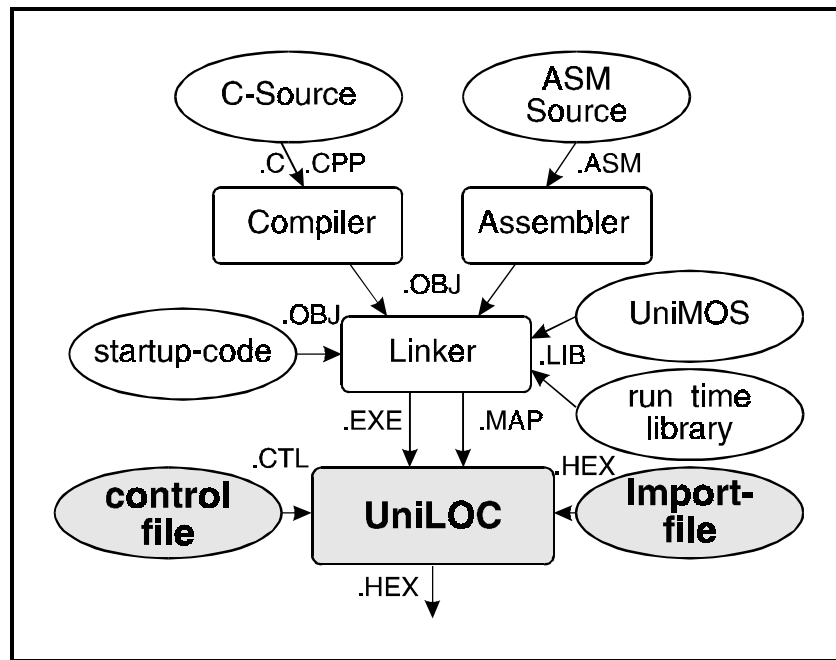


UniLOC



UniLOC V2.3

Features:

- Convert EXE file to HEX file replacing relative addresses with absolute addresses.
- Save initialized data in ROM and copy it into RAM at startup.
- Initialize segments at startup.
- Import other HEX-files.
- Output absolute file in standard hex or extended hex format.
- Print absolute memory map to screen and file.

New in V2.2

- Improved warnings and error messages.
- Proper return code issued.
- Files with empty relocation table supported.
- Absolute output file includes public symbols.
- Startup code now supports floating point emulation library.
- INIT and COPY have been modified.

New in V2.3

- Supports Motorola S-records (16-, 24- and 32-bit).
 - Verify relocation against memory map.
 - Specify the size of an output line.
 - Show/set the „magic word“
 - Native OS/2 version included.
- see README.TXT for a detailed explanation.

UniLOC V2.3 - Description

UniLOC is used to convert an .EXE to a .HEX file in *Hex* or *Extended Hex* format. Such a file can be downloaded to any standard EPROM programmer and emulator. A relocation is also applied since the .EXE file uses relocatable addresses whereas those of a .HEX file are absolute. A standard map file, as generated by the Borland or Microsoft Linkers, is needed in order to perform the relocation.

A few difficulties arise when relocating .EXE files because the file structure was defined for the DOS environment, which is rarely implemented in a controller application. First of all one has to deal with different kinds of memories (RAM or ROM) at arbitrary and non-contiguous addresses. This is different in a PC under DOS, where the .EXE file is loaded in one chunk into a free RAM space, the relocatable addresses are updated and execution can begin. Any variables are automatically initialised but updatable, because it is RAM space. In a controller application, constant data and the program code must be stored in ROM and variable data in RAM. For variables an additional step is needed during the initialization phase of the controller in order to copy the initial values into the RAM locations.

Any .EXE program, which was written for an MS-DOS PC, consists of one or more SEGMENTS, regardless of the programming language. Each segment has a maximum size of 64 kB, and a minimum size of zero for a dummy segment. Segments can be grouped, so that data within such a GROUP can be accessed as if it were be just one segment. Such groups are also limited to 64 kB in length. Classes of segments can be defined, so that one or more segments belong to one CLASS. The size of such a class is only limited by the CPU's 1 MB address space.

UniLOC assigns physical addresses to classes. This is the easiest way for the user because there can be a lot of segments, especially when the large or huge memory models are used, which all belong to the same class. All segments of this class can be located to a physical address with just one command. If, however, control over the position of one segment is required, then a class can be defined, which holds just this one segment.

The .EXE file holds a "*relocation table*", which describes those entries throughout the .EXE file, that have to be relocated. The segments to be relocated are identified by their addresses. In order that segment start addresses remain unique for UniLOC (UniLOC uses the start addresses to identify segments), they should be aligned on different paragraph (16-byte) boundaries, otherwise

separate short segments may have the same start address. This potential "aliasing" effect arises from the CPU addressing architecture, whereby the minimum space between consecutive segments is 16 bytes. Furthermore, where groups have been defined, the classes within the group (which by group definition may not exceed a total size of 64 kB) must be located contiguously and in the same order as shown in the .MAP file. Most compilers implicitly define a DGROUP which must be kept contiguous.

The following precautions apply:

- 1.) **prevent "aliasing" by starting each segment on it's own paragraph.**
- 2.) **keep the segments of a group together.**

Since UniLOC needs only the .MAP and the .EXE files, it does not have or need any information about the programming language which you use. Any language is fine as long as the proper .MAP and .EXE formats are generated. High level languages normally need a proper register and environment (stack) setup, which is not available in an embedded control system. This setup is done by the startup code from the respective compiler, which is, however, heavily dependent on the operating system (MS-DOS). The source of the startup code is shipped together with the compiler. It is the user's responsibility to adapt the source code to the target environment.

All operating system dependent function calls must be either removed from the startup code or emulated by in-built functions. An additional piece of code must be added which copies initialized data from the ROM area into the variables in the RAM area. Using UniLOC's *SAVE* command (described below), a copy of the initialized data class can be written into the ROM while the same number of bytes are reserved in the RAM space. Initialisation can thus be performed at startup time simply by copying the data from ROM to RAM.

A minimum startup code, shipped together with UniLOC, can be used with Borland C++, version 2.0 or 3.1. This also contains the code to copy data from ROM into RAM and to initialize RAM, usually with zeros. This startup code is a sample file for your own developments.

Like the program UniLOC itself, it is shipped free of charge and without any warranty!

UniLOC is started by the following command line:

uniloc ctlfilename <listfilename> /parms

With this command, UniLOC executes the instructions from the named control file; the control filename is a mandatory parameter. The file name of a list file is optional. This list file contains the absolute memory map as well as any warnings. The warning level can be selected by the command line parameter "/W". Four different warning levels can be specified. Each warning level displays additional information on the screen and it includes all information from the previous level. The following warning levels are supported:

/W0 only prints fatal errors, which stop the execution of UniLOC.

/W1 additionally prints warnings, which may have an impact on the usability of the generated output file.

/W2 additionally prints some useful information. This is the default warning level.

/W3 issues a full report of UniLOC's activities in addition to all warnings/errors generated. This may be used as a "problem report" in the event of any questions regarding UniLOC.

A help text is printed when the parameter "/h" or "/?" is specified.

"/m" prints the "magic string", which is the data used in the startup file to find the tables, that describe the actions for INIT and SAVE. There is in fact no magic behind. Just make sure that UniLOC can find this string at the proper place in the startup file, so that INIT and SAVE can be handled.

"/m=xx...xx" sets a new magic string. Note that 20 hexadecimal digits are required. It is not necessary to set the magic string unless you have a specific startup code with one that differs from the default.

"/n=nnn" sets the number of data elements in one line of data in the output file. The maximum value is 252. The transfer speed for the output file to the programmer or emulator is increased with a higher value for n, but note that your target device may impose a constraint on the maximum length of one record.

The control file holds the instructions for UniLOC, which are executed sequentially. A sample control file may built-up as follows:

```
; Test File for UniLOC
; Locate TEST.C and generate Hex file.
;
Mapfile Test.Map
```

```
Infile Test.Exe
Outfile Testl.Hex for 0xf000 to 0xf7ff
Outfile Testh.Hex for 0xf800 to 0xffff offset 0x8000
Ram from 0x00000 to 0x1ffff
Rom from 0xe0000 to 0xfffff
Save DATA in SAVE_DATA
Init BSS STACK with 0h
Import LCA_CLASS from lca.mcs offset 2 savelength
Locate DATA BSS STACK to 0040h
Locate CODE LCA_CLASS SAVE_DATA to 0xf000
Locate RESET to 0xffff
Output CODE LCA_CLASS SAVE_DATA RESET as Hex
```

Comment lines start with a semicolon in the first column of a line. They are ignored completely. All other lines contain UniLOC instructions, which all start with a predefined command name. Instructions may be specified over several lines; any line which does not start with an UniLOC command is interpreted as a continuation line.

UniLOC treats the .EXE file on Class basis. A class consists of one or more segments and can have a size of up to 1 MB, while a segment or a group (of segments) must not be longer than 64 kB. The UniLOC instructions need the class names as parameters. Most instructions do not just support single classes; a list of classes can be supplied and the respective instruction is then applied sequentially to all classes in the list.

Explanation of the UniLOC instructions:

Infile filename
Mapfile filename

The file name of the input .EXE file and the file name of the .MAP file are specified with these instructions. The header and the relocation table of the .EXE file are read into memory when the "infile" instruction is executed. The memory map is read into memory upon execution of the "mapfile" instruction. Both instructions must precede any other UniLOC instruction and each one may only be specified once.

Outfile filename <offset offs>

Extended hex and Mots24/32 file formats only!

This command specifies the file name for the output file. Note that only one outfile command per control file is permitted. All output code and data is written into this file. A 16-bit offset may optionally be specified.

Outfile filename for start to end <offset offs>

Standard hex and Mots16 file format only!

This command specifies the file name for the output file. Note that this command is only valid for standard HEX and Mots16 file output. Such files have a maximum size of 64 kB and so this command allows you to specify an address range (start to end) for the code/data to be output.

An arbitrary offset may be specified for the output file by using the "offset" keyword. By default, the offset address in the output hex file is calculated relative to the specified start address. Therefore, data at the absolute address F123:0 would have the offset 1230 in the output file, when 0xF000 was specified as the start address and no offset keyword was specified. It might be more convenient in some cases to have an offset address of 9230 for this data. In that case the optional parameter "offset 0x8000" may be appended to the above outfile command.

Note that the number of "outfile" commands is limited to 16 per control file.

Import class from file <offset offs> <savelength>

With this instruction, UniLOC imports a standard hex file and moves it to the specified class. The class must already exist (create a dummy class in one of the source files) but it may be shorter than the file to be imported. The size is calculated by UniLOC from the length of the imported file, so that the complete file including the optional offset fits into the class.

The length of the included file must be less than 64 kB and no gaps must occur in the file. No relocation is performed to the imported data. The "savelength" keyword instructs UniLOC to store the number of imported bytes in the first word of the specified class. The "offset" keyword allows to move the data relative to the specified class. "Offset 2" is normally specified together with the "savelength" keyword. This prevents the first two imported bytes from being overwritten.

The "import" command has been implemented to import configuration data for Xilinx LCAs. It is by no means a general purpose import feature.

Save class in newclass

This command creates a duplicate of the classes specified as the first parameter. A new class name must be used for the second parameter. All subsequent commands can refer to this new name as if it were a real class as found in the map file. The SAVE command is used to create a backup copy of initialized data. This copy is stored in ROM and copied back to the RAM locations by the startup code.

Init <list of classes> with data

The Init command instructs the startup code to initialize the specified classes with the databyte. This command is normally used to initialize the non-initialized data areas and the stack with zero.

Locate <list of classes> to address

Locate assigns the specified absolute address to the first class in the list of classes. The next class

in the list will start where the previous one has ended. Alignment of each segment is treated properly.

Output <list of classes> as type

The output command writes the data to the output file using the specified type. Supported types are *HEX*, *EXTENDEDHEX*, *MOTS16*, *MOTS24* and *MOTS32*. Specification of the type is not case sensitive. *Hex* and *Mots16* use 16-bit addresses and therefore the size of such files is limited to 64 kB code/data. This implies that the program start address and segment addresses cannot be inserted into the output file. *Extended Hex* uses a segment/offset format and it can therefore cover the whole 1 MB address range. *Mots24* uses a 24-bit address to cover 16 MB while *Mots32* applies 32-bit addresses for a 4 GB address range. UniLOC always places the code/data into the lower 1 MB. Please see the README file for details of the formats. The list of classes holds only those classes that have to be written into ROM.

The absolute map file

The absolute memory map is printed on the screen and optionally written into the listfile. As it is generated from the linker map file, it is very similar to it. It lists all segments and classes that UniLOC has handled, along with their start and end addresses and the length. The "W" column is only present if one of the RAM/ROM statements has been used to define a memory configuration of the target system. This column shows the warnings due to conflicts between the memory map and the current segment type and location. The "Out" column indicates whether or not this segment has been written to an output file. The "Refs" column displays the number of references found to this segment. If multiple segments have been relocated to the same address, then UniLOC cannot distinguish, to which of the segments the reference applies. It displays the number of references only for the first segment at this address and print a "." for all other segments.

Any additional data from the linker map file is also attached to the list file after updating all expressions of the form <4-digit hex number>:<4-digit hex number>, assuming they were relative addresses, to absolute addresses. This does normally work, but there is one exception: There may be (dummy) segments that contain no bytes at all and no entry in the relocation table refers to them. Such segments need not be touched by UniLOC and so there may be no "LOCATE" instruction for them. Consequently they do not appear in the output file (they are not really in the input file either) and therefore UniLOC cannot assign an absolute address to them, printing a "?????" instead.