Turbo ParTool

By Atlas

February 8, 2026

1. Exporting a .par file using PARex

PARex is a program used to export a `.par` file from **KnightShift** into a directory of `.cpp` files, which represent each section of objects in the `.par` file.

How to use:

1. We are turning on the program.
2. Enter the name of the input file along with its format.
3. Click the enter button and wait.
4. When the console window closes, we know that the program has finished running.

The program also works in ARGC&ARGV mode. Example of use:

```
PARex.exe <par_file_name.par>
```

Quick Export:

To quickly decompile a `.par` file, use `EXPORT.ps1` or `EXPORT.bat`. To modify arguments/configuration, edit the `.bat/.ps1` file using Notepad.

Example of a .bat file:

```
PARex.exe <filename.par>
```

Example of a .ps1 file:

```
.\PARex.exe .\<filename.par>
```

2. Optional PARex configuration

The **PARex.cfg** file allows you to configure how data is exported to `.cpp` files. You can decide whether the selected values will be presented as individual bits (flags) with specific labels or in raw format. If flag mapping is disabled, the data will be saved as `uint32_t`. Disabling flagging significantly speeds up the export process and reduces the compilation time of the generated code. The following parameters are available in the configuration file:

True:

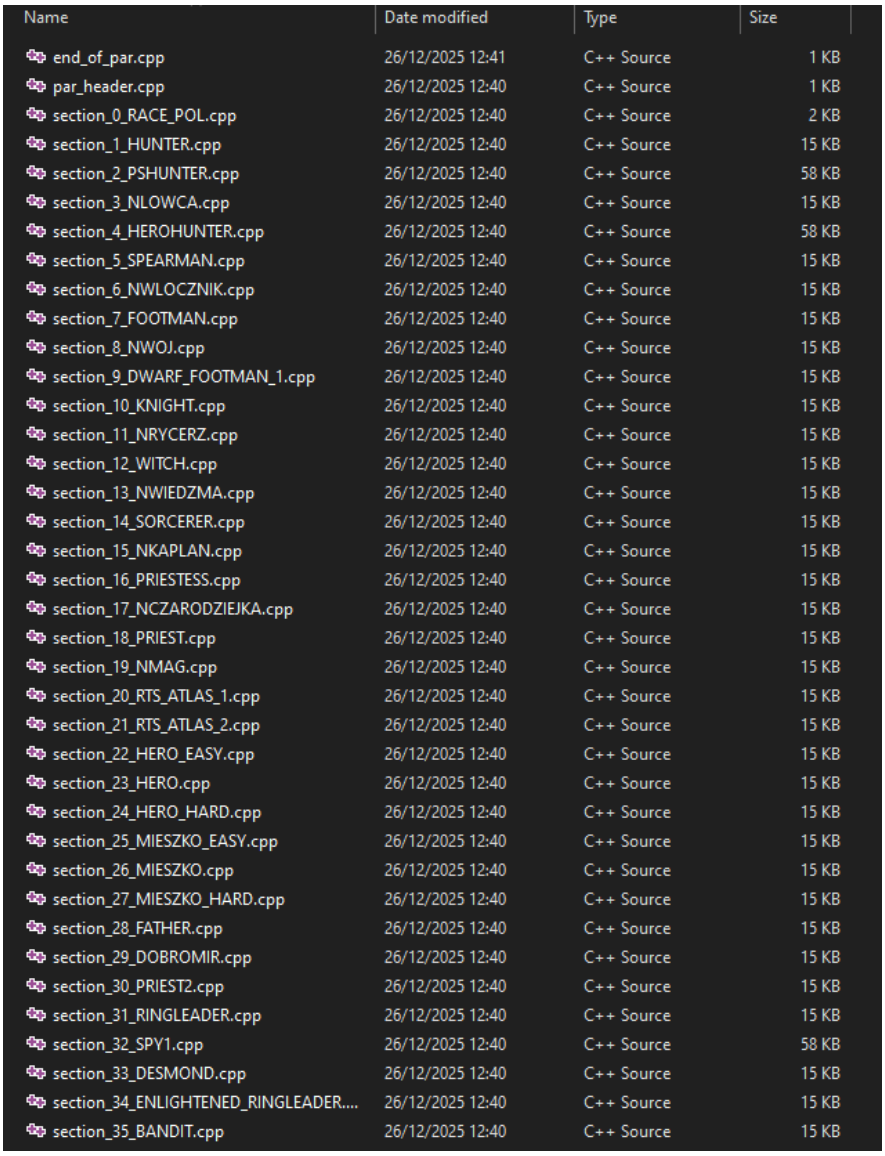
- YES/Yes/yes
- TRUE/True/true
- 1

False:

- NO/No/no
- FALSE/False/false
- 0

3. General overview of the structure of files obtained after export/decompilation

After exporting the `.par` file, we will receive a directory containing `.cpp` files representing sections with objects. The syntax of the files is in `C++`. In addition to the `.cpp` files with sections, we also receive a `section_order.txt` file, which contains the order in which the section files are compiled.



Name	Date modified	Type	Size
end_of_par.cpp	26/12/2025 12:41	C++ Source	1 KB
par_header.cpp	26/12/2025 12:40	C++ Source	1 KB
section_0_RACE_POL.cpp	26/12/2025 12:40	C++ Source	2 KB
section_1_HUNTER.cpp	26/12/2025 12:40	C++ Source	15 KB
section_2_PSHUNTER.cpp	26/12/2025 12:40	C++ Source	58 KB
section_3_NLOWCA.cpp	26/12/2025 12:40	C++ Source	15 KB
section_4_HEROHUNTER.cpp	26/12/2025 12:40	C++ Source	58 KB
section_5_SPEARMAN.cpp	26/12/2025 12:40	C++ Source	15 KB
section_6_NWLOCZNIK.cpp	26/12/2025 12:40	C++ Source	15 KB
section_7_FOOTMAN.cpp	26/12/2025 12:40	C++ Source	15 KB
section_8_NWOJ.cpp	26/12/2025 12:40	C++ Source	15 KB
section_9_DWARF_FOOTMAN_1.cpp	26/12/2025 12:40	C++ Source	15 KB
section_10_KNIGHT.cpp	26/12/2025 12:40	C++ Source	15 KB
section_11_NRYCERZ.cpp	26/12/2025 12:40	C++ Source	15 KB
section_12_WITCH.cpp	26/12/2025 12:40	C++ Source	15 KB
section_13_NWIEDZMA.cpp	26/12/2025 12:40	C++ Source	15 KB
section_14_SORCERER.cpp	26/12/2025 12:40	C++ Source	15 KB
section_15_NKAPLAN.cpp	26/12/2025 12:40	C++ Source	15 KB
section_16_PRIESTESS.cpp	26/12/2025 12:40	C++ Source	15 KB
section_17_NCZARODZIEJKA.cpp	26/12/2025 12:40	C++ Source	15 KB
section_18_PRIEST.cpp	26/12/2025 12:40	C++ Source	15 KB
section_19_NMAG.cpp	26/12/2025 12:40	C++ Source	15 KB
section_20_RTS_ATLAS_1.cpp	26/12/2025 12:40	C++ Source	15 KB
section_21_RTS_ATLAS_2.cpp	26/12/2025 12:40	C++ Source	15 KB
section_22_HERO_EASY.cpp	26/12/2025 12:40	C++ Source	15 KB
section_23_HERO.cpp	26/12/2025 12:40	C++ Source	15 KB
section_24_HERO_HARD.cpp	26/12/2025 12:40	C++ Source	15 KB
section_25_MIESZKO_EASY.cpp	26/12/2025 12:40	C++ Source	15 KB
section_26_MIESZKO.cpp	26/12/2025 12:40	C++ Source	15 KB
section_27_MIESZKO_HARD.cpp	26/12/2025 12:40	C++ Source	15 KB
section_28_FATHER.cpp	26/12/2025 12:40	C++ Source	15 KB
section_29_DOBROMIR.cpp	26/12/2025 12:40	C++ Source	15 KB
section_30_PRIEST2.cpp	26/12/2025 12:40	C++ Source	15 KB
section_31_RINGLEADER.cpp	26/12/2025 12:40	C++ Source	15 KB
section_32_SPY1.cpp	26/12/2025 12:40	C++ Source	58 KB
section_33_DESMOND.cpp	26/12/2025 12:40	C++ Source	15 KB
section_34_ENLIGHTENED_RINGLEADER....	26/12/2025 12:40	C++ Source	15 KB
section_35_BANDIT.cpp	26/12/2025 12:40	C++ Source	15 KB

Figure 1: List of .cpp files after export.

4. Recommended IDE for editing .cpp files

The recommended program for editing `.cpp` files is **geany**. In addition to the ability to edit source code, it also has a very intuitive preview of data structures. The data structure preview is represented by a tree, whose leaves represent the corresponding fields in the data structure. Double-clicking on a field name takes the user to the line with the selected field.

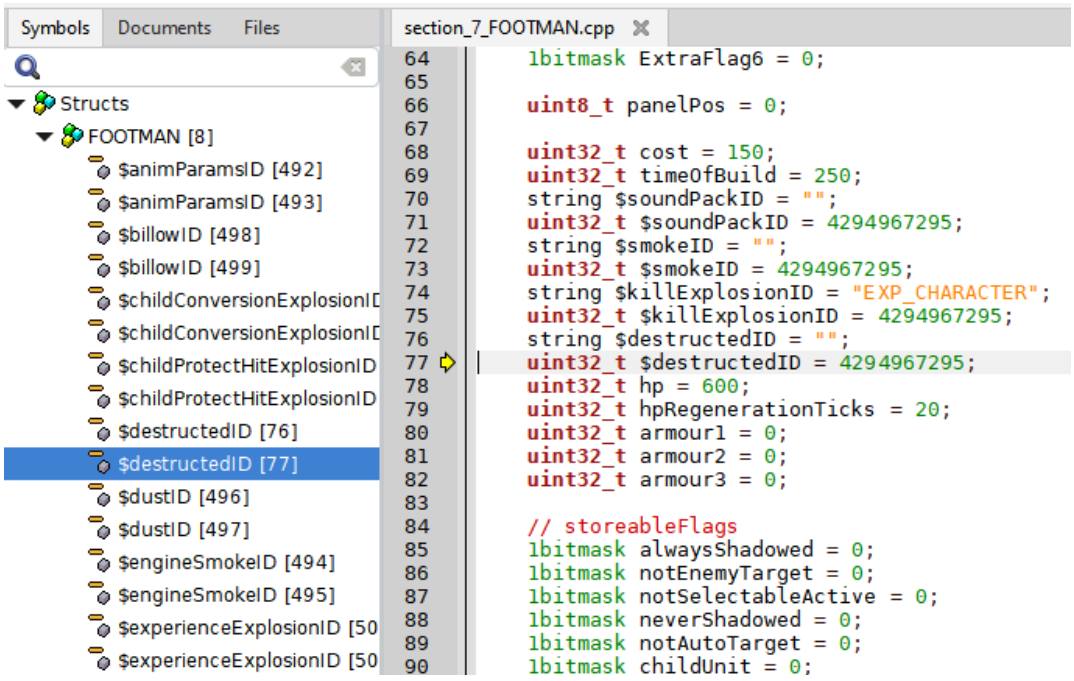


Figure 2: Side panel *Symbols* displaying a list of variables with references to specific lines of code.

5. Comments

Comments can be written in section files using the `//` symbols. Example of use:

```
...
uint32_t armour2 = 0;
uint32_t armour3 = 0;

// storeableFlags                <-- This line will be omitted.
// Ala has a cat                <-- This line will also be omitted.
// CODE WRITTEN IN SLAVIC   <-- This one too.
1bitmask alwaysShadowed = 0;
...
```

6. Data types

1. **bool** - takes up literally one byte and can only store logical values: **false, 0, true, 1**.

   Example of use:

   ```
   bool logic = 0;
   bool logic2 = true;
   ```

2. **char** - takes up literally one byte and can only store a value in the form of a symbol.

   Example of use:

   ```
   char symbol = 'a';
   ```

3. **uint8_t** - It occupies literally one byte and can only store a value in the form of a non-negative number ranging **from 0 to 255**.

   Example of use:

   ```
   uint8_t format_0 = 80;
   ```

4. **uint16_t** - it takes up literally two bytes and can only store a value in the form of a non-negative number ranging **from 0 to 65 535**.

   Example of use:

   ```
   uint16_t panelPos = 2;
   ```

5. **uint32_t** - it occupies literally four bytes and can only store a value in the form of a non-negative number ranging **from 0 to 4 294 967 295**.

   Example of use:

   ```
   uint32_t listTemplateNum = 340;
   ```

6. **uint64_t** - it occupies literally eight bytes and can only store a value in the form of a non-negative number ranging **from 0 to 18 446 744 073 709 551 615**.

   Example of use:

   ```
   uint64_t number_of_sections = 1280;
   ```

7. **int8_t** - it occupies literally one byte and can only store values in the range **from -128 to 127**.
   Example of use:

   ```
   int8_t test = -13;
   ```

8. **int16_t** - it takes up literally two bytes and can only store values ranging from **-32 768 to 32 767**.

   Example of use:

```
int16_t test2 = -1236;
```

9. **int32_t** - takes up literally four bytes and can only store values in the range from **–2 147 483 648 to 2 147 483 648.**

   Example of use:

   ```
   int32_t mapSign = -1;
   ```

10. **int64_t** - it occupies literally eight bytes and can only store values in the form of numbers ranging from **–9 223 372 036 854 775 808 to 9 223 372 036 854 775 807**.

    Example of use:

    ```
    int64_t num = -221280;
    ```

11. **bool[]** - an array of boolean values storing **n** boolean values. It is not necessary to specify the size of the array in square brackets `[]`, as the compiler will check its size anyway. Square brackets are required to emphasize that the type is an array.

    Example of use:

    ```
    bool classID_array[] = {0,1,1,1,1,0,0,0,0,0,0,0,0};
    ```

12. **char[]** - a symbol table occupying **n** bytes and can only store text enclosed in quotation marks. It is not necessary to specify the size of the table in square brackets `[]`, as the compiler will check its size anyway. Square brackets are required to emphasize that the type is an array.

    Example of use:

    ```
    char header_string[header_string_length] = "translateGameParams";
    ```

13. **uint8_t[]** - an array of `uint8_t` values storing **n** values of type `uint8_t`. It is not necessary to specify the size of the array in square brackets `[]`, as the compiler will check its size anyway. Square brackets are required to emphasize that the type is an array.

    Example of use:

    ```
    uint8_t test_array[] = {64,1,3,1,23,0,128};
    ```

14. **uint16_t[]** - an array of `uint16_t` values storing **n** values of type `uint16_t`. Writing the size of the array in square brackets `[]` is not required because the compiler will check its size anyway. Square brackets are required to emphasize that the type is an array.

    Example of use:

    ```
    uint16_t test_array_2[] = {64,503,346,22};
    ```

15. **uint32_t[]** - an array of `uint32_t` values storing **n** values of type `uint32_t`. It is not necessary to specify the size of the array in square brackets `[]`, as the compiler will check its size anyway. Square brackets are required to emphasize that the type is an array.

    Example of use:

    ```
    uint32_t test_array_3[] = {2264,503,55346,322,0};
    ```

16. **uint64_t[]** - an array of `uint64_t` values storing **n** values of type `uint64_t`. It is not necessary to specify the size of the array in square brackets `[]`, as the compiler will check its size anyway. Square brackets are required to emphasize that the type is an array.

    Example of use:

    ```
    uint64_t test_array_4[] = {226433,5034};
    ```

17. **int8_t[]** - an `int8_t` array storing **n** `int8_t` values. It is not necessary to specify the size of the array in square brackets `[]`, as the compiler will check its size anyway. Square brackets are required to emphasize that the type is an array.

    Example of use:

    ```
    int8_t test_array[] = {-64,1,3,-1,-23,0};
    ```

18. **int16_t[]** - an array of `int16_t` values storing **n** `int16_t` values. It is not necessary to specify the size of the array in square brackets `[]`, as the compiler will check its size anyway. Square brackets are required to emphasize that the type is an array.

    Example of use:

    ```
    int16_t test_array_2[] = {-64,503,-346,22};
    ```

19. **int32_t[]** - an array of `int32_t` values storing **n** `int32_t` values. It is not necessary to specify the size of the array in square brackets `[]`, as the compiler will check its size anyway. Square brackets are required to emphasize that the type is an array.

    Example of use:

```
int32_t test_array_3[] = {-2264,503,55346,-322,0};
```

20. **int64_t[]** - an array of `int64_t` values storing **n** `int64_t` values. It is not necessary to specify the size of the array in square brackets `[]`, as the compiler will check its size anyway. Square brackets are required to emphasize that the type is an array.

    Example of use:

    ```
    int64_t test_array_4[] = {-2264,503,55346,-322,0};
    ```

21. **string** - a character string occupying a maximum of **32 767 bytes** and can only store text enclosed in quotation marks. The compiler automatically calculates the size of such a character string and saves it in four bytes before the string in the binary file.

    Example of use:

    ```
    string mesh = "CHARACTERS\FOOTMAN#nfootman.tex";
    ```

22. **GUID** - a globally unique identifier, i.e., an identifier for a specific object in Windows or anywhere else where a unique identifier is needed. It occupies exactly **16 bytes**. In **KnightShift**, it appears in many different binary files, e.g., in maps (LND/MIS), meshes (MSH), or parameter files (PAR).

    Example:

    ```
    GUID header_guid = {FF863ECE-6C69-468A-B0BC-D0244A73316C};
    ```

23. **struct** - a structure is a specific space that can contain certain values of different data types. We use struct when we create a new object in a section or when we create a new section header. The compiler counts the number of occurrences of the word struct in the file to calculate the number of objects in a single section.

    Example of use:

    ```
    struct Section_0_RACE_POL_Header
    {
     32bituniquekey race = POL;
     uint32_t flag_0 = 10;
     uint32_t number_of_objects = count();
    };
    ```

24. **1bitmask** - it occupies literally one bit and can only store a value in the form of a logical **0 or 1**. This data type is used to represent object flags. Eight 1bitmask values make up one byte.

    Example of use:

```
// shadowType

// First byte
1bitmask _shadowSimple = 0;
1bitmask _shadowStatic = 0;
1bitmask _shadowAnimated = 1;
1bitmask _shadowVertical = 0;
1bitmask unknown1 = 0;
1bitmask unknown2 = 0;
1bitmask unknown3 = 0;
1bitmask unknown4 = 0;

// Second byte
1bitmask _moveWind = 0;
1bitmask _noReflect = 0;
1bitmask unknown5 = 0;
1bitmask _hasNotBigEnoughBox = 0;
1bitmask _noSubObjectShadow = 0;
1bitmask _symmetricShadow = 0;
1bitmask unknown6 = 0;
1bitmask unknown7 = 0;

// Third byte
1bitmask _shadowOldStatic = 0;
1bitmask _shadowOldRound = 1;
1bitmask _shadowOldEllipse1 = 0;
1bitmask _shadowOldEllipse2 = 0;
1bitmask _shadowOldEllipse3 = 0;
1bitmask _shadowOldTree = 0;
1bitmask _hasOldSoftwareRendering = 0;
1bitmask ExtraFlag0 = 0;

// Fourth byte
uint8_t shadowType = 0;
```

Another example:

```
// third byte of panelPos
1bitmask tab1 = 0;
1bitmask ExtraFlag0 = 0;
1bitmask ExtraFlag1 = 0;
1bitmask ExtraFlag2 = 0;
1bitmask ExtraFlag3 = 0;
1bitmask ExtraFlag4 = 0;
1bitmask ExtraFlag5 = 0;
```

```
1bitmask ExtraFlag6 = 0;
```

25. **8bitmask** - it occupies 8 bits, i.e. one byte. When entering a value for this data type, we enter constants whose values are presented in the cheat sheet. These values can be used in OR (logical sum) operations, which can be used to combine flags. `8bitmask` is used exclusively in the representation of values labeled **passiveMask**.

    Example of use:

    ```
    // first byte of passiveMask
    8bitmask passiveMask = (_artefactPassive_ | _mapOtherPassive_);
    ```

26. **32bitmask** - it occupies 32 bits, or four bytes. When entering a value for this data type, we enter constants whose values are presented in the cheat sheet. These values can be used in OR (logical sum) operations, which can be used to combine flags.

    Example of use:

    ```
    // magicAnimType
    32bitmask magicAnimType =
     (equipmentAnimTypeMagic2 | magicTalk2 | magicTalkNone);
    ```

27. **32bituniquekey** - it occupies 32 bits, or four bytes. Only one constant can be entered for this data type. This data type only supports constants, so it cannot be directly assigned a numerical value. If you want to use a numerical value, the type must be changed to `uint32_t` or `int32_t`. The constants that can be entered for this type are listed in the cheat sheet.

    Example of use:

    ```
    32bituniquekey standType = standAccurate;
    ```

7. Functions

The ParIm compiler also supports the following functions:

- **uint32_t count();** - a function that counts the number of objects in a section. The function works by counting the number of occurrences of struct in the section file, and then using this information to calculate and return the number of objects in the section. The function will work correctly if the field is named `number_of_objects`.

  Example of use:

  ```
  uint32_t number_of_objects = count();
  ```

- **uint64_t count();** - a function that counts the number of all sections. The function works by counting the number of occurrences of files in the order file, and then using this information to calculate and return the total number of sections. The function will work correctly if the field is named `number_of_sections`.

  Example of use:

  ```
  uint64_t number_of_sections = count();
  ```

8. Values associated with types

For some types of data, we can distinguish constants that are associated with corresponding values.

8.1. 8bitmask

1. passiveMask

Name of the constant	Value in HEX
mapOtherPassive	0x00
bridgePassive	0x01
pontoonBridgePassive	0x02
singleBridgePassive	0x03
singlePontoonBridgePassive	0x04
bridgeRuinPassive	0x05
pontoonBridgeRuinPassive	0x06
artefactPassive	0x07
tunnelEntrancePassive	0x08
healthPlacePassive	0x09
conversionPlacePassive	0x0A
teleportPassive	0x0B
birdPassive	0x0C
waterAnimalPassive	0x0D
mapNothingPassive	0x10
mapBuildingPassive	0x20
mapRockPassive	0x30
mapTreePassive	0x40
mapWallPassive	0x50
mapEditorPassive	0x60

8.2. 32bitmask

1. objectType

Name of the constant	Value in HEX
animNone	0x00000000
animWalk	0x00000001
animRotor	0x00000002
animRotation	0x00000002
moveLand	0x00000000
moveAmphibia	0x00000100
moveShip	0x00000200
moveFlyable	0x00000300
notMoveable	0x00010000

2. magicAnimType

Name of the constant	Value in HEX
equipmentAnimTypeNone	0x00000000
equipmentAnimTypeFight	0x00000001
equipmentAnimTypeFight2	0x00000002
equipmentAnimTypeShoot	0x00000003
equipmentAnimTypeMagic1	0x00000004
equipmentAnimTypeMagic2	0x00000005
equipmentAnimTypeMagic3	0x00000006
equipmentAnimTypeMagic4	0x00000007
magicTalkNone	0x00000000
magicTalk1	0x00010000
magicTalk2	0x00020000
magicTalk3	0x00030000
magicTalk4	0x00040000

8.3. 32bituniquekey

1. magicType

Name of the constant	Value in HEX
NULL	0x00000000
magicImmortalShield	0x00000001
magicFreeze	0x00000002
magicCapturing	0x00000003
magicStorm	0x00000004
magicSeeing	0x00000005
magicConversion	0x00000006
magicFireRain	0x00000007
magicRemoveStormFireRain	0x00000008
magicTeleportation	0x00000009
magicGhost	0x0000000A
magicWildAnimal	0x0000000B
magicTrap	0x0000000C
magicGetHP	0x0000000D
magicSingleFreeze	0x0000000E
magicBlindAttack	0x0000000F
magicTimedCapturing	0x00000010
magicOurWildAnimal	0x00000011
magicOurHoldWildAnimal	0x00000012
magicOurMagicMirror	0x00000013
magicRandConversion	0x00000014
magicAroundDamage	0x00000015
magicSelfHealing	0x00000016
magicFireWall	0x00000017

2. buildingType

Name of the constant	**Value in HEX**
buildingNormal	0x00000000
buildingFactory	0x00000001
buildingHarvestFactory	0x00000002
buildingGate	0x00000003
buildingBridgeGate	0x00000004
buildingTower	0x00000005
buildingWall	0x00000006
buildingCopula	0x00000007

3. trapType

Name of the constant	**Value in HEX**
NULL	0x00000000
typeHoldTrap	0x00000001
typeDamageTrap	0x00000002
typeHoldTrapOnce	0x00000003

4. wasteSize

Nazwa stałej	**Value in HEX**
smallFlyingWaste	0x00000000
mediumFlyingWaste	0x00000001
bigFlyingWaste	0x00000002

5. groupTemplateNum

Name of the constant	Value in HEX
groupSword	0x0000005B
groupDrop	0x0000005C
groupMag	0x0000005D
groupSpecial	0x0000005E
groupAnimal	0x0000005F
groupMulti	0x00000060
groupBuilding	0x00000061

6. positionType

Name of the constant	Value in HEX
positionStartingPoint	0x00000000
positionMarkPoint	0x00000001
positionProductionPoint	0x00000002

7. explosionFlags

Name of the constant	Value in HEX
noEarthquake	0x00000000
smallEarthquake	0x00000001
mediumEarthquake	0x00000002
bigEarthquake	0x00000003

8. raceFlags

Name of the constant	Value in HEX
eCanBePlayedInSkirmish	0x00000001

9. standType

Name of the constant	Value in HEX
standNone	0x00000000
standAccurate	0x00000001
standVertical	0x00000002
standCoarsly	0x00000003
standSwing	0x00000004
standWater	0x00000005
standMoveDownSmall	0x00000010
standMoveDownMedium	0x00000020
standMoveDownBig	0x00000030
standTurn	0x00000040
standTurnToFlat	0x00000080
standMoveSmall	0x00000100
standMoveMedium	0x00000200
standMoveBig	0x00000300
standWaterPlant	0x00000345
standTree	0x00000340
standTreeFall	0x00000380
standRock	0x00000150
standStone	0x00000140
standFish	0x00000045

10. experienceExplosionPos

Name of the constant	Value in HEX
expPosZero	0x00000000

11. equipmentFlags

Name of the constant	Value in HEX
shieldArmourType	0x00000001
maxArmourType	0x00000002

12. hitType

Name of the constant	Value in HEX
singleHit	0x00000000
multiHit	0x00000001

13. type (missile)

Name of the constant	Value in HEX
NULL	0x00000000
missileSword	0x00000001
missileInvisible	0x00000002
missileCannon	0x00000003
missileDropBomb	0x00000004
missileBomb	0x00000005
missileElectric	0x00000006
missileLightning	0x00000007
missileMeteor	0x00000008

14. race

Name of the constant	Value in HEX
NEUTRAL	0x00000000
POL	0x00000001

9. Adding a new object

To add a new object, copy the structure (struct) representing the object you want to model on and paste it, for example, immediately after the object you are copying.

```
// WE COPY THE WOLF STRUCTURE PRESENTED BELOW...
struct WOLF
{
 string obj_name = "WOLF";
 uint32_t research_switch = 1;
 string research = "RES_NOT_FOR_BUILD";
 bool classID_array[] = {0,1,1, ... ,1,0,1,0,1,0,0,0,1,0};
 uint32_t classID = 3146001;
 string mesh = "CHARACTERS\WOLF";
 string lowResMesh = "";
 string lowRes2Mesh = "";

 ...

 uint32_t scriptParams = 0;
 string $childProtectHitExplosionID = "";
 uint32_t $childProtectHitExplosionID = 4294967295;
 string $childConversionExplosionID = "";
 uint32_t $childConversionExplosionID = 4294967295;
};

// AND FOR EXAMPLE, WE PASTED IT HERE...

struct DOG
{
 string obj_name = "DOG";
 uint32_t research_switch = 1;
 string research = "RES_NOT_FOR_BUILD";
 bool classID_array[] = {0,1,1,1, ... ,0,0,0,0,0,0,0,1,0,1,0};
 uint32_t classID = 3146001;
 string mesh = "CHARACTERS\LABRADOR#Dog.tex";
 string lowResMesh = "";
 string lowRes2Mesh = "";
 string interfaceMesh = "";

 ...

 string $childProtectHitExplosionID = "";
 uint32_t $childProtectHitExplosionID = 4294967295;
```

```
 string $childConversionExplosionID = "";
 uint32_t $childConversionExplosionID = 4294967295;
};
```

After performing the operation, we have the following view:

```
struct WOLF
{
 string obj_name = "WOLF";
 uint32_t research_switch = 1;
 string research = "RES_NOT_FOR_BUILD";
 bool classID_array[] = {0,1,1, ... ,1,0,1,0,1,0,0,0,1,0};
 uint32_t classID = 3146001;
 string mesh = "CHARACTERS\WOLF";
 string lowResMesh = "";
 string lowRes2Mesh = "";

 ...

 uint32_t scriptParams = 0;
 string $childProtectHitExplosionID = "";
 uint32_t $childProtectHitExplosionID = 4294967295;
 string $childConversionExplosionID = "";
 uint32_t $childConversionExplosionID = 4294967295;
};

struct WOLF
{
 string obj_name = "WOLF";
 uint32_t research_switch = 1;
 string research = "RES_NOT_FOR_BUILD";
 bool classID_array[] = {0,1,1, ... ,1,0,1,0,1,0,0,0,1,0};
 uint32_t classID = 3146001;
 string mesh = "CHARACTERS\WOLF";
 string lowResMesh = "";
 string lowRes2Mesh = "";

 ...

 uint32_t scriptParams = 0;
 string $childProtectHitExplosionID = "";
 uint32_t $childProtectHitExplosionID = 4294967295;
 string $childConversionExplosionID = "";
```

```
 uint32_t $childConversionExplosionID = 4294967295;
};

struct DOG
{
 string obj_name = "DOG";
 uint32_t research_switch = 1;
 string research = "RES_NOT_FOR_BUILD";
 bool classID_array[] = {0,1,1,1, ... ,0,0,0,0,0,0,0,1,0,1,0};
 uint32_t classID = 3146001;
 string mesh = "CHARACTERS\LABRADOR#Dog.tex";
 string lowResMesh = "";
 string lowRes2Mesh = "";
 string interfaceMesh = "";

 ...

 string $childProtectHitExplosionID = "";
 uint32_t $childProtectHitExplosionID = 4294967295;
 string $childConversionExplosionID = "";
 uint32_t $childConversionExplosionID = 4294967295;
};
```

Now you need to change the structure name and object name. The locations of the fields to be changed are shown below:

```
struct WOLF // <--- Here is the name of the structure.
{
 string obj_name = "WOLF"; // <--- Here is the name of the object.
 uint32_t research_switch = 1;
 string research = "RES_NOT_FOR_BUILD";
 bool classID_array[] = {0,1,1, ... ,1,0,1,0,1,0,0,0,1,0};
 uint32_t classID = 3146001;
 string mesh = "CHARACTERS\WOLF";
 string lowResMesh = "";
 string lowRes2Mesh = "";

 ...

 uint32_t scriptParams = 0;
 string $childProtectHitExplosionID = "";
 uint32_t $childProtectHitExplosionID = 4294967295;
 string $childConversionExplosionID = "";
```

```
 uint32_t $childConversionExplosionID = 4294967295;
};
```

After making the changes, the structure code looks as follows:

```
struct SAINT_WOLF_TEST
{
 string obj_name = "SAINT_WOLF_TEST";
 uint32_t research_switch = 1;
 string research = "RES_NOT_FOR_BUILD";
 bool classID_array[] = {0,1,1, ... ,1,0,1,0,1,0,0,0,1,0};
 uint32_t classID = 3146001;
 string mesh = "CHARACTERS\WOLF";
 string lowResMesh = "";
 string lowRes2Mesh = "";

 ...

 uint32_t scriptParams = 0;
 string $childProtectHitExplosionID = "";
 uint32_t $childProtectHitExplosionID = 4294967295;
 string $childConversionExplosionID = "";
 uint32_t $childConversionExplosionID = 4294967295;
};
```

The object parameters can be modified as desired. The section file can be saved and then compiled into a PAR file. It is also worth remembering to add the new object to the `EditorDef.txt` file in order to check it in the map editor.

10. Adding a new section

To add a new section, copy the file of another section that you want to use as a template and paste it into the same directory. Then rename the file so that it does not conflict with other files.

Example:

- base file = `section_8_NWOJ.cpp`
- new section file = `section_8_NS_0_NOWA_SEKCJA.cpp`

Inside the section file:

In the section file, we should note the header structure:

```
struct Section_8_NWOJ_Header
{
 32bituniquekey race = POL;
 uint32_t flag_ = 1;
 uint32_t number_of_objects = count();
};
```

The name of the structure should be the same as the file name, but with the suffix `_Header` added at the end. The modified header looks like this:

```
struct Section_8_NS_0_NOWA_SEKCJA_Header
{
 32bituniquekey race = POL;
 uint32_t flag_ = 1;
 uint32_t number_of_objects = count();
};
```

All objects below the header should be deleted. To add new ones, paste the structures representing them into the file. Adding new objects is explained in the previous chapter.

If a new section has already been added, its name should also be added to the **section_order.txt** file, which stores the order in which sections are compiled. It is best to place the name of the new section file immediately after the name of the file on which we based it. The name in the list should be the same as the name of the new section file, but without the format.

Example of a list before adding a new section:

```
...
section_6_NWLOCZNIK
section_7_FOOTMAN
section_8_NWOJ
section_9_DWARF_FOOTMAN_1
section_10_KNIGHT
...
```

Example of a list after adding a new section:

```
...
section_6_NWLOCZNIK
section_7_FOOTMAN
section_8_NWOJ
section_8_NS_0_NOWA_SEKCJA
```

```
section_9_DWARF_FOOTMAN_1
section_10_KNIGHT
...
```

11. Compiling .cpp files into .par files using PARim

PARim is a program, or more precisely a compiler used to compile an entire directory of `.cpp` files into `.par` files.

How to use:

1. We are turning on the program.
2. Enter the name of the directory.
3. Press enter and it should be done in a moment.
4. Closing the console window signals the end of the program.

The program also works in ARGC&ARGV mode. Example of use:

```
PARim.exe <input directory name>
```

Quick Import:

To quickly compile a directory with `.cpp` files, use `IMPORT.ps1` or `IMPORT.bat`. To modify arguments/configuration, edit the `.bat/.ps1` file with Notepad.

Example of a .bat file:

```
PARim.exe <input directory name>
```

Example of a .ps1 file:

```
.\PARim.exe .\<input directory name>
```