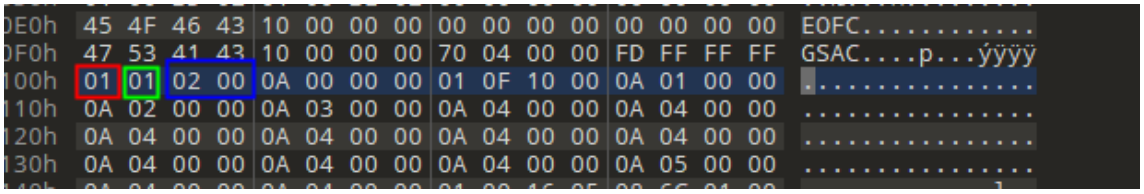
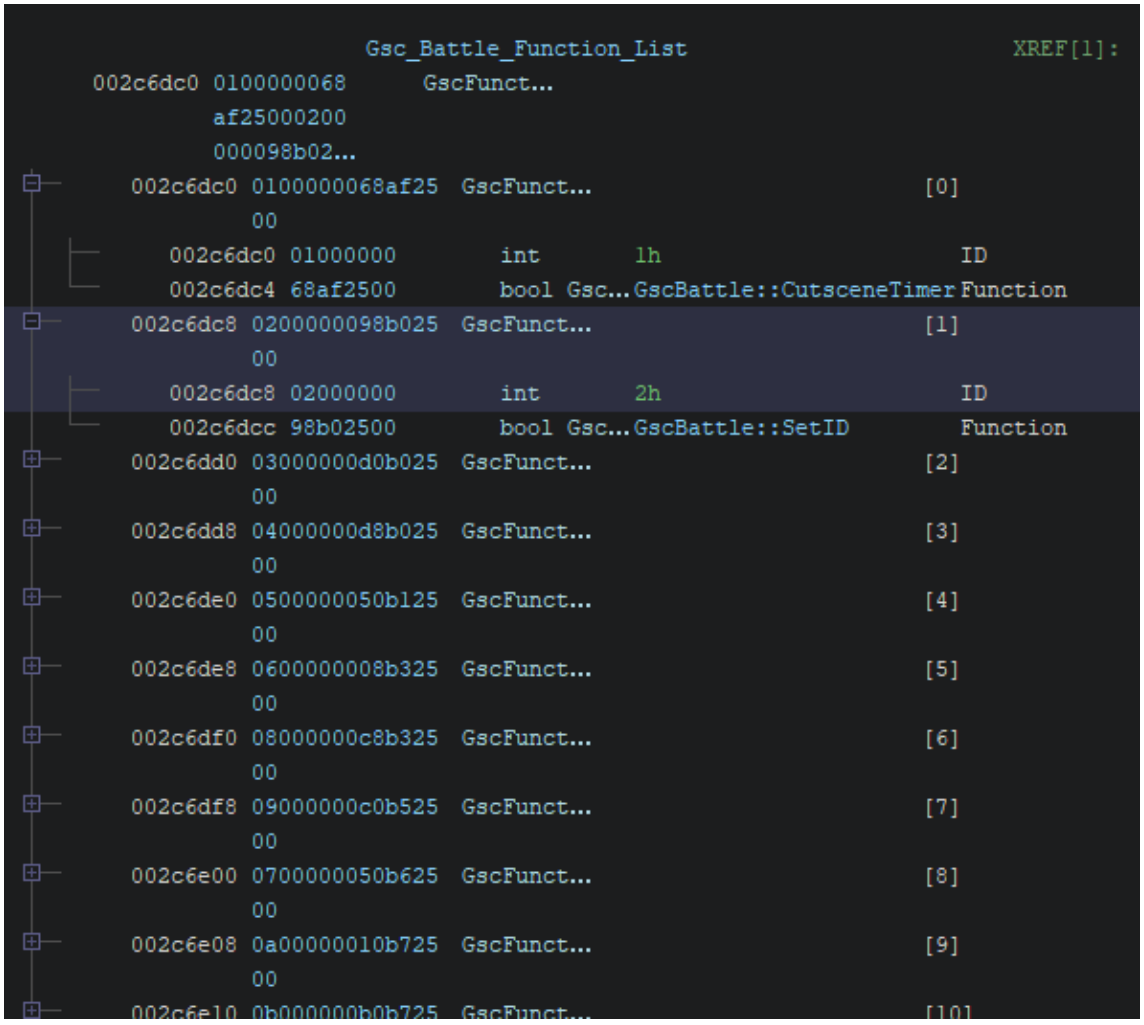
To begin with, as you know, GSC flies work as if they were a script which follows a flow like if it was a code.

For example, here we have the first instruction of a GSAC



The type of instructions is defined by the first byte, the red one. In this case, since it's 01 it indicates the call of a function. Green is the ammount of parameters needed for the function and blue is the ID of the function.

Inside the Code, we have this list



The first one is the ID, as it is the same one marked in Blue in the first Screenshot, and the second one is the function that will execute

This is the function it calls for:

```
// 0025b098

bool GscBattle::SetID(uint event,int *registers)

{
  int battleID;

  if ((event == 0) || (event == 4)) {
    battleID = GscReader::ReadIntParameter();
    GscResources::SetBattleID(battleID);
  }
  return true;
}
```

As mentioned in the first screenshot, it's using a parameter in which is the scenario ID, the parameters are those which start with 0A, 1A or 2A.

We also have the instructions which first byte isn't 01 but rather 02

```
30h  47 53 41 43 10 00 00 00 20 00 00 00 E8 03 00 00  GSAC.... ...è...
90h  02 00 FD FF 02 00 FC FF 02 00 FB FF 02 00 FE FF  .ýÿ..üÿ..ûÿ..þÿ
A0h  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
30h  45 4F 46 43 10 00 00 00 00 00 00 00 00 00 00 00  EOFC............
```

These functions are to call for another GSAC

Red: Instruction Type

Green: GSAC ID

For example, that GSAC calls for other 4 that are already in the GSAC which establish the Battle's Settings

There's also an Instruction 03 but it remains unused (not seen an instance which is used), but works very simmilar to function 02

And Last, we got the properties

```
0h  45 4F 46 43 10 00 00 00 00 00 00 00 00 00 00 00
0h  47 53 41 43 10 00 00 00 40 05 00 00 FB FF FF FF
0h  01 00 0F 00 08 76 02 00 0A 00 00 00 0A 16 00 00
0h  08 76 02 00 0A 05 00 00 0A 00 00 00 08 76 02 00
0h  0A 0C 00 00 0A 16 00 00 08 76 02 00 0A 0F 00 00
0h  0A 00 00 00 08 76 02 00 0A 18 00 00 0A 16 00 00
```

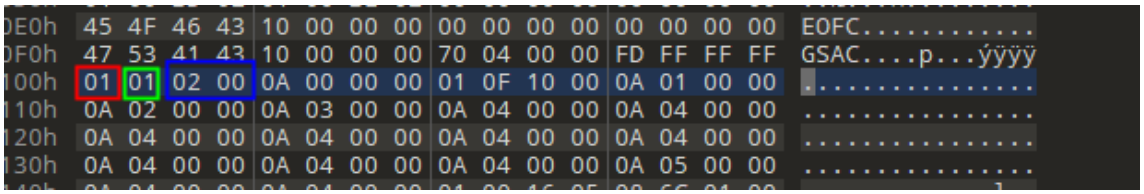
Red: Always 8

Green: ID of the property. It's always assigned an letter

Blue: Amount of Parameters

Properties usually act as parameters that can be optional and can have parameters but sometimes they only need to be there to serve it's porpuse.

Now, let's look at the list of Functions in Story Mode



DE0h	45	4F	46	43	10	00	00	00	00	00	00	00	00	00	00	00	EOFC.....
DF0h	47	53	41	43	10	00	00	00	70	04	00	00	FD	FF	FF	FF	GSAC....p...ýÿÿÿ
100h	01	01	02	00	0A	00	00	00	01	0F	10	00	0A	01	00	00	.....
110h	0A	02	00	00	0A	03	00	00	0A	04	00	00	0A	04	00	00	.....
120h	0A	04	00	00	0A	04	00	00	0A	04	00	00	0A	04	00	00	.....
130h	0A	04	00	00	0A	04	00	00	0A	04	00	00	0A	05	00	00	.....

I'll mention them acording to the ID, which is marked in Blue in here.

Function 1: Stops the flow of the execution until it reaches to a determined time

```
/ 0025af68

ool GscBattle::CutsceneTimer(uint event,int *registers)


 bool bVar1;
 int gscStageTime;
 float timer;

 bVar1 = false;
 if (event < 5) {
                    // WARNING: Switch is manually overridden
   switch(event) {
   case 2:
     gscStageTime = GscStageItem::GetCurrentItemTime();
     timer = GscReader::ReadFloatParameter();
     if (gscStageTime < (int)(timer * 300.0)) {
       return false;
     }
   case 4:
     bVar1 = true;
   }
 }
 return bVar1;
```

For example: if we assigned an animation before the Function 1 and then another one after the function, it's not going to assign the second one until the time assigned for that function is reached

In the screenshot above, we see that it's using a float parameter, so after that function we have an parameter 1A

0A: Int

1A: Float

2A: String

Since we know the ID and the amount of parameters we could even know how it would look in hex. 01 + Amount of Parameters + Function ID -> 01 01 01 00

Function 2: Simply uses an parameter to assign the ID of the scenario and with that assign the Subtitles, LPS and ADX Files

```
// 0025b098

bool GscBattle::SetID(uint event,int *registers)

{
  int battleID;

  if ((event == 0) || (event == 4)) {
    battleID = GscReader::ReadIntParameter();
    GscResources::SetBattleID(battleID);
  }
  return true;
}
```

Function 3: Does literally nothing, but it’s always the first one seen on an GSAC

```
// 0025b0d0

bool GscBattle::StartCutscene(uint event,int *registers)

{
  return true;
}
```

```
F0h | 47 53 41 43 10 00 00 00 20 02 00 00 10 27 00 00 | GSAC.... ....'..
00h | 01 00 03 00 01 01 09 00 0A 04 00 00 01 00 05 00 | ................
10h | 01 07 21 03 0A 00 00 00 1A 7F 00 00 1A 80 00 00 | ..!.........€..
20h | 1A 80 00 00 1A 80 00 00 1A 81 00 00 1A 80 00 00 | .€...€.......€..
30h | 01 07 21 03 0A 16 00 00 1A 82 00 00 1A 80 00 00 | ..!.....‚...€..
40h | 1A 80 00 00 1A 80 00 00 1A 83 00 00 1A 80 00 00 | .€...€...ƒ...€..
50h | 01 06 B1 04 1A 84 00 00 1A 85 00 00 1A 86 00 00 | ..±..„....…...†..
60h | 1A 87 00 00 1A 88 00 00 1A 80 00 00 01 00 B2 04 | .‡...ˆ...€....².
70h | 08 6C 07 00 1A 89 00 00 1A 8A 00 00 1A 8B 00 00 | .l...‰...Š...‹..
80h | 1A 8C 00 00 1A 8D 00 00 1A 8E 00 00 1A 80 00 00 | .Œ.......Ž...€..
90h | 01 02 85 03 0A 16 00 00 0A 8F 00 00 08 6C 00 00 | ..….........l..
A0h | 01 02 43 06 0A 16 00 00 0A 00 00 00 08 77 00 00 | ..C..........w..
B0h | 01 06 B1 04 1A 90 00 00 1A 91 00 00 1A 92 00 00 | ..±......‘...’..
C0h | 1A 93 00 00 1A 94 00 00 1A 80 00 00 01 00 B2 04 | .“...”...€....².
D0h | 08 6C 07 00 1A 95 00 00 1A 96 00 00 1A 97 00 00 | .l...•...–...—..
```

Function 4: Is similar to 3, but it serves a porpuse. It's always found at the end of the GSAC and it's it's porpuse is to assign again the camera of the Character and allows both Characters to Move

```
// 0025b0d8

bool GscBattle::EndCutscene(uint event,int *registers)

{
  if (((event == 0) || (event == 4)) &&
     ((GscResources::Data.field58_0x15c == (byte *)0x0 ||
      (((*GscResources::Data.field58_0x15c & 0xe) == 0 ||
       ((*GscResources::Data.field58_0x15c & 0x10) == 0)))))) {
    If_GSC_Resume_Actor_Control(0);
    If_GSC_Resume_Actor_Control(1);
    If_Reset_Battle_Camera();
  }
  return true;
}
```

```
0A 0D 00 00 01 00 0D 00 01 00 0E 00 01 02 08 00  ................
0A 05 00 00 0A 04 00 00 01 00 04 00 00 00 00 00  ................
45 4F 46 43 10 00 00 00 00 00 00 00 00 00 00 00  EOFC............
47 53 41 43 10 00 00 00 B0 01 00 00 11 27 00 00  GSAC....°....'..
```

Function 5: Establishes the preset position of the characters on the Stage

This function is the same one you'd find when starting a Battle in Duel Mode

```
/ 0025b150

ool GscBattle::SetActorsDefaultTransform(uint event,int *registers)


 Vector4 position [2];
 Vector4 rotation [2];

 if ((event == 0) || (event == 4)) {
   if (GscResources::Data.field58_0x15c != (byte *)0x0) {
     if ((*GscResources::Data.field58_0x15c & 0x10) != 0) {
       FUN_0012c988();
       Vfx::FUN_0012cbc0(2);
       Vfx::FUN_0012cbc0(3);
       GetMapActorPosition(0,position,rotation,false);
       GetMapActorPosition(1,position + 1,rotation + 1,false);
       If_GSC_Set_Actor_Position_Matrix(0,position);
       If_GSC_Set_Actor_Rotation_Matrix(0,rotation);
       If_GSC_Set_Actor_Position_Matrix(1,position + 1);
       If_GSC_Set_Actor_Rotation_Matrix(1,rotation + 1);
       If_GSC_Set_Actor_Cutscene_Animation_Data(0.0,0,0,true);
       If_GSC_Set_Actor_Cutscene_Animation_Data(0.0,1,0,true);
     }
     if (GscResources::Data.field58_0x15c != (byte *)0x0) {
       return true;
     }
   }
   GetMapActorPosition(0,position,rotation,false);
   GetMapActorPosition(1,position + 1,rotation + 1,false);
   If_GSC_Set_Actor_Position_Matrix(0,position);
   If_GSC_Set_Actor_Rotation_Matrix(0,rotation);
   If_GSC_Set_Actor_Position_Matrix(1,position + 1);
   If_GSC_Set_Actor_Rotation_Matrix(1,rotation + 1);
   If_GSC_Set_Actor_Cutscene_Animation_Data(0.0,0,0,true);
   If_GSC_Set_Actor_Cutscene_Animation_Data(0.0,1,0,true);
 }
 return true;
```

Function 6: Turns off the effects, specifically the Auras, Ki Charging Auras and Camera Shakes

```
// 0025b308

bool GscBattle::DisableAllVfx(uint event,int *registers)

{
  if ((event == 0) || (event == 4)) {
    if (GscResources::Data.field58_0x15c != (byte *)0x0) {
      if ((*GscResources::Data.field58_0x15c & 0x10) != 0) {
        If_GSC_Set_Actor_Aura_VFX_Off(0);
        If_GSC_Set_Actor_Aura_VFX_Off(1);
        GSC_Set_Actor_Charge_Aura_VFX_Off(0);
        GSC_Set_Actor_Charge_Aura_VFX_Off(1);
        If_GSC_Set_Screen_Shake(0.0);
        FUN_0012c9a0();
      }
      if (GscResources::Data.field58_0x15c != (byte *)0x0) {
        return true;
      }
    }
    If_GSC_Set_Actor_Aura_VFX_Off(0);
    If_GSC_Set_Actor_Aura_VFX_Off(1);
    GSC_Set_Actor_Charge_Aura_VFX_Off(0);
    GSC_Set_Actor_Charge_Aura_VFX_Off(1);
    If_GSC_Set_Screen_Shake(0.0);
  }
  return true;
}
```

Function 8: (it's not always in Order, 7 goes after) is the one that manages the Battle Events, such as Clashes, Beam Struggles, ect.

```
// 0025b3c8

bool GscBattle::BattleEventHandler(uint event,int *registers)

{
  uint battleEvent;
  int character;
  uint role;
  int blastID;

  if ((event != 0) && (event != 4)) {
    return true;
  }
  Uncheck_GSC_Change_Character_Flag();
  battleEvent = GscReader::ReadIntParameter();
  character = GscReader::ReadIntParameter();
  role = (int)battleEvent >> 0xf & 1;
  switch(battleEvent & 0xfff) {
  case 1:
    FUN_0012c7d0();
```

It has 2 parameters. The 1st one is the ID of the Event (and which character is applied to) and the second one is the character (used for (De)Transformations)

The first 2 despite not being named, it's know what is they're used for:

```
  case 1:
    FUN_0012c7d0();
    break;
  case 2:
    FUN_0012c7f0();
    break;
  case 3:
    GSC_Perform_Event_Map_Destruction();
    break;
  case 4:
    GSC_Perform_Event_Melee_Clash();
    break;
```

The 2 does the exact same thing, but it's used on different situations, it's like pressing the start button

Maybe you might have noticed that when when adding an event on the first GSAC, it doesn't start until you press the Start Button. That's because it's assigned to the event of pressing the Start Button (Case 1) Case 2 it's the same, but it's applied on the GSAC where the Battle Ends

List of Events:

1: Battle Start

2: Battle Finish

3: Stage Destruction (The Planet is Destroyed)

4: Clash

5, 6 and 7: Aerial Clash

8: Beam Struggle (First Blast 2 vs First Blast 2)

9: Beam Struggle (First Blast 2 vs Second Blast 2)

10: Beam Struggle (Second Blast 2 vs First Blast 2)

11: Beam Struggle (Second Blast 2 vs Second Blast 2)

12: Beam Struggle (Ultimate Blast vs Ultimate Blast)

16: Character Swap (To Slot 1)

17: Character Swap (To Slot 2)

18: Character Swap (To Slot 3)

19: Character Swap (To Slot 4)

20: Character Swap (To Slot 5)

21 and 22: Transformation

23: Fusion

24: Max Power!

25: Use First Blast 1

26: Use Second Blast 1

27: Use First Blast 2

28: Use Second Blast 2

29: Use Ultimate Blast

All these values are in Decimal, not Hex

Function 9: Indicates the end of the Battle. It has 1 parameter that indicated the result of the Battle

```
// 0025b5c0

bool GscBattle::SetEndBattleType(uint event,int *registers)

{
  bool bVar1;
  bool bVar2;
  int endType;

  bVar2 = false;
  if (event < 5) {
                    // WARNING: Switch is manually overridden
    switch(event) {
    case 0:
      *registers = 0;
      endType = GscReader::ReadIntParameter();
      if (endType < 0) {
        bVar2 = true;
      }
      else {
        EndBattle(endType);
        bVar2 = false;
      }
      break;
    case 2:
      bVar1 = FUN_0012c968();
      bVar2 = false;
      if (bVar1) {
        bVar2 = 0 < *registers;
        *registers = *registers + 1;
      }
    }
  }
  return bVar2;
```

0: You Win! (K.O.)

1: You Lose! (K.O.)

2: You Win! (Ring Out)

3: You Lose! (Ring Out)

Function 7:

```
/ 0025b650

nt GscBattle::LinksHandler(uint event,int *registers)


 bool bVar1;
 int condition;
 int id;
 int iVar2;

 if ((event == 0) || (event == 4)) {
   GscResources::FUN_00259380();
   iVar2 = 0;
   while( true ) {
     bVar1 = GscReader::FindProperty('a',iVar2);
     if (!bVar1) break;
     condition = GscReader::ReadIntParameter();
     id = GscReader::ReadIntParameter();
     GscResources::AddGscStageLink(condition,id);
     iVar2 = iVar2 + 1;
   }
   iVar2 = 0;
   while( true ) {
     bVar1 = GscReader::FindProperty('v',iVar2);
     if (!bVar1) break;
     condition = GscReader::ReadIntParameter();
     id = GscReader::ReadIntParameter();
     GscResources::AddVoiceLink(condition,id,1);
     iVar2 = iVar2 + 1;
   }
 }
 return 1;
```

Here we see the use of the properties, they don't have any parameters, but it does have properties and can have an indefinite amount of properties this is the function that assigns the links between and predetermined condition and an Audio or a GSAC

For example:

```
D90h  08 6C 07 00 1A A1 00 00 1A AC 00 00 1A AD 00 00  .l...¡...¬...-..
DA0h  1A AE 00 00 1A AF 00 00 1A B0 00 00 1A 80 00 00  .®...¯...°...€..
DB0h  01 02 43 06 0A 00 00 00 0A 0F 00 00 08 77 00 00  ..C..........w..
DC0h  01 00 06 00 01 00 07 00 08 61 02 00 0A 5D 00 00  .........a...]..
DD0h  0A B1 00 00 08 61 02 00 0A B2 00 00 0A B3 00 00  .±...a...²...³..
DE0h  08 61 02 00 0A B5 00 00 0A B4 00 00 08 76 02 00  .a...µ...´...v..
DF0h  0A B5 00 00 0A 18 00 00 08 76 02 00 0A B5 00 00  .µ.......v...µ..
E00h  0A 0D 00 00 01 00 0D 00 01 00 0E 00 01 02 08 00  ................
E10h  0A 05 00 00 0A 04 00 00 01 00 04 00 00 00 00 00  ................
```

We have 3 properties of type "a" and type "v"

Type "a" links to other GSAC while "v" links to the Audio files

Function 10: Is what assigns the Stage, Music, Time, Announcer and whatever activates Stages Destruction or not

```
// 0025b710

bool GscBattle::SetBattleSettings(uint event,int *registers)


{
 int map;
 int bgm;
 int time;
 int narrator;
 int mapDestructionFlag;

 if ((event == 0) || (event == 4)) {
   map = GscReader::ReadIntParameter();
   bgm = GscReader::ReadIntParameter();
   time = GscReader::ReadIntParameter();
   narrator = GscReader::ReadIntParameter();
   mapDestructionFlag = GscReader::ReadIntParameter();
   fun_00126ee0();
   Set_Main_Battle_Settings(false,Story_Chapter,bgm,time,narrator,map,SUB41(mapDestructionFlag,0));
 }
 return true;
}
```

Once again, all of these Ids are in Decimal, that would be the Function 0A

Function 11: Assigns the amount of Characters of each Team

```
// 0025b7b0

bool GscBattle::SetTeamSettings(uint event,int *registers)

{
  int role;
  int teamCount;
  int controller;

  if ((event == 0) || (event == 4)) {
    role = GscReader::ReadIntParameter();
    teamCount = GscReader::ReadIntParameter();
    controller = 2;
    if (role == 0) {
      controller = 0;
    }
    Set_Battle_Settings_Actor_Player_Data(role,controller,0,teamCount,false,false,0,(ulong *)0x0);
  }
  return true;
}
```

Function 12: Assigns a Character of the Team

```
// 0025b818

bool GscBattle::SetCharacterData(uint event,int *registers)

{
  ulong *puVar1;
  uint player;
  int character;
  int skin;
  int damagedFlag;
  int aiLevel;
  int health;
  uint role;
  uint teamID;
  int zItemTable [8];
  int storyLevel;

  if ((event == 0) || (event == 4)) {
    player = GscReader::ReadIntParameter();
    teamID = player & 0xfff;
    role = (int)player >> 0xf & 1;
    character = GscReader::ReadIntParameter();
    skin = GscReader::ReadIntParameter();
    damagedFlag = GscReader::ReadIntParameter();
    aiLevel = GscReader::ReadIntParameter();
    zItemTable[7] = GscReader::ReadIntParameter();
    health = GscReader::ReadIntParameter();
    zItemTable[0] = GscReader::ReadIntParameter();
    zItemTable[1] = GscReader::ReadIntParameter();
    zItemTable[2] = GscReader::ReadIntParameter();
    zItemTable[3] = GscReader::ReadIntParameter();
    zItemTable[4] = GscReader::ReadIntParameter();
    zItemTable[5] = GscReader::ReadIntParameter();
    zItemTable[6] = GscReader::ReadIntParameter();
    if (-1 < character) {
      storyLevel = (MenuDisplay::Data->HistorySettings).Level;
      if (storyLevel == 1) {
        aiLevel = aiLevel + 6;
      }
      else if (storyLevel == 2) {
        aiLevel = aiLevel + 9;
```

Parameters: Player, Character, Costume, if it's Battle Damaged, CPU level, Strategy Type Z-Item, HP and Z-Items

As you can see in the Screenshot, in Normal Mode you add 6 to the Level, while on Hard Mode, you add 9

Function 13: ….Doesn’t have parameters, only properties and all optional

All properties have 1 parameter

Function 13 is to establish data to a character on the player’s team

List of the Properties:

C: ID of the character on the team that will apply all this data (if it’s not set, then it will be assigned to the current one).

h: Assign HP.

H: Add HP.

I: Assign Ki.

F: Add Ki.

b: Assign Blast Stocks.

B: Same as F “Add Ki” (It’s meant to be “Add Blast Stocks” but someone at Spike made an oppsie)

0, 1, 2, 3, 4, 5 and 6: Assign Z-Items, each number corresponds to a slot (0 is the first one and 6 is the Last.

Remember that the properties are in letters, so it wouldn’t be ID 0, but rather ID 0x30 (0 on char)

Function 14: Same as 13, but for the CPU and 2 additional properties:

l (“L” in low caps, not an “i”): CPU Level

a: Seems broken, but it’s meant to be a supposed 8th Slot for a Z-Item, but in the end, it replaces the first slot of the Z-Item of the next Team Member.

Function 15: Doesn't have parameters, but it does have one property that can be repeated an indefinite amount of times;

It links the Character Audios (Used to know which character is speaking or if the Subtitles are White or Gray)

```
// 0025bfa8

bool GscBattle::SetVoices(uint event,int *registers)

{
  bool hasProperty;
  int voice;
  int player;
  int counter;

  if ((event == 0) || (event == 4)) {
    counter = 0;
    while( true ) {
      hasProperty = GscReader::FindProperty('v',counter);
      if (!hasProperty) break;
      voice = GscReader::ReadIntParameter();
      player = GscReader::ReadIntParameter();
      GscResources::SetVoice(voice,player);
      counter = counter + 1;
    }
  }
  return true;
}
```

The Property "v" as it's seen here

```
0h  47 53 41 43 10 00 00 00 40 05 00 00 FB FF FF FF  GSAC....@...ûÿÿÿ
0h  01 00 0F 00 08 76 02 00 0A 00 00 00 0A 16 00 00  .....v..........
0h  08 76 02 00 0A 05 00 00 0A 00 00 00 08 76 02 00  .v...........v..
0h  0A 0C 00 00 0A 16 00 00 08 76 02 00 0A 0F 00 00  .........v......
0h  0A 00 00 00 08 76 02 00 0A 18 00 00 0A 16 00 00  .....v..........
0h  08 76 02 00 0A 0D 00 00 0A 16 00 00 08 76 02 00  .v...........v..
0h  0A 1F 00 00 0A 05 00 00 08 76 02 00 0A 20 00 00  .........v... ..
0h  0A 16 00 00 08 76 02 00 0A 21 00 00 0A 00 00 00  .....v...!......
0h  08 76 02 00 0A 22 00 00 0A 16 00 00 08 76 02 00  .v..."........v..
```

Function 16: Assigns the Rewards given uppon completion of the scenario. Has 15 parameters:

3 Z-Points Reward (one for each Difficulty)

3 Items

3 Stages

3 Characters

3 Story Mode Scenarios

```
/ 0025c018

ool GscBattle::SetRewards(uint event,int *registers)


 HistoryMenuSettings *pHVar1;
 Map_ID mapReward;
 Character_ID_32 characterReward;
 int reward;
 MenuDisplay *Menu_Display;

 Menu_Display = MenuDisplay::Data;
 pHVar1 = &MenuDisplay::Data->HistorySettings;
 if ((event == 0) || (event == 4)) {
   reward = GscReader::ReadIntParameter();
   pHVar1->Zeni[0] = reward;
   reward = GscReader::ReadIntParameter();
   (Menu_Display->HistorySettings).Zeni[1] = reward;
   reward = GscReader::ReadIntParameter();
   (Menu_Display->HistorySettings).Zeni[2] = reward;
   reward = GscReader::ReadIntParameter();
   (Menu_Display->HistorySettings).Items[0] = reward;
   reward = GscReader::ReadIntParameter();
   (Menu_Display->HistorySettings).Items[1] = reward;
   reward = GscReader::ReadIntParameter();
   (Menu_Display->HistorySettings).Items[2] = reward;
   mapReward = GscReader::ReadIntParameter();
   (Menu_Display->HistorySettings).Maps[0] = mapReward;
   mapReward = GscReader::ReadIntParameter();
   (Menu_Display->HistorySettings).Maps[1] = mapReward;
   mapReward = GscReader::ReadIntParameter();
   (Menu_Display->HistorySettings).Maps[2] = mapReward;
   characterReward = GscReader::ReadIntParameter();
   (Menu_Display->HistorySettings).Characters[0] = characterReward;
   characterReward = GscReader::ReadIntParameter();
   (Menu_Display->HistorySettings).Characters[1] = characterReward;
   characterReward = GscReader::ReadIntParameter();
   (Menu_Display->HistorySettings).Characters[2] = characterReward;
   reward = GscReader::ReadIntParameter();
   (Menu_Display->HistorySettings).RewardBattles[0] = reward;
```

Function 801: Assigns the Possition and Rotation of the Character

```
// 0025a120

bool GscBattle::SetActorTransform(uint event,int *registers)

{
  int player;
  uint role;
  float rotationAxis;
  float PI;
  Vector4 position;
  Vector4 rotation;

  if ((event == 0) || (event == 4)) {
    player = GscReader::ReadIntParameter();
    position.X = GscReader::ReadFloatParameter();
    position.Y = GscReader::ReadFloatParameter();
    position.Z = GscReader::ReadFloatParameter();
    role = player >> 0xf & 1;
    position.W = 1.0;
    PI = ::PI;
    rotationAxis = GscReader::ReadFloatParameter();
    rotation.X = (rotationAxis * PI) / 180.0;
    rotationAxis = GscReader::ReadFloatParameter();
    rotation.Y = (rotationAxis * PI) / 180.0;
    rotationAxis = GscReader::ReadFloatParameter();
    rotation.W = 1.0;
    rotation.Z = (rotationAxis * PI) / 180.0;
    If_GSC_Set_Actor_Position_Matrix(role,&position);
    If_GSC_Set_Actor_Rotation_Matrix(role,&rotation);
  }
  return true;
}
```

First parameter indicates to which character is applied, the following 3 parameters are possition, while the last 3 are rotation.

Function 802: Same as 801, but it only assigns possition

```
// 0025a210

bool GscBattle::SetActorPosition(uint event,int *registers)

{
  int player;
  Vector4 position;

  if ((event == 0) || (event == 4)) {
    player = GscReader::ReadIntParameter();
    position.X = GscReader::ReadFloatParameter();
    position.Y = GscReader::ReadFloatParameter();
    position.Z = GscReader::ReadFloatParameter();
    position.W = 1.0;
    If_GSC_Set_Actor_Position_Matrix(player >> 0xf & 1,&position);
  }
  return true;
}
```

Function 803: Same as before, but only assigns rotation:

```
// 0025a280

bool GscBattle::SetActorRotation(uint event,int *registers)

{
  int player;
  float rotationAxis;
  float radius;
  Vector4 rotation;
  float PI;

  if ((event == 0) || (event == 4)) {
    player = GscReader::ReadIntParameter();
    PI = ::PI;
    rotationAxis = GscReader::ReadFloatParameter();
    radius = 180.0;
    rotation.X = (rotationAxis * PI) / 180.0;
    rotationAxis = GscReader::ReadFloatParameter();
    rotation.Y = (rotationAxis * PI) / radius;
    rotationAxis = GscReader::ReadFloatParameter();
    rotation.W = 1.0;
    rotation.Z = (rotationAxis * PI) / radius;
    If_GSC_Set_Actor_Rotation_Matrix(player >> 0xf & 1,&rotation);
  }
  return true;
}
```

Function 804: Makes the Character appear (In the case it was invisible)

```
// 0025a340

bool GscBattle::EnableActorVisibility(uint event,int *registers)

{
  int player;

  if ((event == 0) || (event == 4)) {
    player = GscReader::ReadIntParameter();
    If_GSC_Set_Actor_Hidden_Status_Off(player >> 0xf & 1);
  }
  return true;
}
```

Function 805 makes the Character invisible

```
// 0025a380

bool GscBattle::DisableActorVisibility(uint event,int *registers)

{
  int iVar1;

  if ((event == 0) || (event == 4)) {
    iVar1 = GscReader::ReadIntParameter();
    If_GSC_Set_Actor_Hidden_Status_On(iVar1 >> 0xf & 1);
  }
  return true;
}
```

Function 806 activates the Aura

```
// 0025a3c0

bool GscBattle::EnableActorAura(uint event,int *registers)

{
  int player;

  if ((event == 0) || (event == 4)) {
    player = GscReader::ReadIntParameter();
    If_GSC_Set_Actor_Aura_VFX_On(player >> 0xf & 1);
  }
  return true;
}
```

Function 807 turns the Aura off

```
// 0025a400

bool GscBattle::DisableActorAura(uint event,int *registers)

{
  int player;

  if ((event == 0) || (event == 4)) {
    player = GscReader::ReadIntParameter();
    If_GSC_Set_Actor_Aura_VFX_Off(player >> 0xf & 1);
  }
  return true;
}
```

Function 808 turns Aura Charge

Function 809 turns it off

Function 810 activates that Ki Explosion when reaching Max Power

Function 901 assigns the Character Animation

```
// 0025a4f0

bool GscBattle::SetActorAnimation(uint event,int *registers)

{
  bool bVar1;
  bool loop;
  bool wait;
  int player;
  int animation;
  uint role;
  float animationSpeed;

  bVar1 = false;
  switch(event) {
  case 0:
  case 4:
    player = GscReader::ReadIntParameter();
    animationSpeed = 0.0;
    role = player >> 0xf & 1;
    animation = GscReader::ReadIntParameter();
    bVar1 = GscReader::FindProperty('t');
    if (bVar1) {
      animationSpeed = GscReader::ReadFloatParameter();
    }
                    // Set Animation on Loop?
    loop = GscReader::FindProperty('l');
                    // Wait until Animation Finishes to Continue?
    wait = GscReader::FindProperty('w');
    bVar1 = loop;
    if (!wait) {
      bVar1 = true;
    }
    If_GSC_Set_Actor_Hidden_Status_Off(role);
    *registers = role;
    If_GSC_Set_Actor_Cutscene_Animation_Data(animationSpeed,role,animation,loop);
    if (event == 4) {
      bVar1 = true;
    }
```

It can have 3 properties

"t" has 1 parameter and assigns the speed of the animation

"l" is to activate the loop of the animation

And "w" (property we'll see on other functions) is the "wait" property, it indicates that the flow stops there until the process ends, in this case, it would be used for stop there until the animation is finished.

Function 902 has no parameters. When it's present, it indicates that the player can move, this function is unused since Function 4 already exists

Function 701 indicates what should be loaded on the Subtitles File

```
/ 0025a880

ool GscBattle::LoadTxtPak(uint event,int *registers)


 int battleID;

 if ((event == 0) || (event == 4)) {
   battleID = GscResources::GetBattleID();
   GscResources::SetTxtPak(battleID + 0x211);
 }
 return true;
```

Function 702 is the same, but for LPS Files

```
// 0025a8b8

ool GscBattle::LoadLpsPak(uint event,int *registers)

{
  int GSC_ID;
  int GSC_LPS_Base_File_ID;

  if ((event == 0) || (event == 4)) {
    GSC_LPS_Base_File_ID = 0x275;
    if ((DbzsmSaveData::Data->FLAGDATA_Memcard_0x1608 & English_Voices) == Disabled) {
      GSC_LPS_Base_File_ID = 0x243;
    }
    GSC_ID = GscResources::GetBattleID();
    GscResources::SetLpsPak(GSC_LPS_Base_File_ID + GSC_ID);
  }
  return true;
}
```

Function 1301 doesn't really do anything, but it has 1 parameter, generally which is a string attached to a comment (used during Debugging)

```
// 0025a878

bool GscBattle::DebugComment(uint event,int *registers)

{
  return true;
}
```

It's only used on the last Unused GSC Slots

Function 1302 establish the settings of the subtitles, it doesn’t have parameters, but it does have several propreties:

First is the property "l" which indicates the region of the subtítles and has one parameter:

0: JP

1: USA

2: EUR

3: KOR (Unused)

"n" has one parameter: which indicates whatever the subtitles are white or grey.

"d" indicates the possition of the text on X and Y. They’re 2 type int parameters

"a" indicates the size of the text on X and Y. They’re 2 type float parameters

"s" is the scale and is a float parameter

"c" is the color of the Text. Has 4 parameters that correspon to the RGBA

"p" is separation. Has 2 parameter, First one is separation between lines y and the second one is between letters

"w" is the text alignment. Only has 1 Parameter:

0: Right

1: Center

2: Left

"T" is the type of letter. Only has 1 parameter. Value 2 indicates Bold which is the one used in Story Mode, Which gives the outline.

"O" is the size of the outline in X and Y. They’re 2 type int parameters

And last is "C" is the color of the outline. Has 4 parameters that correspon to the RGBA

Function 1001 start a fade out

```
// 0025a658

bool GscBattle::EnableFadeOut(uint event,int *registers)

{
  bool bVar1;
  bool bVar2;
  float Fade_Time;

  bVar2 = false;
  if (event < 5) {
                    // WARNING: Switch is manually overridden
    switch(event) {
    case 0:
      Fade_Time = GscReader::ReadFloatParameter();
      bVar1 = GscReader::FindProperty('W');
      if (bVar1) {
                    // Set White Fade
        *registers = 1;
      }
      else {
                    // Set Black Fade
        *registers = 0;
      }
      Set_Fade(Fade_Time,*registers,true);
      registers[1] = 0;
                    // Call 'wait' Function to Wait until Action ends to continue Cutscene
      bVar1 = GscReader::FindProperty('w');
      registers[2] = (int)bVar1;
      if (!bVar1) {
        bVar2 = true;
      }
      break;
    case 1:
      if (registers[2] == 0) {
        bVar2 = false;
      }
```

It has 1 parameter, which is the time that the fade out lasts, it can have the property “W” which indicates that the fade out is white, it doesn’t have parameters, it can also have the wait property “w”, which indicates the flow stops until the fade out ends

Function 1002 is the same, but with a fade in

Function 1003 forces to end the fade

```
// 0025a838

bool GscBattle::DisableFade(uint event,int *registers)

{
  if ((event == 0) || (event == 4)) {
                    // Force Black Fade End
    FUN_00252aa8(0);
                    // Force White Fade End
    FUN_00252aa8(1);
  }
  return true;
```

Function 1201 habilitates the cutscene camaras

```
// 00259d10

bool GscBattle::CameraOn(uint event,int *registers)

{
  float rotationAxis;
  float PI;
  Vector4 position;
  Vector4 rotation;

  if (event < 4) {
    if (event != 0) {
      return false;
    }
  }
  else if (event != 4) {
    return false;
  }
  GscResources::Data.field70_0x170[2] = 0;
  GscResources::Data.field70_0x170[0] = 0;
  GscResources::Data.field70_0x170[1] = 0;
  PI = ::PI;
  position.X = GscReader::ReadFloatParameter();
  position.Y = GscReader::ReadFloatParameter();
  position.Z = GscReader::ReadFloatParameter();
  rotationAxis = GscReader::ReadFloatParameter();
  rotation.X = (rotationAxis * PI) / 180.0;
  rotationAxis = GscReader::ReadFloatParameter();
  rotation.Y = (rotationAxis * PI) / 180.0;
  rotationAxis = GscReader::ReadFloatParameter();
  rotation.Z = (rotationAxis * PI) / 180.0;
  FUN_0012bd70(&position,&rotation);
  return true;
}

```

Has 3 parameters float for the possition and 3 more for rotation

Function 1202 doesn’t have them, but has properties

"I" indicates another point for the camera to move the camera

That property has 7 parameters. First one is the speed from the previous possition to that possition, the following 3 are for possition and the last 3 for rotation.

all 7 are floats, can also have the wait property, which indicate to stop the flow until the camera finishes

Function 1203 restablishes the camera back to the player

```
// 00259df8

bool GscBattle::CameraOff(uint event,int *registers)

{
  if ((event == 0) || (event == 4)) {
    If_Reset_Battle_Camera();
  }
  return true;
}
```

Unused since Function 4 exists

Function 1204 indicate Camera Shake

```
// 00259e28

bool GscBattle::CameraShakeOn(uint event,int *registers)

{
  bool loop;
  float shake;

  if (event < 4) {
    if (event != 0) {
      return false;
    }
    shake = GscReader::ReadFloatParameter();
    If_GSC_Set_Screen_Shake(shake);
    loop = GscReader::FindProperty('l');
    if (loop) {
      GscResources::Data.CameraShake = shake;
    }
  }
  else if (event != 4) {
    return false;
  }
  return true;
}
```

Has 1 parameter float, which indicates the force of the shake and can have the property "l" which indicates loop to keep that shake

Function 1205 has no parameters. It stops the camera shake

```
// 00259eb0

bool GscBattle::CameraShakeOff(uint event,int *registers)

{
  if ((event == 0) || (event == 4)) {
    GscResources::Data.CameraShake = 0.0;
    If_GSC_Set_Screen_Shake(0.0);
  }
  return true;
}
```

Function 1501 establishes a Music change

```
/ 0025abc8

ool GscBattle::SetBgm(uint event,int *registers)


 int bgm;

 if ((event == 0) || (event == 4)) {
   bgm = GscReader::ReadIntParameter();
   Play_BGM(bgm + 0x10568);
 }
 return true;
```

Function 1502 lowers the volume of the Music, has no parameters, but can have the property wait, which stops the flow until the music ends

```
// 0025ac08

bool GscBattle::SetBgmLowVolume(uint event,int *registers)

{
  bool bVar1;

  bVar1 = false;
  if (event < 5) {
                    // WARNING: Switch is manually overridden
    switch(event) {
    case 0:
      Set_BGM_Low_Vol();
                    // Looks for 'wait' Function to Wait until Action ends to Continue Cutscene
      bVar1 = GscReader::FindProperty('w');
      if (bVar1) {
        bVar1 = false;
      }
      else {
        bVar1 = true;
      }
      break;
    case 2:
      bVar1 = BGM_ADXT_Stopped();
    }
  }
  return bVar1;
}
```

Function 1601 has one parameter int that indicates the ID of the Audio of the SeBattle

```
// 0025ac78

bool GscBattle::PlaySeBattleAudio(uint event,int *registers)

{
  int audio;

  if (event == 0) {
    audio = GscReader::ReadIntParameter();
    If_Play_VAG_SE_Audio(4,audio);
  }
  return true;
}
```

Function 1602 is an Audios from the 300 ADX Files from Story Mode. Has one parameter int with the Audio ID

```
bool GscBattle::PlayVicAudio(uint event,int *registers)

{
  bool bVar1;
  bool bVar2;
  int audio;
  int iVar3;
  int ADX_File_ID;

  bVar1 = false;
  if (event < 5) {
                    // WARNING: Switch is manually overridden
    switch(event) {
    case 0:
      audio = GscReader::ReadIntParameter();
                    // Find "VIC" Property
      bVar1 = GscReader::FindProperty('h');
      if (bVar1) {
        iVar3 = GscReader::ReadIntParameter();
        *registers = iVar3;
      }
                    // 'wait' Function
      bVar1 = GscReader::FindProperty('w');
      *registers = 0;
      bVar1 = !bVar1;
      ADX_File_ID = GscResources::GetAudioFileID(audio);
      Play_VIC_Audio(0,ADX_File_ID);
      break;
    case 2:
      bVar2 = VIC_ADXT_Stopped(*registers);
```

Can have the property "h" which has 1 parameter that indicates the audios channel, since it can reproduce 2 audios at the same time. It can also have the wait property, which stops the flow until the audio ends.

Function 1603 is the same as before, but has 2 parameters instead of 1.

The first one is the player that reproduces the audio and the second one is the ID of the audio

Same properties as before, this one is for Character Audios and the other one for any Audio

Function 1701 has 1 parameter, which indicates a button. This function stops the flow until the button assigned is pressed

```
// 0025aff0

bool GscBattle::EnableCutsceneButtonSkip(uint event,int *registers)

{
  bool bVar1;
  int button;

  bVar1 = false;
  if (event < 5) {
                    // WARNING: Switch is manually overridden
    switch(event) {
    case 0:
      button = GscReader::ReadIntParameter();
      *registers = 0;
      registers[1] = button;
      break;
    case 2:
      if (*registers == 0) {
        if ((Pad_Data_ARRAY_00333800[0].Menu_Button_Inputs & 1 << (registers[1] & 0x1fU)) == NULL) {
          bVar1 = false;
        }
        else {
          *registers = 1;
        }
      }
      else {
        bVar1 = true;
      }
      break;
    case 4:
      bVar1 = true;
    }
  }
  return bVar1;
}
```