Reconstruction of Smart Phone Camera Effect Parameter Management Subsystem

Wanpeng Tang[1(✉)] and Guoming Rao[2]

[1] Guangzhou City Polytechnic, Guangzhou, Guangdong, China
46186518@qq.com
[2] Spreadtrum Communications (Shanghai Co. Ltd.), Shanghai, China

Abstract. For the camera effect parameter management and debugging of smart phone platform, customers have many complaints and dissatisfaction, the most important problems are: poor readability, poor separability, poor scalability, compilation free, there are a lot of redundant parameters. In this paper, a new system is constructed for parameter management, which is analyzed and reconstructed from the aspects of parameter structure, parameter search, parameter storage and parameter debugging, and the software system is redesigned to solve the above software problems completely from the system.

Keywords: process debugging parameters · Scenario · Mode Mode · block

1 The Research Status

With the concentration of smartphone customers to premium brand customers, the smartphone projects of premium brand customers have higher and higher requirements for camera effects, and the scene debugging requirements are more and more refined. At the same time, the company's chip camera module has expanded to more than 50, and the parameter scale of each module has also increased rapidly, Under the circumstance that the project period is determined, higher requirements are put forward in terms of commissioning scale and commissioning schedule. Meanwhile, brand mobile phone customers pursue the gene of product differentiation, requiring that the chip platform must be easy to integrate the post-processing algorithm of each camera [1]. To meet the above requirements, the camera effect parameter management of the current smartphone platform has many problems in the system, mainly in the following aspects: poor readability, poor separability, poor scalability, unable to completely achieve compile-free, poor debugging, and a large number of redundant parameters.

In view of the above problems, this study redesigns the current smart phone effect parameter management software subsystem, completely solves the above problems, and meets the needs of customers in the smart phone platform on the effect parameter debugging and integration.

A. Li et al. (Eds.): 6GN 2022, LNICST 504, pp. 240–250, 2023.
https://doi.org/10.1007/978-3-031-36011-4_21

2 Defects and Reconstruction Ideas of the Original Effect Parameter Management Subsystem

The current camera effect parameter management subsystem is still based on the system design of function machine era, which has the following design defects:

2.1 Poor Readability and Redundancy

The current parameter file is in C format, but there are a lot of incomprehensible hexadecimal data, and each module is intertwined and difficult to understand, there are great difficulties in dealing with multi-person debugging, and lack of corresponding annotations; After years of development of the whole debugging parameters, NR Block is bound with many denoising modules. If a single module is added, other NR modules need to be added synchronously, resulting in redundancy of parameters and low reuse of parameters. As a result, when multiple scene parameters are added to effect parameters, the parameter file is large, leading to large memory consumption.

2.2 Poor Divisibility and Scalability

Currently, parameter files are separated according to mode. For example, capture and preview are described in two files respectively, but each file contains the parameters of each module. Such parameters between different file can't reuse, and there are a lot of redundant parameters between different files.The debugger of different modules may modify the same parameter file. In addition, it is difficult to extend third-party algorithms and new modes. Coupling and compatibility should be considered. Due to customization requirements, customers often require different effect parameters for different application scenarios. At present, Mode Mode is difficult to meet integration requirements [2].

2.3 Compilation is not Exempt

The parameters of the current file is C description file, it is necessary to form parameter library files at compile time for running load, while debugging can also go directly to read C parameter file format, but there are two major drawbacks: first, crash will occur when the data size of the C file is inconsistent with that of the loaded SO-file; Second, you cannot update parameters locally; to update parameters, you must update them all.

Then, according to the above problems, consider the following points for system reconstruction design:

A) Considering the readability and convenience of THE XML file format, the parameter file format will be changed to XML format, and each parameter needs to be annotated in THE XML format to describe the default value, recommended value range, allowed value range, and debugging frequency of the parameter. Considering the performance of XML files when loading, the parameters of mass production products are bin files in binary format, which can be exchanged between XML and bin format. When defining XML, remove some of the old parameters that still exist

in the parameter file and are not actually used; When multiple groups are configured, the maximum number of groups is not supported. Currently, the number of fixed groups is changed to a variable number for SMART and NR modules. The flexible number of Smart groups also requires that the Smart interpolation for each module should be distributed to each module.

B) Effect debugger debugging development trend is according to the algorithm module block or according to the module block field debugging, so the segmentation should be from the perspective of the module block. By storing parameters in different files according to the module block, the debugger of different modules will not have the problem of modifying the same parameter file. This design is especially beneficial for multi-user debugging. In addition, for third-party algorithms, their parameters often exist as independent files, which are stored as different parameter files in modules, and are also very friendly to the compatibility of third-party algorithms. Smart modules should also be scattered among modules. The original Mode Mode needs to be divided into more dimensions. The initial idea is to take Scenario as the outline, scene characteristics as the context, and algorithm module as the purpose to configure parameter architecture.

C) Use BIN files and XML files to solve the compile-free problem. Parameters themselves are only parameters and do not need to be compiled. Parameters are divided into independent small files according to the block module, and parameters of a certain module can be updated independently.

3 Parameter Storage Mode

3.1 Parameter File and Its Organizational Form

The parameter file adopts the format of BIN + XML. The parameter file of each module block is an independent bin file or an XML file. For example, the original Bayer NR module is independent of LM.BIN file or LM. XML file. The LM. BIN and LM. XML files can be converted to each other through tools. The debugger mainly uses XML files for debugging and parameter submission The LM. BIN and LM. XML files can be pushed to the phone, and the debugger and phone can parse these two formats respectively.

3.2 Organization Form of Parameter Files

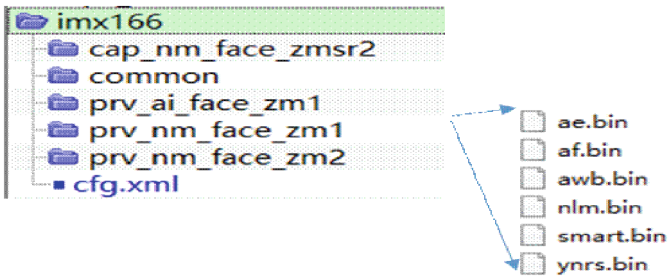


Fig. 1. Organization of a parameter file

For a module, the parameter file is divided into the module parameter root directory, Scenario subdirectory and module parameter file according to the hierarchical structure. There is also a unique cfg. XML file under the module parameter root directory, as shown in Fig. 1.

A Root Folder

For a module, the root directory of the parameters is named after the module, as before. Modules are usually referred to sensors, such as IMX166. If a project consists of multiple modules that use IMX166, you can use sensor_name + suffix, for example, IMX166_rear stands for rear and IMX166_front indicates forward photography.

B Subdirectories

Each subdirectory corresponds to a Scenario, which is the abbreviation of the scenario name. The shorthand rules can be spliced together according to each field of scenario. The mapping between Scenario ID and Scenario name (that is, subfolder) is described in cfg.xml.

C cfg.xml

The ID of scenario, name of Scenario and reuse relation of algorithm modules involved in scenario are described.

D Module Block Parameter File

The module block parameter is separated from each mode file and becomes an independent file: all algorithm modules used in Scenario should be separated. Not all module parameters in a scenario will exist. When the NLM parameters of Scenario #1 are recycled, the nlM.bin/NLM.xml file in the Scenario #2 folder will exist. The lm. Bin/lm. XML file will not exist in the scenario#1 folder. This reuse relationship is described in the cfg.xml file [3].

4 Design the Overall Parameter Framework Based on Scenario

4.1 Definition of Scenario

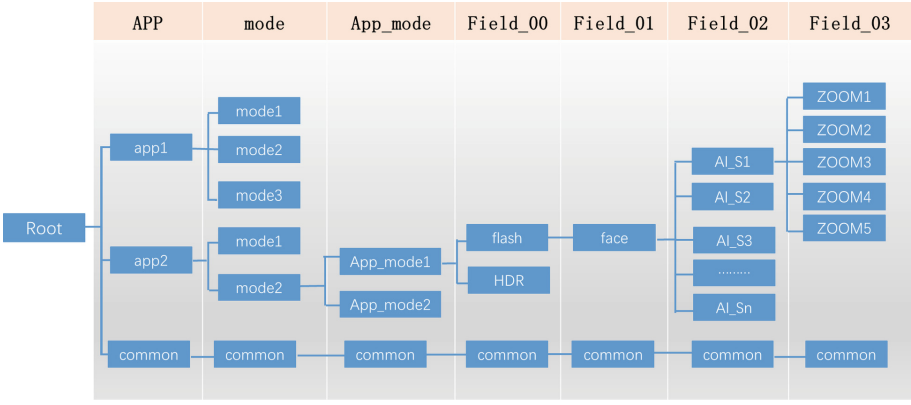


Fig. 2. Composing fields of Scenario

Scenario is defined based on the attributes of the applicable Scenario. Currently, it is defined from seven aspects: App, mode(related to image size), App_mode (related to App's level 1/level 2 menu), Field00–Field03 (describe scene feature from 4 levels), as shown in Fig. 2.

4.2 Field Instance List of Scenario

Table 1. Platform Scenario instantiation table

app	mode	app_mode	field_00	field_01	field_02	field_03	ID
common	common	common	Common	common	**common**	common	00
factory	preview_ binning	**night**	**flash**	**face**	ai_sunriseset	**zoom_2x**	01
wechat	**preview_ full**	panorama	hr		ai_firework	zoom_4x	02
tictok	capture_ binning	Apture_boke h			ai_food	zoom_8x	03
facebook	capture_full	portait			ai_foliage	zoom_16x	04
messager	video_previ ew_720p	pro			ai_document	zoom_32x	05
qq	video_previ ew_1080p	time_lapse			ai_pet	zoom_64x	06
snapchat	video_recor ding_720p	slow_motion			ai_flower	zoom_128x	07
whatapps	video_recor ding_1080p	moving_capt ure			ai_sky	zoom_sr_2x	08
	Video_call	capduringpr eview			ai_snow	zoom_sr_4x	09
		qr			ai_overcast	zoom_sr_8x	0a
		sport			ai_chinese_ building	zoom_sr_16 x	0b
		landscape			ai_building	zoom_sr_32 x	0c
		firework			ai_car	zoom_sr_64 x	0d
		autumn			ai_bicycle	zoom_sr_12 8x	oe
		refocus			ai_autumn_le af		0f
		filter			ai_beach		10
		fusion			ai_lake		11
		night_pr			ai_waterfall		12

Based on the above rules, the following Scenario list is instantiated, as shown in Table 1. All scenarios can be formed by free combination of each field. The actual Scenario supported by the platform is only a subset of all possible scenarios.

1) According to the ID column in Table 1, the Scenario ID of the current scenario is spliced from left to right in the above table. Scenario ID is defined as a 64-bit unsigned number with the highest bit complement 0x00. Such as red font marked fields in Table 1 (wechat/preview_full/night/flash/face/common/zoom_2x), the ID after stitching is 0 x00–02-02–01-02–02-01–01.
2) Match the ID level by level from left to right. If the ID exists, the field ID is the actual ID. If the field id does not match, the common ID is used.
3) Mode extension response: If the current conflict mode is no longer in the future, it can be extended by adding the items of this field to different fields. If HDR is first in field_00, then HDR takes precedence. If FLASH is first in field_00, then FLASH takes precedence.

4.3 Scenario-Block Map

The algorithm blocks involved in a Scenario are described by scenario-block Map. Generally, each platform should have a scenario-block Map table. The scenario-block Map table may change in different versions of the platform, and customers may even customize their own Map table according to their needs. Therefore, Scenario-block maps should be easy to configure and change, and should be consistent on the tool side and mobile side.

Table 2. Example of scenario-block Map

Scenario_id	Scenario_name	"BLC"	"Y_AFL3"	"BFCHS"	"HIST"	"HIST2"	"Y_GAMMA"	"RGBGAIN"	"ADTFL"	"PDAE"	"4in1"	"AEMDAPT"	"AEALSC_2D"	"NR_3dlayer_nr"	"NR_baynr"	"NR_bpc"	"NR_3dnr"	"NR_cap_3dnr"	"NR_ccue_uvdiv"	"NR_RRcfa"	"NR_RRcnr"	"NR_RRree"	"CMC10"	"NR_grgb"	"NR_ircbalance"	"NR_imvstan"	"NR_RRtm"	"NR_RRppe"	"NR_RRgb_alm"	"NR_RRgb_dither"	"NR_uvdn"	"NR_uvp_dcost"	"RGBA_cdM"	"NR_RRvst"	"NR_RRynm"	"NR_yuv_notsel"	"HSV"	"POS_TEXRI"	"CCE"	"Y_L1"	"SMAART"	"SLS3"	"AE32"	"AWB32"	"ENVDAF"	"PAF"	"AFT"	"AFAS"	"DFLH"	"AE_SYNC"	"ATM4GAMMA"	"TOF"	"HDR3.0"	mfnr	fdr
0x0000 0001	common	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
0x0001 0001	prv1_xx_xx	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0
0x0002 0001	prv2_xx_xx	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0
0x0005 0001	cap1_xx_xx	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	1
0x0006 0001	cap2_xx_xx	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	1

According to the example of scenario-block map in Table 2. Each row of the Map is a Scenario, and each column is an algorithm module (Block). Adding a Scenario means adding rows, and adding an algorithm module means adding columns.

The values in the table are defined as follows:

1: Scenario requires the parameters of the block.

0: Scenario does not require the parameters of the block.

Note that the scenario-block Map is sparse, and the Scenario IDS and column ids are not continuous. Corresponding rows or columns exist only when necessary. Scenario not described in the table, but if it may actually occur, it means that this Scenario will be replaced by Common if it occurs – that is, all scenarios not described are merged into Common [4].

4.4 Reuse Relationship of Blocks in Scenario

In order to reduce disk and memory usage, blocks of each Scenario should be allowed to reuse corresponding blocks of other Scenarios. A block with a reuse relationship will have only one memory entity; In principle, there should be only one bin or XML file entity. The following is a concrete example of reuse relationships, mechanisms, and implementations.

Table 3. Example table of scenario-block reuse relationship description

Scenario_name	"BLC"	"Y_AFL3"	"BCHS"
common	common	common	common
prv1_xx_xx_xx	common	common	common
prv2_xx_xx_xx	prv2_xx_xx_xx	prv2_xx_xx_xx	common
cap1_xx_xx_xx	common	common	common
cap2_xx_xx_xx	prv2_xx_xx_xx	prv2_xx_xx_xx	common

Table 3 is a tabular description of cfg. XML, which actually describes that Scenario prV1_XX_XX_XX BLC, Y_AFL3 and BCHS completely use Scenario common. Therefore, the prv1_XX_XX_XX folder will have no bin and XML files with three blocks. Prv2_xx_xx_xx BLC and Y_AFL3 do not reuse other Scenarios, so prV2_XX_XX_XX folder will have blc.bin and y_afl3.bin files, but PRV2_XX_xx_XX BCHS reuse common, Therefore, the bchs.bin file does not exist under prV2_xx_XX_xx. Similarly, cap2_xx_xx_XX BLC and Y_AFL3 are multiplexed from prV2_xx_xx_xx BLC. Bin and y_afl3.bin files. Cap2_xx_xx_xx does not have two BLCOK bin files. The Common directory contains bin or XML files for all blocks.

Table 3 Equivalent examples of cfg. XML are shown in Fig. 3. The equivalent data description of cfG. XML is shown in Table 4. Based on the data description table, the memory description for parameter loading can be further standardized.

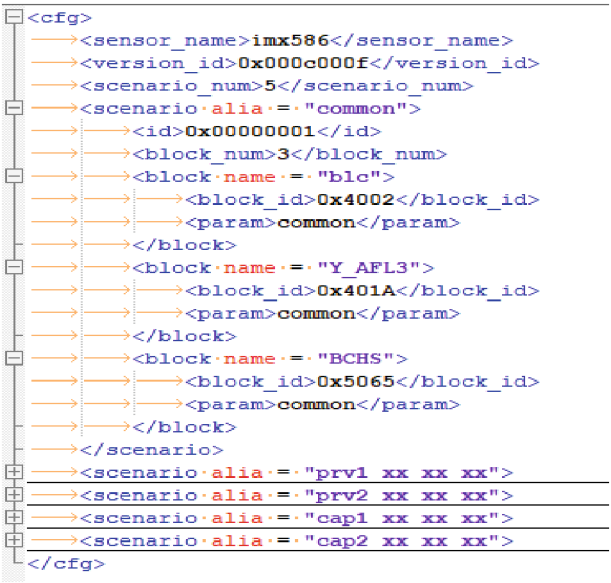


Fig. 3. Example of the cfg. XML file

Table 4. Example table of datatization of scenario-block reuse relationship

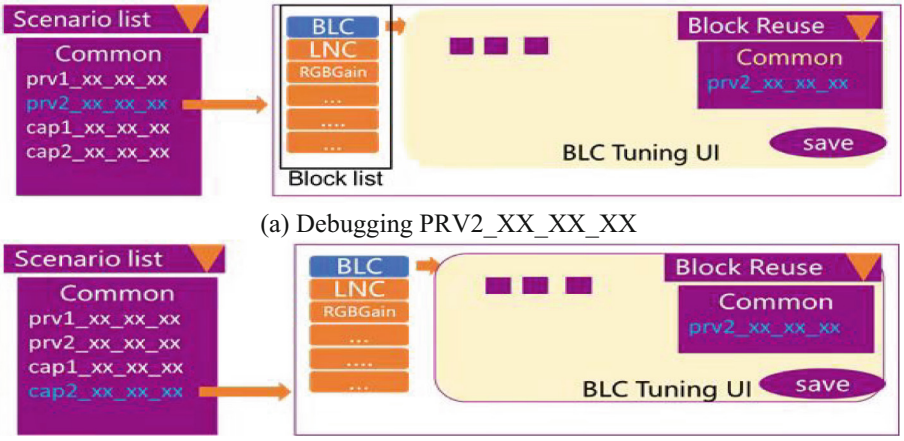
Scenario_name	"BLC"	"Y_AFL3"	"BCHS"
0x00000001	0x00000001	0x00000001	0x00000001
0x00001001	0x00000001	0x00000001	0x00000001
0x00002001	0x00002001	0x00002001	0x00000001
0x00005001	0x00000001	0x00000001	0x00000001
0x00006001	0x00002001	0x00002001	0x00000001

4.5 Interface Design of Scenario Block Parameter Reuse Relationship in Debugging Tools

In order to facilitate the debugging personnel to configure the effect parameter multiplexing relationship, improve the efficiency of parameter multiplexing. Guide the debugger to take full advantage of parameter multiplexing to minimize memory and disk consumption, the debugging tool is designed as follows:

The Scenario List:

1) The Scenario List drop-down table should list all scenarios supported by the platform.
2) Select a Scenario and call out all non-0 blocks according to the configuration in the scenario-block map table to form a Block list to be debugged. Each Block can be a button as before. Click to call out the debugging interface of the module.



(a) Debugging PRV2_XX_XX_XX

(b) during cap2_XX_XX_XX debugging

Fig. 4. Shows the interface design of the BLC debugging tool

The operation and working mechanism are described, as shown in Fig. 4.

1) The Block reuse menu invokes all Scenario names of the Block data entity.
2) Select a Scenario and load the block data of the Scenario on the block debugging interface.
3) If the referenced block data is modified, the data of the block will be saved in the folder of the current Scenario (not the referenced Scenario) during save, and the reuse relation will point to the current Scenario.

For example, prV2_XX_XX_XX BLC has no data entity at the beginning. Prv2_xx_xx_xx starts debugging from Block.

Reuse Select Common from the drop - down list to reuse BLC data of Common. If the modification is performed on the BLC debugging interface, the data is saved to the blc.bin file in the prV2_XX_XX_XX folder, and cfg. XML is modified to indicate that PRV2_XX_XX_XX BLC uses prV2_XX_XX_XX_XX BLC parameters. Note that blc.bin in the common folder will not be modified at this time.

When debugging CAP2_XX_XX_XX, if both COMMON and PRV2_XX_XX_XX have BLC block entities,

Then the block reuse drop-down menu will list two Scenarios, common and PRV2_XX_XX_XX.

Assume that Scenario prV2_XX_XX_XX is selected, BLC data is not modified and click Save. Since BLC data is not modified, there is no need to save BLC. Bin to cap2_XX_XX_XX directory. Do not create cap2_xx_xx_xx if the cap2_xx_xx_xx directory does not already exist. However, the cfg. XML file needs to be modified to record that the BLC of CAP2_XX_XX_XX comes from the Scenario prV2_XX_XX_XX [5].

5 Commissioning and Conclusion

Through the reconstruction of camera effect parameter subsystem, the coupling pain point of parameter debugging for a long time was solved and the parameters were improved.The scalability of data improves the efficiency of fast integration of third-party algorithms. It used to take one month to integrate and stabilize a new algorithm, but now it only takes two weeks.

1) Parameter Storage Format and Advance Integration Ability

XML (xxx. XML + cfg. XML can be parsed and converted using tools). In cfg. XML, the relationship between Scenario and block parameters is configured. The implementation of XML format will allow algorithm and integration to get rid of the dependence of camera effect debugging tools, and advance the time point of integration to algorithm pre-research/simulation stage. Without tools, directly modify the content of XML file, effect simulation and debugging can be carried out.

2) Scenario Supported by the Current System

At present, a total of 121 scenarios are summarized before parameter reconstruction, which are described from 7 dimensions. Dimension classification and inclusion relations can be updated and expanded as required. Supporting 121 scenarios does not mean that you need to debug 121 scenarios. You need to make full use of the reuse relationship of parameters between scenarios, which gives the debugger great flexibility to implement different debugging strategies for different customers, different projects and different effect requirements.

3) Smart Function Outlook

The current SMART mechanism is relatively crude, and the new parameters will allow you to customize your SMART policy for each algorithm module. Although the current smart parameters are uniform, each block defines its own SMART parameters, which allows each block to have its own SMART policy. In other words, smart parameters can vary with different blocks.

In subsequent evolution, each block can allow the evolution of its own smooth and switch strategies, which require parameters determined by the module itself.

References

1. Vijayakumar, V.T.R., Subha, B.: Product quality and its relationship on customer satisfaction and brand loyalty of mobile phone users in Chennai city. ZENITH Int. J. Multi. Res. **3**(7), 264–270 (2013)
2. Khatri, F.I., Zogbi, G., Boroson, D.M.: Telescope divisibility limitations due to synchronization of array-based photon counting receivers in laser communications links. Physica A Stat. Mech. Appl. **6877**, 70–77 (2008)
3. China Academy of Telecommunications Technology, Researchers Submit Patent Application: Method and Device for Processing Quality of Service Parameter in Handover Scenario.USPTO 20190215735 (2019)

4. González-Portillo, L.F., Muñoz-Antón, J., Martínez-Val, J.M.: An analytical optimization of thermal energy storage for electricity cost reduction in solar thermal electric plants. Appl. Energy **185**, 531–546 (2017)
5. Wang, D., Chen, K., Wang, L.: Practical XML Course, Tsinghua University Press (2014)