aFrame

electrorganic percussion

EN

aFrame API Specification

for Ver 1.10

1. Introduction

The aFrame has external communication function by updating to firmware version 1.10 or later.

The aFrame communicates with the host CPU via CP2012 virtual COM port interface.
Serial port settings are shown in the table below.

The following is the specification of the API (Application programming Interface) function.

Fig 1

BaudRate	115200 bps
DataBits	8
StopBits	1
Parity	none
HandShake	none

2. Function List

Functions are classified as three types:

- Get Data from aFrame.
- Set Data to aFrame.
- External Editing aFrame.

[note] It is necessary to set the MODE_EDIT_EXT mode for using External Editing aFrame type function. For details on the mode, see Appendix A.

[note] For details on the External Editing, see Appendix B.

Get Data from aFrame type function list is shown below.

Fig 2

Function Name	Commnad	Description
GetVersion	aFG0	Get aFrame firmware version.
GetMode	aFG1	Get currently mode from 10 types. For detail on the mode, see Appendix A.
GetSwitch	aFG2	Get currently switch state.
GetEncoder	aFG3	Get currently Encoder value.
GetLED	aFG4	Get currently LED state.
GetLCD	aFG5	Get currently LCD strings.
GetGroupBank	aFG6	Get currently Group.
GetGroupNum	aFG7	Get currently Number.
GetPeakLevel	aFG8	Get input(Center and Edge) and output(L, R) peak level.
GetGetPressure	aFG9	Get pitch and mute pressure level.
GetCurrentGroupToneNum	aFGA	Get currently group tone information.
GetCurrentToneName	aFGB	Get currently selected instrument and effect tone name.
GetCurrentToneData	aFGC	Get currently selected instrument or effect tone data.
GetProjectGroupList	aFGD	Get project group information list.
GetProjectToneNameList	aFGE	Get project instrument or effect tone name list.
GetProjectToneDataList	aFGF	Get project instrument or effect tone data list.

Set Data to aFrame type function list is shown below.

Fig 3

Function Name	Commnad	Description
SetExtModeSw	aFS1	Set aFrame to MODE_EDIT_EXT or MODE_HOME. For detail on the mode, see Appendix A.
SetSwitch	aFS2	Set switch state.
SetEncoder	aFS3	Set encoder value.
SetLED	aFS4	Set LED state.
SetLCD	aFS5	Set LCD strings.
SetGroupBank	aFS6	Set group bank up or down. This function is valid on the MOD_HOME only.
SetGroupNum	aFS7	Set group number up or down. This function is valid on the MOD_HOME only.

External Editing aFrame type function list is shown below.

Fig 4

Function Name	Commnad	Description
ExtSelectGroup	aFE0	Select group and number.
ExtWriteGroup	aFE1	Write group and number.
ExtChangeToneNum	aFE2	Change instrument or effect tone number.
ExtChangeEditBuffParam	aFE3	Change instrument or effect parameter value.
ExtChangeEditBuffName	aFE4	Change instrument or effect tone name.
ExtWriteEditBuffToProject	aFE5	Write instrument or effect tone data on the edit buffer to current project.
ExtGetEditBuff	aFE6	Get current instrument or effect tone data on the edit buffer.
ExtSetEditBuff	aFE7	Set instrument or effect one data on the edit buffer.
ExtGetProject	aFE8	Get project data.
ExtSetProject	aFE9	Set project data.

Typedef

The following typedef is used after the next chapter.

Fig 5

New Name	Basic Type
WORD	unsigned short
DWORD	unsigned long

3. Get, Set type function

GetVersion

Prototype	char[] GetVersion(void);
Command	aFG0<CR+LF>

Parameter
None

Return
aFrame firmware version information<CR+LF>

ex. VER.1.10-BLD.001 2017/08/22 12:38:12

GetMode

Prototype	short GetMode(void);
Command	aFG1<CR+LF>

Parameter
None

Return
Mode<CR+LF>

A list of return values is shown in the table below.
See the Appendix A for a description of each mode.

Fig 6

Mode	Value
MODE_HOME	0
MODE_EDIT_INST_SE	1
MODE_EDIT_EFFECT_SE	2
MODE_EDIT_INST	3
MODE_EDIT_EFFECT	4
MODE_GROUP_SELECT	5
MODE_LEVEL_INPUT	6
MODE_LEVEL_OUTPUT	7
MODE_SYSTEM	8
MODE_EDIT_EXT	9

SetExtModeSw

Prototype	int SetExtModeSw(WORD SW);
Command	aFS1:SW<CR+LF>

Parameter
SW
0:OFF (MODE_EDIT_EXT to MODE_HOME)
1:ON (MODE_HOME to MODE_EDIT_EXT)

Return
0:OK or 1:NG<CR+LF>

GetSwitch

Prototype	WORD GetSwitch(void);
Command	aFG2<CR+LF>

Parameter
None

Return
SW_Data<CR+LF>

A list of return values is shown in the table below.

Fig 7

SW Name	Allocated Bit	Supplement
SW_OFF	0x0000	
SW_L1	0x0001	
SW_L2	0x0002	
SW_L3	0x0004	
SW_L4	0x0008	
SW_L5	0x0010	
-	0x0020	
SW_CUR_DOWN	0x0040	
SW_CUR_UP	0x0080	
SW_R1	0x0100	
SW_R2	0x0200	
SW_R3	0x0400	
SW_R4	0x0800	
SW_R5	0x1000	
SW_ENC	0x2000	
-	0x4000	
SW_MUTE	0x8000	REMOTE CTRL only

SetSwitch

Prototype	int SetSwitch(WORD SW_Data);
Command	aFS2:SW_Data<CR+LF>

Parameter
SW_Data
A list of return values is shown in the table Fig 7 above.

Return
0:OK<CR+LF>

GetEncoder

Prototype	short GetEncoder(void);
Command	aFG3<CR+LF>

Parameter
None

Return
Enc_Data<CR+LF>

The value from -32768 to 32767 are returned.

SetEncoder

Prototype	int SetEncoder(Short Enc_Data);
Command	aFS3:Enc_Data<CR+LF>

Parameter
Enc_Data
The value range is -32768 to 32767.

Return
0:OK<CR+LF>

GetLED

Prototype	DWORD GetLED(void);
Command	aFG4<CR+LF>

Parameter

None

Return

LED_Data<CR+LF>

A list of return values is shown in the table below.

Fig 8

LED Name	Allocated Bit	Supplement
LED_OFF	0x00000000	
LED_L1_RED	0x00000001	
LED_L1_GREEN	0x00000002	
LED_L2_RED	0x00000004	
LED_L2_GREEN	0x00000008	
LED_L3_RED	0x00000010	
LED_L3_GREEN	0x00000020	
LED_L4_RED	0x00000040	
LED_L4_GREEN	0x00000080	
LED_L5_RED	0x00000100	
LED_L5_GREEN	0x00000200	
LED_RMTC_RED	0x00000400	REMOTE CTRL: MUTE/PEAKC
LED_RMTC_GREEN	0x00000800	REMOTE CTRL: ----/PEAKC
LED_PEAKC_RED	0x00001000	
LED_PEAKC_GREEN	0x00002000	
LED_PEAKE_RED	0x00004000	
LED_PEAKE_GREEN	0x00008000	
LED_R1_RED	0x00010000	
LED_R1_GREEN	0x00020000	
LED_R2_RED	0x00040000	
LED_R2_GREEN	0x00080000	
LED_R3_RED	0x00100000	
LED_R3_GREEN	0x00200000	
LED_R4_RED	0x00400000	
LED_R4_GREEN	0x00800000	
LED_R5_RED	0x01000000	
LED_R5_GREEN	0x02000000	
LED_RMTE_RED	0x04000000	REMOTE CTRL: MUTE/PEAKE
LED_RMTE_GREEN	0x08000000	REMOTE CTRL: ----/PEAKE
LED_ENCR_RED	0x10000000	
LED_ENCR_GREEN	0x20000000	
LED_ENCL_RED	0x40000000	
LED_ENCL_GREEN	0x80000000	

SetLED

Prototype	int SetLED(DWORD LED_Data);
Command	aFS4:LED_Data<CR+LF>

Parameter

LED_Data

A list of return values is shown in the table Fig 8 above.

Return

0:OK<CR+LF>

GetLCD

Prototype	char[] GetLCD(WORD addr, WORD num);
Command	aFG5:addr,num<CR+LF>

Parameter

A list of parameter values is shown in the table below.

Fig 9

Parameter Name	Value Range	Supplement
addr	0 - 63	0 - 15: Line1 32 - 47: Line2
num	1 - 64	

Return

ASCII code Strings<CR+LF>

SetLCD

Prototype	int SetLCD(WORD addr, char[] strings);
Command	aFS5:addr,strings<CR+LF>

Parameter

A list of parameter values is shown in the table below.

Fig 10

Parameter Name	Value Range	Supplement
addr	0 - 63	0 - 15: Line1 32 - 47: Line2
strings	0x20 - 0x7E	ASCII Code

Return

0:OK<CR+LF>

GetGroupBank

Prototype	WORD GetGroupBank(void);
Command	aFG6<CR+LF>

Parameter

None

Return

Group Bank<CR+LF>

SetGroupBank

Prototype	int SetGroupBank(WORD Up_Down);
Command	aFS6:Up_Down<CR+LF>

Parameter

Up_Down

1: Group Bank Up

0: Group Bank Down

[note] This command is active in the MODE_HOME only.

Return

0:OK or 1:NG<CR+LF>

GetGroupNum

Prototype	WORD GetGroupNum(void);
Command	aFG7<CR+LF>

Parameter
None

Return
Group Number<CR+LF>

SetGroupNum

Prototype	int SetGroupNum(WORD Up_Down);
Command	aFS7:Up_Down<CR+LF>

Parameter
Up_Down
1: Group Bank Up
0: Group Bank Down
[note] This command is active in the MODE_HOME only.

Return
0:OK or 1:NG<CR+LF>

GetPeakLevel

Prototype	char[] GetPeakLevel(void);
Command	aFG8<CR+LF>

Parameter
None

Return
PeakInCenter, PeakInEdge, PeakOutLch, PeakOutRch<CR+LF>

The value range is from 0 to 15.

GetGetPressure

Prototype	char[] GetGetPressure(void);
Command	aFG9<CR+LF>

Parameter
None

Return
PressurePitch, PressureMute<CR+LF>

The value range is from 0 to 15.

GetCurrentGroupToneNum

Prototype	char[] GetCurrentGroupToneNum(void);
Command	aFGA<CR+LF>

Parameter
None

Return
group, number, max, inst_num, effect_num<CR+LF>

The value range is shown in the table below.

Fig 11

Name	Description	Value Range
group	Group number. 0: A 1: B 2: C 3: D 4: A' 5: B' 6: C' 7: D'	0 - 7
number	Tone number.	0 - (Max - 1)
max	Number of tones of the selected group.	1 - 40 (default: 10)
inst_num	Instrument number	0 - 79
effect_num	Effect number.	0 - 79

GetCurrentToneName

Prototype	char[] GetCurrentToneName(void);
Command	aFGB<CR+LF>

Parameter
None

Return
IXX:inst_name<CR+LF>
EXX:effect_name<CR+LF>
[note] XX means any one of 0 - 79.

GetCurrentToneData

Prototype	char[] GetCurrentToneData(WORD sel);
Command	aFGC:sel<CR+LF>

Parameter
sel is selecting instrument and effect.
0: Instrument
1: Effect

Return
algo_num<CR+LF>
name<CR+LF>
prm_num<CR+LF>
Data[prm_num]<CR+LF>

The value range is shown in the table below.

Fig 12

Name	Description	Value Range
algo_num	Algorithm number.	0 - 6
name	Name.	-
prm_num	Number of the parameter	Depend on Algorithm
Data[prm_num]	Parameter values ex. 0,1,2,3...63<CR+LF>	Depend on parameters

GetProjectGroupList

Prototype	char[] GetProjectGroupList(WORD group);
Command	aFGD:group<CR+LF>

Parameter

group means in the table below.

Fig 13

parameter	Value Range
group	0 - 7 0: A 1: B 2: C 3: D 4: A' 5: B' 6: C' 7: D'

Return

Gmax:max<CR+LF>
inst_num_1,effect_num_1<CR+LF>
:
:
inst_num_(max-1),effect_num_(max-1)<CR+LF>
:
inst_num_38,effect_num_38<CR+LF>
inst_num_39,effect_num_39<CR+LF>

} Selected group's active range.

GetProjectToneNameList

Prototype	char[] GetProjectToneNameList(WORD sel);
Command	aFGE:sel<CR+LF>

Parameter
sel is selecting instrument and effect.
0: Instrument
1: Effect

Return
Tone_ Name_0<CR+LF>
Tone_ Name_1<CR+LF>
:
:
Tone_ Name_79<CR+LF>

GetProjectToneDataList

Prototype	char[] GetProjectToneDataList(WORD sel, Short num);
Command	aFGF:sel,num<CR+LF>

Parameter
sel is selecting instrument and effect.
0: Instrument
1: Effect

num is instrument/effect tone number.
The value range is 0 to 79.
-1 means all tone data.

Return
algo_num<CR+LF>
name<CR+LF>
prm_num<CR+LF>
Data[prm_num]<CR+LF>

[note] If num is -1, repeat above data 80 times.
The value range is shown in the table below.

Fig 14

Name	Description	Value Range
algo_num	Algorithm number.	0 - 6
name	Name.	-
prm_num	Number of the parameter	Depend on Algorithm
Data[prm_num]	Parameter values ex. 0,1,2,3...63<CR+LF>	Depend on parameters

4. External Editing type function

[note] See Appendix B and Appendix C for data structure and flow.

ExtSelectGroup

Prototype	int ExtSelectGroup(WORD group, WORD num, WORD lcd);
Command	aFE0:group,num,lcd<CR+LF>

Parameter

Fig 15

parameter	Value Range
group	0 - 7 0: A 1: B 2: C 3: D 4: A' 5: B' 6: C' 7: D'
num	0 - 39
lcd	0:OFF 1:ON

Execution Result

dspMemoryGrp = group;
dspMemoryNum = num;
InstPatchSel = dspMemory[group][num].patch1;
EffectPatchSel = dspMemory[group][num].patch2;
set InstPatchEdit from InstPatch[InstPatchSel]
set EffectPatchEdit from EffectPatch[EffectPatchSel]

Return

0:OK or 1:NG or -1:Not MODE_EDIT_EXT<CR+LF>

ExtWriteGroup

Prototype	int ExtWriteGroup(WORD group, WORD num, WORD max, WORD lcd);
Command	aFE1:group,num,max,lcd<CR+LF>

Parameter

Fig 16

parameter	Value Range
group	0 - 7 0: A 1: B 2: C 3: D 4: A' 5: B' 6: C' 7: D'
num	0 - 39
max	1 - 40
lcd	0:OFF 1:ON

Execution Result

dspMemoryGrp = group;
dspMemoryNum = num;
dspMemoryMax[group] = max;
dspMemory[group][num].patch1 = InstPatchSel;
dspMemory[group][num].patch2 = EffectPatchSel;

Return

0:OK or 1:NG or -1:Not MODE_EDIT_EXT<CR+LF>

ExtChangeToneNum

Prototype	int ExtChangeToneNum(WORD sel, WORD num, WORD lcd);
Command	aFE2:sel,num,lcd<CR+LF>

Parameter

Fig 17

parameter	Value Range
sel	0: Inst 1: Effect
num	0 - 79
lcd	0:OFF 1:ON

Execution Result

```
if (sel == Inst)
    InstPatchSel = num;
    set InstPatchEdit from InstPatch[num]

if (sel == Effect)
    EffectPatchSel = num;
    set EffectPatchEdit from EffectPatch[num]
```

Return

0:OK or 1:NG or -1:Not MODE_EDIT_EXT<CR+LF>

ExtChangeEditBuffParam

Prototype	int ExtChangeEditBuffParam(WORD sel, WORD num, short prm, WORD lcd);
Command	aFE3:sel,num,prm,lcd<CR+LF>

Parameter

Fig 18

parameter	Value Range
sel	0: Inst 1: Effect
num	0 - 79
prm	-32768 to 32767
lcd	0:OFF 1:ON

Execution Result

```
if (sel == Inst)
    InstPatchEdit.Data[num] = prm;

if (sel == Effect)
    EffectPatchEdit.Data[num] = prm;
```

Return

0:OK or 1:NG or -1:Not MODE_EDIT_EXT<CR+LF>

ExtChangeEditBuffName

Prototype	int ExtChangeEditBuffName(WORD sel, WORD lcd, char[] strings);
Command	aFE4:sel,lcd,strings<CR+LF>

Parameter

Fig 19

parameter	Value Range
sel	0: Inst 1: Effect
lcd	0:OFF 1:ON
strings	Tone Name String (ASCII Code)

Execution Result

```
if (sel == Inst)
    InstPatchEdit.name[] = "string"; // INST

if (sel == Effect)
    EffectPatchEdit.name[] = "string"; // EFFECT
```

Return

0:OK or 1:NG or -1:Not MODE_EDIT_EXT<CR+LF>

ExtWriteEditBuffToProject

Prototype	int ExtWriteEditBuffToProject(WORD sel, WORD num, WORD lcd);
Command	aFE5:sel,num,lcd<CR+LF>

Parameter

Fig 20

parameter	Value Range
sel	0: Inst 1: Effect
num	0 - 79
lcd	0:OFF 1:ON

Execution Result

if (sel == Inst)
write InstPatch[num] from current InstPatchEdit

if (sel == Effect)
write EffectPatch[num] from current EffectPatchEdit

Return

0:OK or 1:NG or -1:Not MODE_EDIT_EXT<CR+LF>

ExtGetEditBuff

Prototype	char[] ExtGetEditBuff(WORD sel, WORD mode, WORD lcd);
Command	aFE6:sel,mode,lcd<CT+LF>

Parameter

Fig 21

parameter	Value Range
sel	0: Inst 1: Effect
mode	0:BIN 2:TXT
lcd	0:OFF 1:ON

Execution Result

```
[data]:mode0(BIN)
C0 00 . . . . . .                    // HEX(raw data)
<---> <-------------------------->
size      data[size]                 // data[size-2:size-1] = checkSum

[data]:mode1(TXT)
0<CR+LF>                             // algo_num
Harmo Drum<CR+LF>                    // name
67<CR+LF>                            // prm_num
50,0,,,,,,,,,,,,,,,,,,,,,,,,64,<CR+LF>   // data[prm_num]
5117<CR+LF>                          // checkSum
```

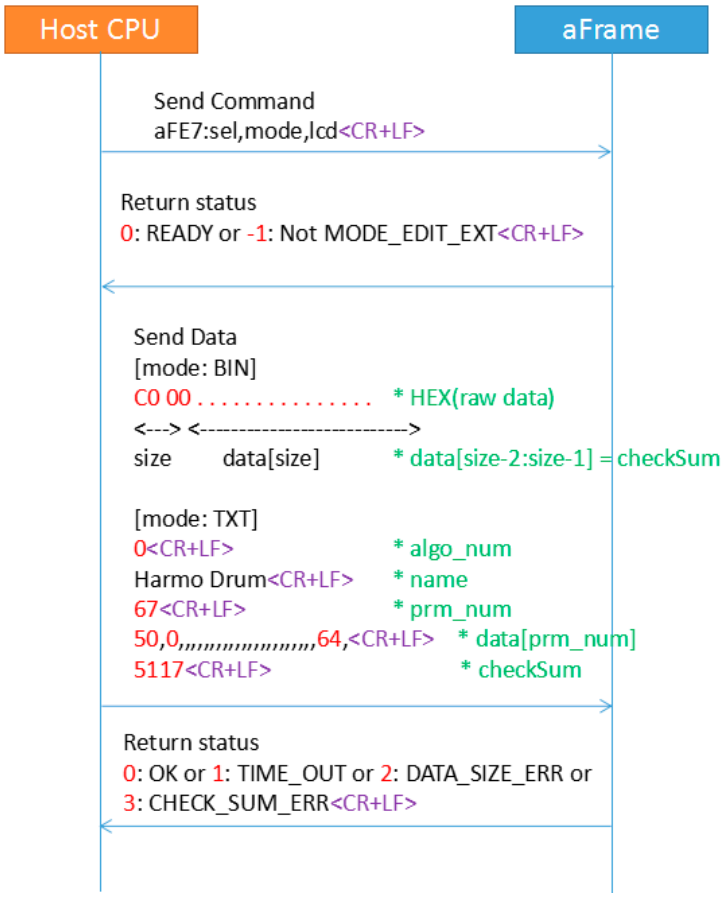
Return

Data or -1:Not MODE_EDIT_EXT<CR+LF>

ExtSetEditBuff

Prototype	int ExtSetEditBuff(WORD sel, WORD mode, WORD lcd)
Command	aFE7:sel,mode,lcd<CR+LF>

This function command has 2 phase. The sequence is shown below.



Parameter

Fig 22

parameter	Value Range
sel	0: Inst 1: Effect
mode	0:BIN 2:TXT
lcd	0:OFF 1:ON

Execution Result

```
[data]:mode0(BIN)                    // same as aFE6
[data]:mode1(TXT)                    // same as aFE6

if (sel == Inst)
    set data to InstPatchEdit

if (sel == Effect)
    set data to EffectPatchEdit
```

Return

[Command Phase]
0:READY or -1:Not MODE_EDIT_EXT<CR+LF>

[Data Phase]
0:OK or 1:TIME_OUT or 2:DATA_SIZE_ERR or 3:CHECK_SUM_ERR<CR+LF>

ExtGetProject

Prototype	char[] ExtGetProject(WORD wait, WORD lcd);
Command	aFE8:wait,lcd<CR+LF>

Parameter

Fig 23

parameter	Value Range
wait	0 - 1000 [ms] Wait time[ms] each 1024byte
lcd	0:OFF 1:ON

Execution Result

```
[data]:(BIN)
00 7F . . . . . .                    // HEX(LZSS encode data)
<---> <--------------------------->
size        data[size]

* SET checkSum
* DspPrj.chbuf[DSP_PROJECT_BUF_SIZE] -> encode_LZSS() -> data[size]
```

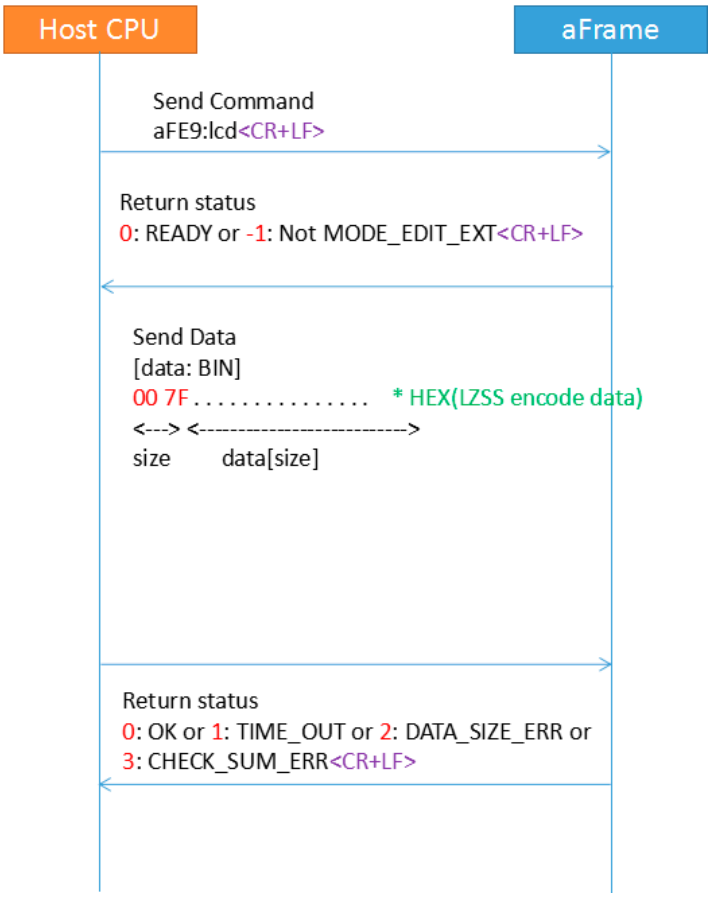
Return

Data or -1:Not MODE_EDIT_EXT<CR+LF>
[note] Received data is encoded with LZSS. It is need to decode at the Host CPU side.

ExtSetProject

Prototype	int ExtSetProject(WORD lcd);
Command	aFE9:lcd<CR+LF>

This function command has 2 phase. The sequence is shown below.
On the Data sending phase, data is need to encode with LZSS.



Parameter

Fig 24

parameter	Value Range
lcd	0:OFF 1:ON

Execution Result

[data]:(BIN) // same as aFE8

* data[size] -> decode_LZSS() -> DspPrj.chbuf[DSP_PROJECT_BUF_SIZE]
* TEST DspPrj.tb.checkSum

* set InstPatchEdit from DspPrj.tb.InstPatch[DspPrj.tb.InstPatchSel]
* set EffectPatchEdit from DspPrj.tb.EffectPatch[DspPrj.tb.EffectPatchSel]

Return

[Command Phase]

0:READY or -1:Not MODE_EDIT_EXT<CR+LF>

[Data Phase]

0:OK or 1:TIME_OUT or 2:DATA_SIZE_ERR or 3:CHECK_SUM_ERR<CR+LF>

Appendix A. Mode

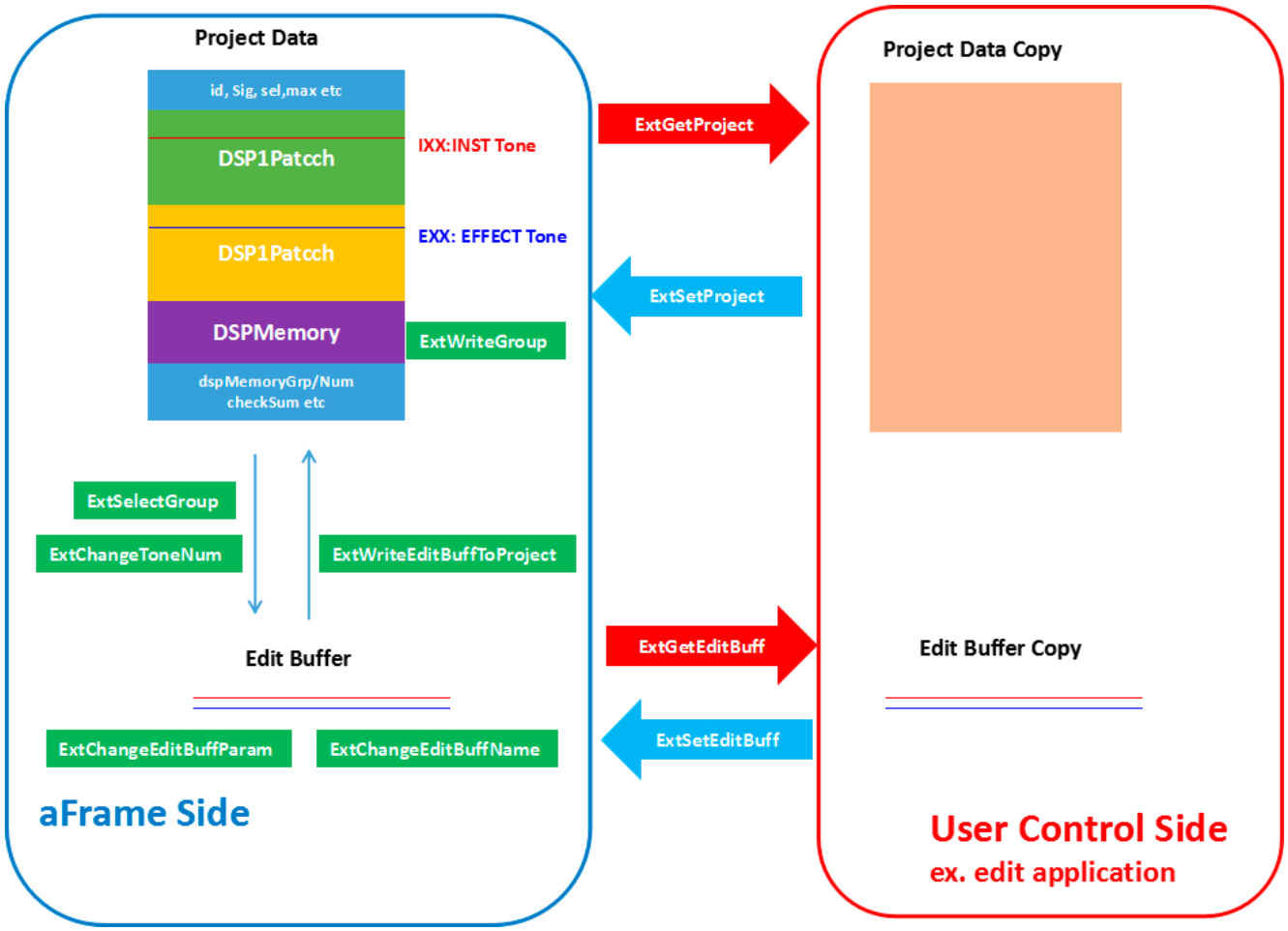
aFrame has a 10 mode.

Fig 25

Mode	Value	Description
MODE_HOME	0	Other than below mode.
MODE_EDIT_INST_SE	1	Editing Instument simply.
MODE_EDIT_EFFECT_SE	2	Editing Effect simply.
MODE_EDIT_INST	3	Editing Instrument. [1 PITCH] + [2 DECAY]
MODE_EDIT_EFFECT	4	Editing Effect. [3 BEND] + [4VOLUME]
MODE_GROUP_SELECT	5	Editing Group Select. [2 DECY] + [3 BEND]
MODE_LEVEL_INPUT	6	Displaying Input Level meter. [1 PITCH] + [3 BEND]
MODE_LEVEL_OUTPUT	7	Displaying Output Level meter. [2 DECAY] + [4 VOLUME]
MODE_SYSTEM	8	Editing System Parameter. [1 PITCH] + [4 VOLUME]
MODE_EDIT_EXT	9	External Editing Mode In this mode, aFrame hardware cannot edit.

Appendix B. External Editing Mode

On the MODE_EDIT_EXT mode, user can control aFrame using External Editing functions. The data diagram of Function execution result is shown below.



Appendix C. Project Data Structure

Union and Struct define is shown below.

```
/*****************************************************************************/
#define DSP_PROJECT_BUF_SIZE    0x7F00
#define INST_PATCH_NUM   80
#define EFFECT_PATCH_NUM        80
#define DSP_MEMORY_NUM          40
#define DSP_MEMORY_GRP8
/*****************************************************************************/
typedef struct _DSP_PATCH {
    short           algo_num;
    char name[20];
    short           prm_num;
    short           Data[84];
} DSP_PATCH;        // 192byte(48long)
/*****************************************************************************/
typedef struct _DSP_MEMORY {
    short           patch1;
    short           patch2;
} DSP_MEMORY;     // 4byte(1long)
/*****************************************************************************/
typedef union _DSP_PROJECT {
    struct {
        long            ID;                                     // 4
        char            name[28];                               // 28
        char            Signature[24];                          // 24
        short           InstPatchSel;                           // 2
        short           EffectPatchSel;                         // 2
        short           InstPatchMax;                           // 2
        short           EffectPatchMax;                         // 2
        DSP_PATCH       InstPatch[INST_PATCH_NUM];              // 15360 (192*80)
        DSP_PATCH       EffectPatch[EFFECT_PATCH_NUM];          // 15360 (192*80)
        DSP_MEMORY      dspMemory[DSP_MEMORY_GRP][DSP_MEMORY_NUM];   // 1280 (4*8*40)
        short           dspMemoryGrp;                           // 2
        short           dspMemoryNum;                           // 2
        short           dspMemoryMax[DSP_MEMORY_GRP];           // 16
        char            chDummy[424];                           // 424
        long            checkSum;                               // 4
    } tb;                       // 32512 (0x7F00)
    char                chbuf[DSP_PROJECT_BUF_SIZE];
    short               sbuf[DSP_PROJECT_BUF_SIZE/2];
    long                lbuf[DSP_PROJECT_BUF_SIZE/4];
} DSP_PROJECT;
/*****************************************************************************/
DSP_PROJECT DspPrj;
DSP_PATCH       InstPatchEdit;
DSP_PATCH       EffectPatchEdit;
/*****************************************************************************/
```