General Notice

When using this document, keep the following in mind:

1. This document is confidential. By accepting this document you acknowledge that you are bound by the terms set forth in the nondisclosure and confidentiality agreement signed separately and in the possession of SEGA. If you have not signed such a nondisclosure agreement, please contact SEGA immediately and return this document to SEGA.

2. This document may include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new versions of the document. SEGA may make improvements and/or changes in the product(s) and/or the program(s) described in this document at any time.

3. No one is permitted to reproduce or duplicate, in any form, the whole or part of this document without SEGA's written permission. Request for copies of this document and for technical information about SEGA products must be made to your authorized SEGA Technical Services representative.

4. No license is granted by implication or otherwise under any patents, copyrights, trademarks, or other intellectual property rights of SEGA Enterprises, Ltd., SEGA of America, Inc., or any third party.

5. Software, circuitry, and other examples described herein are meant merely to indicate the characteristics and performance of SEGA's products. SEGA assumes no responsibility for any intellectual property claims or other problems that may result from applications based on the examples described herein.

6. It is possible that this document may contain reference to, or information about, SEGA products (development hardware/software) or services that are not provided in countries other than Japan. Such references/information must not be construed to mean that SEGA intends to provide such SEGA products or services in countries other than Japan. Any reference of a SEGA licensed product/program in this document is not intended to state or imply that you can use only SEGA's licensed products/programs. Any functionally equivalent hardware/software can be used instead.

7. SEGA will not be held responsible for any damage to the user that may result from accidents or any other reasons during operation of the user's equipment, or programs according to this document.

NOTE: A reader's comment/correction form is provided with this document. Please address comments to :

SEGA of America, Inc., Developer Technical Support (att. Evelyn Merritt)
150 Shoreline Drive, Redwood City, CA 94065

SEGA may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

(11/2/94- 002)

SEGA™

SEGA OF AMERICA, INC.
Consumer Products Division

Branching Playback Library User's Manual

Doc. # ST-136-D-R2-082495

READER CORRECTION/COMMENT SHEET

Keep us updated!

If you should come across any incorrect or outdated information while reading through the attached document, or come up with any questions or comments, please let us know so that we can make the required changes in subsequent revisions. Simply fill out all information below and return this form to the Developer Technical Support Manager at the address below. Please make more copies of this form if more space is needed. Thank you.

General Information:

Your Name ______________________ **Phone** ______________

Document number ST-136-D-R2-082495 **Date** ______________

Document name Branching Playback Library User's Manual

Corrections:

Chpt.	pg. #	Correction

Questions/comments: ______________________

____

____

____

Where to send your corrections:

Fax:	(415) 802-1717 Attn: Evelyn Merritt, Developer Technical Support	Mail:	SEGA OF AMERICA Attn: Evelyn Merritt, Developer Technical Support 150 Shoreline Dr. Redwood City, CA 94065

1. Overview

The Branching Playback Library (BPL) enables seamless reading of data streams, based on a pre-defined scenario. This allows the system to branch between streams smoothly during reads.

The BPL, however, manages only the data streams that are necessary for branching. Use a decode-only library such as MPEG and Cinepak in conjunction with BPL to play back data such as audio and video.

1.1 Organization of the Library

Figure 1.1 shows the organization of CD-related libraries.

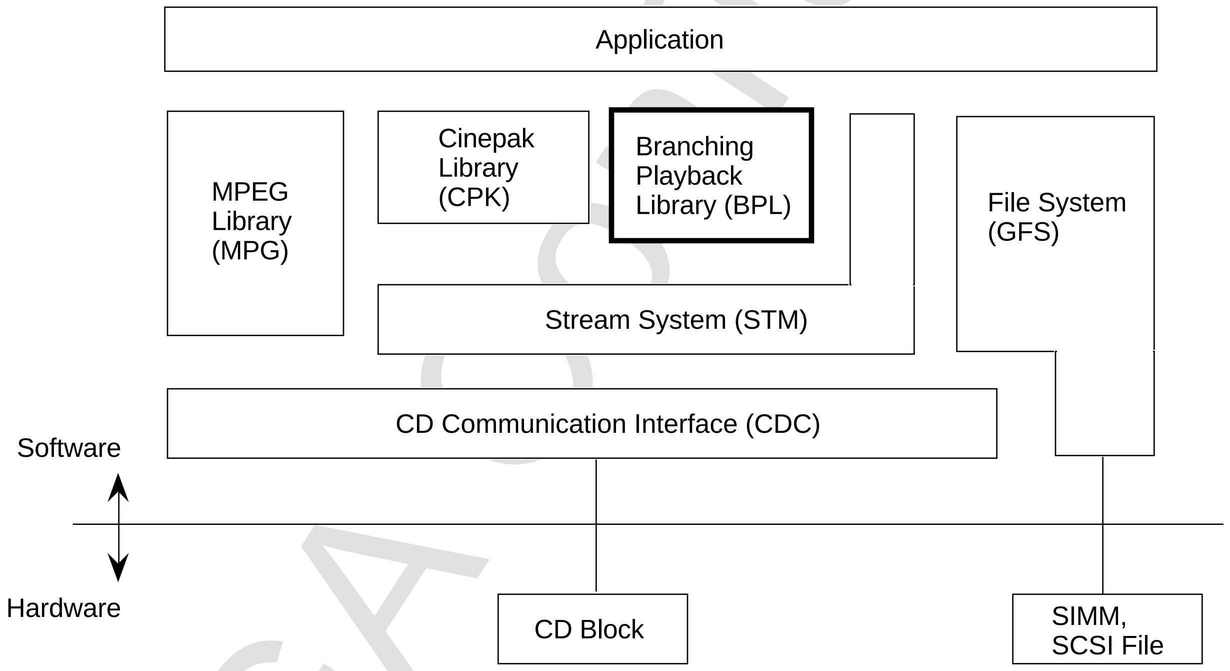


Figure 1.1 Organization of CD-related libraries

The Branching Playback Library requires each of the following libraries: Stream System, File System, and CD Communication Interface.

1.2 Summary of Branching Playback Library Functions

1. Setting the Branch Destination (Scenario) Information
This function sets destination stream candidates as destination (scenario) information.

2. Pre-reading the Streams Necessary for Branching
The BPL manages the opening/closing of streams to smoothly branch between streams. By pre-reading an open stream (a branching destination stream candidate) into the CD buffer, the stream can be fetched without interruption when the branch destination is determined.

3. Destination Selection Function
The BPL selects the actual destination from the destination candidates.

4. Destination Stream Notification Function
Based on the selected destination, the BPL notifies the application of the next stream to be played.

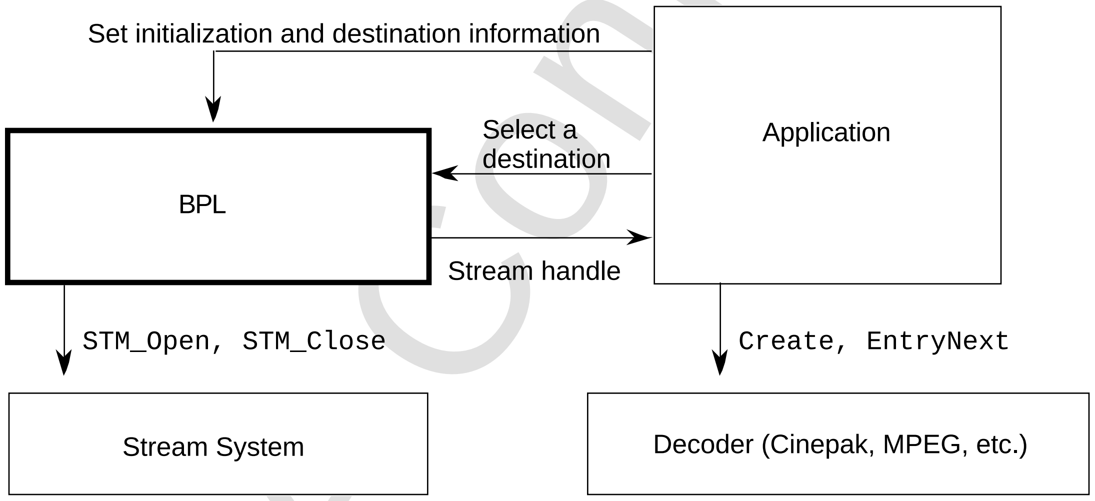


Figure 1.2 Stream system overview diagram

2. Basic Items

2.1 Definitions

Table 2.1 Terminology

Term	Meaning
Branch stream	Equivalent to a file on a CD. The BPL reads a stream based on a scenario that is set for a branch stream. Different types of stream data (e.g., audio, video) can be fetched by performing channel-interleaving within a branch stream (normally, interleaving by means of a subheader).
Branch stream ID	This ID identifies the branch stream. Given this ID, the read file, stream key, or destination information can be set or fetched.
Branch number	This number specifies the branch destination. Equivalent to the event types such as input from a control pad.

Table 2.2 List of abbreviations

Abbreviation	Meaning	Description
`BPL`	branch play	branching playback
`bstm`	branch stream	branch stream
`bstmid`	branch stream ID	branch stream ID
`brno`	branch No.	branch No.
`bstmmax`	branch stream max	Total number of branch streams
`brmax`	branch max	Total number of branches

Other terms that appear in this manual are based on the CD Communication Interface, the File System, and the Stream System Libraries.

2.2 Restrictions on Names

The BPL uses the following function, variable, type, and macro names:

Function/variable name:	`BP~` and `bp~`
Type name:	`Bp~`
Macro name:	`BP~`

The libraries required by the BPL use the following global symbols:

Table 2.3 Symbol names and libraries

Library Name	Symbol
Stream System	`ST~, st~, St~`
File System	`GF~, gf~, Gf~`
CD Communication Interface	`CD~, cd~, Cd~`

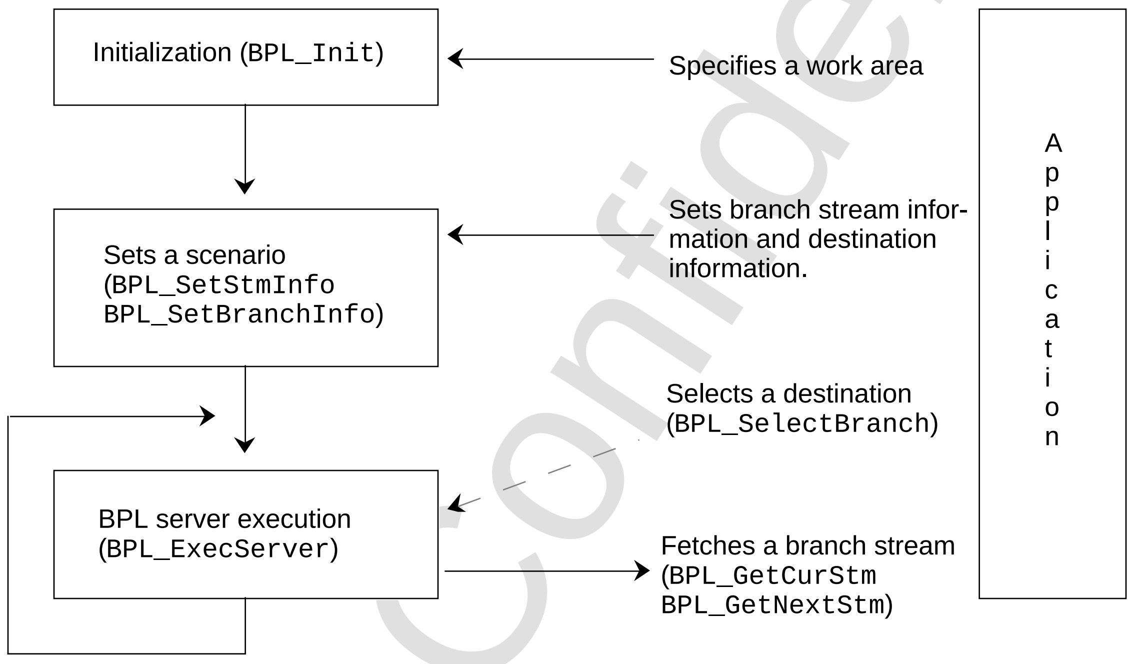
These symbols must not be used by the application program.

3. How the BPLWorks

3.1 Flow of Processing

The BPL reads a stream according to a given scenario and notifies the application of the stream handle that must be decoded.

Figure 3.1 shows the flow of main processing events.



Initialization (BPL_Init)
Specifies a work area
Sets a scenario (BPL_SetStmInfo BPL_SetBranchInfo)
Sets branch stream information and destination information.
Selects a destination (BPL_SelectBranch)
BPL server execution (BPL_ExecServer)
Fetches a branch stream (BPL_GetCurStm BPL_GetNextStm)
Application

Figure 3.1 Flow of processing

3.2 Scenario

A scenario is information that indicates how branching playback is to be performed as a function of time (the order in which streams are to be played).

Branched streams are specified in file units. Audio and video data can be fetched by channel-interleaving within a file.

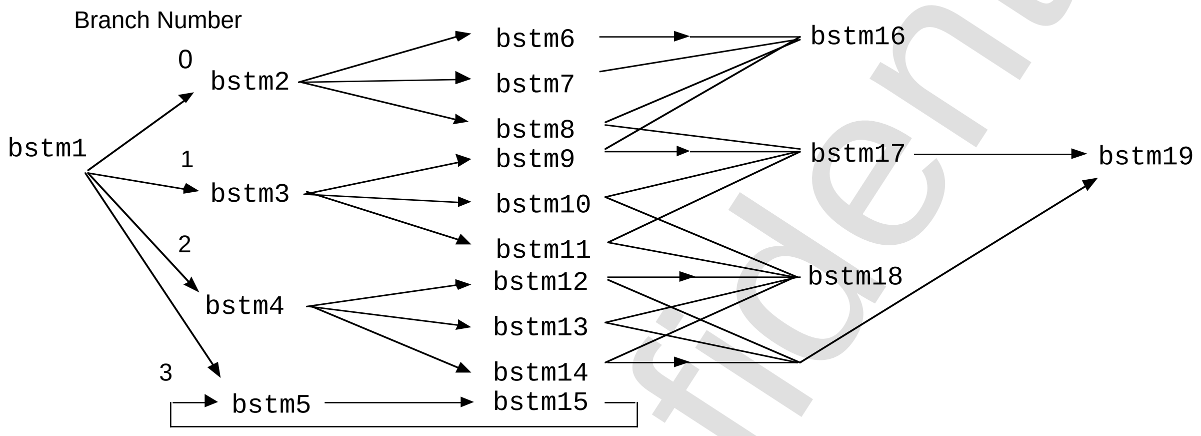


Figure 3.2 Stream-branching

(a) This scenario specifies `bstm1` as the branch stream to be read first. The BPL then starts reading `bstm1`.

(b) The application fetches the branch stream that is currently being read and sets it in the decoder.

(c) After reading `bstm1`, the BPL begins reading branch candidates (branch streams that may be fetched next) `bstm2`, `bstm3`, `bstm4`, and `bstm5`. Effective use of the CD buffer and smoother branching is made possible by pre-reading branch candidate streams.

(d) The application fetches events such as input from a control pad and selects the destination for branching. If branch numbers 0~3 are assigned to branches `bstm2`, `bstm3`, `bstm4`, and `bstm5`, and if 1 is specified, reading of `bstm2`, `bstm4` and `bstm5`, which is no longer needed, is canceled. If necessary, the application fetches the destination stream and sets it in the decoder.

(e) After fetching `bstm1`, the BPL begins fetching `bstm3`.
If the application specifies the execution of branching to the Branch Play Server, the BPL begins reading `bstm9`, `bstm10`, and `bstm11`, as in (c).

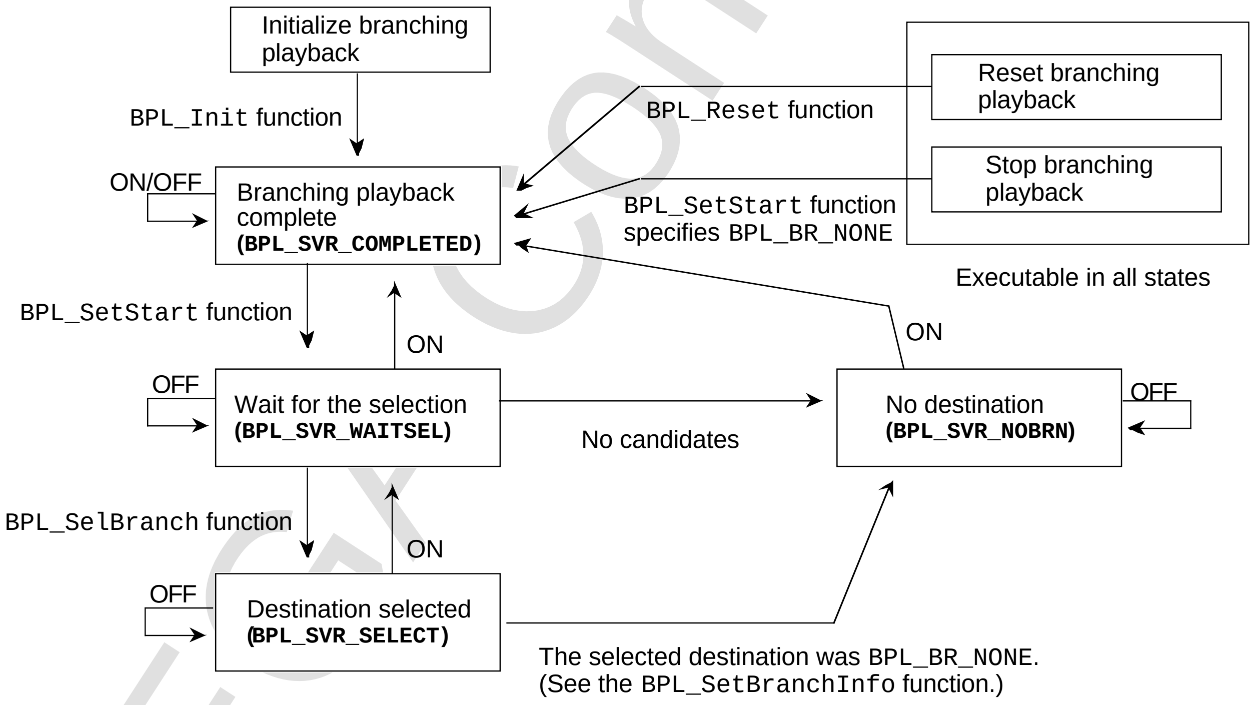
3.3 Changing Branching Playback States

Table 3.1 shows branching playback states. Figure 3.1 shows a branching playback state transition diagram.

Table 3.1 Branching playback states

State	Description
End of branching playback	Branching playback ended. The stream group and the streams that were opened by the BPL (the current stream and candidate streams) are all closed.
Wait for a destination selection	Branch candidates were pre-read, but a destination has not been selected. All streams among the branch candidates are subject to pre-reading. Only the current stream can be accessed. Destination streams cannot be accessed.
Determine destination	A destination was selected from the branch candidates. Only the selected destination is pre-read. Both the current stream and the destination stream can be accessed.
No destination	There are no branch candidates or destinations for the current stream. The last stream is being played.

The server function can get the branching playback status.



Initialize branching playback
BPL_Init function
ON/OFF
Branching playback complete (BPL_SVR_COMPLETED)
BPL_Reset function
Reset branching playback
Stop branching playback
BPL_SetStart function specifies BPL_BR_NONE
Executable in all states
BPL_SetStart function
ON
ON
OFF
Wait for the selection (BPL_SVR_WAITSEL)
No candidates
No destination (BPL_SVR_NOBRN)
OFF
BPL_SelBranch function
ON
OFF
Destination selected (BPL_SVR_SELECT)
The selected destination was BPL_BR_NONE.
(See the BPL_SetBranchInfo function.)

- ON/OFF: Branch execution switch for the Branching Playback Server function BPL_ExecServer.
  If ON is specified in a destination selection wait or no destination state, then branching playback is terminated.
  If ON is specified in the selected selected state, then branch streams are changed.

Figure 3.3 BP state transition diagram

3.4 Executing Branching (Branch Stream-Switching)

(1) Executing branching

When branching is performed in the destination selected state (by turning on the branch execution switch of the Branching Playback Server), branch streams are switched as follows:

(a) The current stream, A, is closed.
(The BPL stops reading A and deletes any data that remains in the CD buffer.)
(b) The destination stream, B, becomes the current stream.
(c) The destination stream becomes undefined.

Table 3.2 Switching branch streams by executing branching

Branch Stream	Before Branching	After Branching (after switching)
Current stream (obtained by the `BPL_GetCurStm` function)	A	B (A is closed)
Destination stream (obtained by the `BPL_GetNextStm` function)	B	Undefined until the next destination is selected and determined by the `BPL_SelectBranch` function

The selection of a destination always precedes the execution of branching (switching). However, the selection and switching operations are generally performed asynchronously.

(2) Opening and closing a stream

The BPL opens both the current stream and branch candidate streams. The BPL employs the following opening and closing procedures:

(a) Starting playback stream specified by the `BPL_SetStart` function is opened first as the current stream.
(b) When the reading of the current stream begins, branch candidate streams are opened.
(c) When a destination is selected, all other branch candidates are closed, and only the destination is pre-read.
(d) When branching is executed, the current stream is closed.
The destination stream becomes the current stream, and steps(b)~(d) are repeated.
(e) When branching playback is completed, the stream group is closed.

(3) Timing for branch stream-switching

Table 3.3 shows the timing types for branch stream-switching.

Table 3.3 Timing for branch stream-switching

Timing	Description
Natural switching	Switches to destination stream B upon completion of decoding stream A.
Forced switching	Force switch to destination stream B regardless of whether stream A is being decoded.

Branching must not be executed until the decoder finishes processing the current stream; even when a destination is determined (to prevent truncation of the stream data that is being decoded).

Regardless of whether normal or forced switching is performed, switching processes for the decoder should be executed first. The branch execution switch should be turned on only after switching is complete.

4. Organization of Files on Disc

The total amount of streams that can be pre-read is limited by the capacity of the CD buffer (a maximum of 200 sectors). Therefore, streams that exceed the limit and can not be pre-read may result in branching delays.

1. Non-interleaved branch candidates

Suppose that A's branch candidates are B and C and that files are positioned on disc as shown in Figure 4.1. Then, only file B can be pre-read.

There will be no problems if the pre-read data of A is sufficient to seek/branch to B or C. However, if both B and C need to be pre-read in order to enable delayed branch selection timing, branching to C in this example cannot be performed without delay.

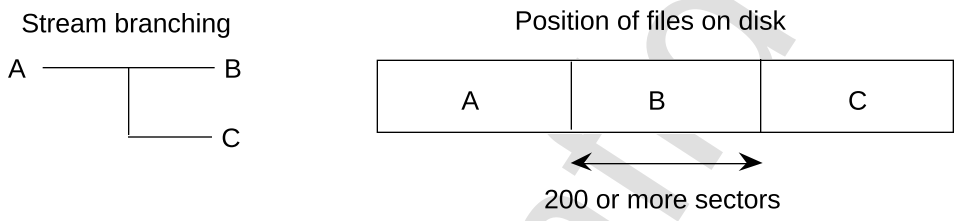
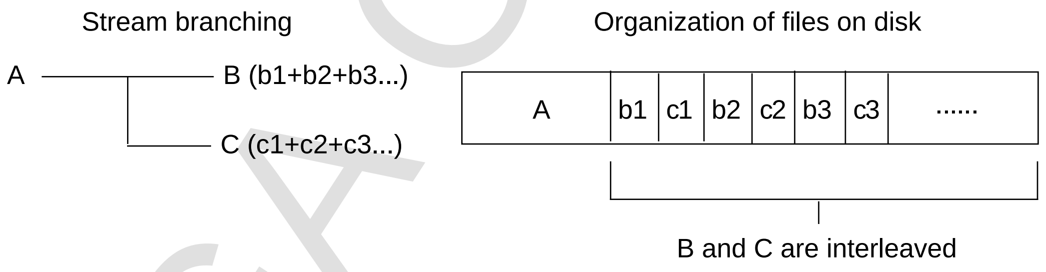


Figure 4.1 Non-interleaved candidate branches (C cannot be pre-read)

2. Interleaved branch candidates

As shown in Figure 4.2, one method of enabling branches to B and C without delay after A is played is to interleave B and C immediately after A.



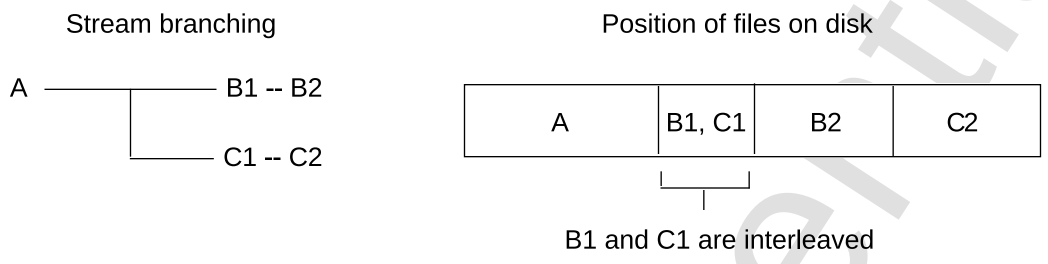
Note: Two branch candidate files exist: B and C.

Figure 4.2 Interleaved branch candidates (all of B and C)

3. Partially interleaved branch candidates

As shown in Figure 4.3, it is also possible to split B into B1 and B2, and C into C1 and C2 and to interleave only B1 and C1.

In this case, it is sufficient to interleave only parts of B and C (B1 and C1). This technique allows a seek to B2 and C2 and enables highly independent operation. However, the technique requires the division of files.



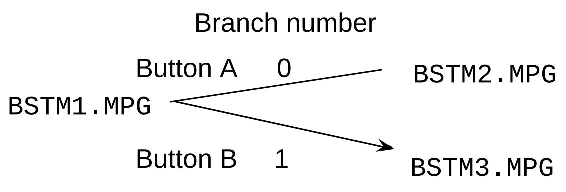
Note: Four branch candidate files exist: B1, B2, C1, and C2.

Figure 4.3 Partially-interleaved branch candidates (parts of B and C)

5. Basic Examples

5.1 Scenario Processing

Figure 5.1 shows an example of a branching playback scenario.



Button A is pressed while `BSTM1.MPG` is being played →`BSTM2.MPG` is played after `BSTM1.MPG`.
Button B is pressed while `BSTM1.MPG` is being played →`BSTM3.MPG` is played after `BSTM1.MPG`.

Figure 5.1 Example of a branching playback scenario

The following is a sample program that sets this scenario.

```
#define BSTM_MAX    3   /* Total number of branch streams(BSTM1.MPG,
                         BSTM2.MPG, BSTM3.MPG)  */
#define BRANCH_MAX  2   /* Total number of branches (number of arrows in
                         Figure 5.1) */
#define KEY_MAX     2   /* Total number of stream key types */
#define A_BTN       0   /* Branch number assigned to button A */
#define B_BTN       1   /* Branch number assigned to button B */
#define BR_NUM      2   /* Number of branches per stream */
#define BSTM1_ID    0   /* Branch stream ID of BSTM1.MPG */
#define BSTM2_ID    1   /* Branch stream ID of BSTM2.MPG */
#define BSTM3_ID    2   /* Branch stream ID of BSTM3.MPG */

/* Work area for the BPL */
Sint32 work_bpl[BPL_WORK_SIZE(BSTM_MAX, BRANCH_MAX, KEY_MAX)/sizeof(Sint32)];

void    setScenario(void)
{
      StmKey  key[KEY_MAX];   /* Area for setting a stream key */
      Sint32  brtbl[BR_NUM];  /* Area for setting a destination */
      Sint32  fid;            /* File ID */

      /* Initialization of branching playback */
      BPL_Init(BSTM_MAX, BRANCH_MAX, KEY_MAX, work_bpl);

      /* Setting branch stream information */
      STM_KEY_CN(key + 0) = STM_KEY_CIMSK(key + 0) = STM_KEY_NONE;
      STM_KEY_CN(key + 1) = STM_KEY_CIMSK(key + 1) = STM_KEY_NONE;
      STM_KEY_SMMSK(key + 0) = STM_KEY_SMVAL(key + 0) = STM_SM_VIDEO;
      STM_KEY_SMMSK(key + 1) = STM_KEY_SMVAL(key + 1) = STM_SM_AUDIO;
      fid = GFS_NameToId(“BSTM1.MPG”);
      BPL_SetStmInfo(BSTM1_ID, fid, KEY_MAX, key);
      fid = GFS_NameToId(“BSTM2.MPG”);
```

```
        BPL_SetstmInfo(BSTM2_ID, fid, KEY_MAX, key);
        fid = GFS_NameTold("BSTM3.MPG");
        BPL_SetStmInfo(BSTM3_ID, fid, KEY_MAX, key);

        /* Set destination information */
        brtbl[A_BTN] = BSTM2_ID;                        /* Branch to BSTM2.MPG
                                                        if button A is pressed */
        brtbl[B_BTN] = BSTM3_ID;                        /* Branch to BSTM3.MPG
                                                        if button B is pressed */
        BPL_SetBranchInfo(BSTM1_ID, BR_NUM, brtbl);     /* Set the destination for
                                                        BSTM1.MPG */
}
```

5.2 Branching Playback Processing

The following is an example of a branching playback program. (Refer to Section 5.1 for the scenario.)

```
Sint32   work_gfs[GFS_WORK_SIZE(BSTM_MAX*KEY_MAX)/sizeof(Sint32)];
Sint32   work_stm[STM_WORK_SIZE(GRP_MAX, BSTM_MAX*KEY_MAX)/sizeof(Sint32)];
Sint32   brno;                       /* Branch number */
StmHn    stmtbl[KEY_MAX];            /* Stream handle table */
Sint32   bpl_stat;                   /* Branching playback status */
Sint32   decode_stat;                /* Decoder operation status */
DecodeHn dc_hn = NULL;               /* Decoder handle */
Bool     chgsw = OFF;                /* Branch execution switch */
Bool     endflag = FALSE;
Sint32   ret;

/* Initialization of the libraries */
GFS_Init(···);                       /* Initialize the File System */
STM_Init(···);                       /* Initialize the Stream System */
initDecoder();                       /* Initialize the decoder */
setScenario();                       /* Set a scenario (see 5.1) */

/* Branching playback */
BPL_SetStart(BSTM1_ID);              /* Specify a stream to begin playback*/
BPL_GetCurStm(KEY_MAX, stmtbl);      /* Fetch the first branch stream */
dc_hn = createDecodeHn(stmtbl);      /* Create a decoder handle */
while (endflag == FALSE) {
    bpl_stat = BPL_ExecServer(chgsw);       /* Execute the Branching Playback
                                               Server */
    chgsw = OFF;
    STM_ExecServer();                       /* Execute the stream server */
    decode_stat = execDecoder(dc_hn);       /* Execute the server function of
the decoder */

    switch (bpl_stat) {
    case BPL_SVR_COMPLETED:                 /* Branching playback complete status */
        endflag = TRUE;
        break;
    case BPL_SVR_WAITSEL:                   /* Destination selection wait
                                               state */
        /* Get pad input (0:button A, 1:button B, negative: no input */
        brno = getPadEvent();
        if (brno >= 0) {
            BPL_SelectBranch(brno);         /* Select a destination */
        }
        break;
    case BPL_SVR_SELECT:                 /* Destination determined state */
    case BPL_SVR_NOBRN:                  /* No-destination state */
        if (decode_stat != COMPLETED) {  /* Decoding completion check */
            break;
        }
        chgsw = ON;                            /* Branch execution switch on */
        ret = BPL_GetNextStm(KEY_MAX, stmtbl); /* Get a destination stream */
        if (ret >= 0) {                        /* If there is a destination */
            destoroyDecodeHn(dc_hn);           /* Clear the decoder handle */
            dc_hn = createDecodeHn(stmtbl);    /* Create a decoder handle */
        }
        break;
    }
}
destoroyDecodeHn(dc_hn);            /* Clear the decoder handle */
```

The BPL automatically opens and closes a stream by using the Stream System. For a description of the decoder, refer to the applicable library manuals.

6. Data Specifications

6.1 Basic Data

Title	Data	Data Name	No.
Data specifications	Basic data		1.0

1. Basic Data Types

Type	Descripton
Uint8	Unsigned 1-byte integer
Sint8	Signed 1-byte integer
Uint16	Unsigned 2-byte integer
Sint16	Signed 2-byte integer
Uint32	Unsigned 4-byte integer
Sint32	Signed 4-byte integer
Bool	Boolean 4-byte integer (logical constants are used as Boolean)

2. Logical Constants

Logical constants are used as Boolean values:

Constant	Value	Description
FALSE	0	Represents the FALSE logical value.
TRUE	1	Represents the TRUE logical value.
OFF	0	Represents the switch off (FALSE) state.
ON	1	Represents the switch on (TRUE) state.

6.2 Constants

Title	Data	Data Name	No.
Data specifications	Constant		2.0

1. Error Codes

The value of `BPL_ERR_OK` is 0. Other error codes take negative values.

Constant	Description
`BPL_ERR_OK`	Normal termination
`BPL_ERR_KYOVRFLW`	Too many stream keys
`BPL_ERR_BROVRFLW`	Too many destination settings
`BPL_ERR_BSTMID`	Illegal branch stream ID
`BPL_ERR_BRNO`	Illegal branch number
`BPL_ERR_BRSPC`	Destination already specified
`BPL_ERR_NOKEY`	No corresponding stream key set
`BPL_ERR_OPNSTM`	Stream open failure

2. Other

Constant	Value	Description
`BPL_STMKEY_MAX`	6	Number of stream keys that can be set to a branch stream.

7. Function Specifications

Table 7.1 shows a list of BPL functions.

Table 7.1 List of functions (1)

Function		Function Name	No.
Scenario processing			1.0
	Initialize branching playback	`BPL_Init`	1.1
	Reset branching playback	`BPL_Reset`	1.2
	Set branch stream information	`BPL_SetStmInfo`	1.3
	Get branch stream information	`BPL_GetStmInfo`	1.4
	Set destination information	`BPL_SetBranchInfo`	1.5
	Get destination information	`BPL_GetBranchInfo`	1.6
Branching playback-processing			2.0
	Set playback start stream	`BPL_SetStart`	2.1
	Execute Branching Playback Server	`BPL_ExecServer`	2.2
	Select destination	`BPL_SelectBranch`	2.3
	Get current stream	`BPL_GetCurStm`	2.4
	Get destination stream	`BPL_GetNextStm`	2.5
	Get stream group	`BPL_GetStmGrp`	2.6

7.1 Scenario Processing

Title	Data	Data Name	No.
Data specifications	Initialize branching playback	`BPL_Init`	1.1

[Format] `Sint32 BPL_Init(Sint32 bstmmax, Sint32 brmax, Sint32 keymax, void *work)`

[Input]

`bstmmax`:	Total number of branch streams
`brmax`:	Total number of branches
`keymax`:	Total number of stream key types
`work`:	Work area

[Output] None

[Function value] Error code

[Function] Initializes the work area for the BPL. Clears previously set scenario information. Execute this function before the BPL is used.

[Remarks]

(a) Determine the size of the work area from the `BPL_WORK_SIZE (bstmmax`, `brmax`, `keymax`) byte.
Allocate work areas at 4-byte boundaries.
Example: `Uint32 work[BPL_WORK_SIZE(bstmmax, brmax, keymax)/sizeof(Uint32)];`

(b) When stream keys of different types are assigned to different branch streams, the sum of the types is the value of `keymax`.
Example: If `key1` (3 types of keys) is assigned to `bstm1`, and `key2` (4 types of keys) is assigned to `bstm2`, then `keymax`, which is the sum of `key1` and `key2`, will be 7.
If `key1` is assigned to both `bstm1` and `bstm2`, then `keymax`, which is `key1`, will be 3.

(c) The `BPL_Init` function does not close the stream group that is currently used. To force an initialization of the BPL while it is being used, execute the `BPL_Reset` function.

Title	Data	Data Name	No.
Data specifications	Reset branching playback	`BPL_Reset`	1.2

[Format] `Sint32 BPL_Reset(void)`

[Input] None

[Output] None

[Function value] Error code

[Function] Suspends access to a branch stream and resets the branching playback (closes the stream group currently being used by the BPL and initializes all information).

Title	Data	Data Name	No.
Function specifications	Set branch stream information	`BPL_SetStmInfo`	1.3

[Format] `Sint32 BPL_SetStmInfo(Sint32 bstmid, Sint32 fid, Sint32 nkey, StmKey *keytbl)`

[Input]

`bstmid:`	Branch stream ID (`0≤bstmid<bstmmax`)
`fid`:	File ID
`nkey`:	Number of stream keys (`nkey≤BPL_STMKEY_MAX`)
`keytbl`:	Stream key table

[Output] None

[Function value] Error code

[Function] Assigns branch stream information (information on the individual streams that are actually read) to a branch stream.

[Remarks]

(a) By assigning multiple stream keys to a file, the function can read channel interleaved-data.

(b) The maximum number of stream keys that can be assigned to a branch stream is `BPL_STMKEY_MAX`.
The `BPL_Init` function specifies the total number of stream key types that can be used in all streams.

Title	Data	Data Name	No.
Function specifications	Get branch stream information	`BPL_GetStmInfo`	1.4

[Format] `Sint32 BPL_GetStmInfo(Sint32 bstmid, Sint32 *fid, Sint32 *nkey, StmKey *keytbl)`

[Input] `bstmid`: Branch stream ID

[Output]

`fid:`	File ID
`nkey`:	Number of stream keys (`nkey ≤BPL_STMKEY_MAX`)
`keytbl`:	Stream key table

[Function value] Number of destinations that are already set (an error code results if this number is negative)

[Function] Gets the branch stream information that is assigned to a branch stream.
Refer to the destination information-setting function (`BPL_SetBranchInfo`) for the number of destinations.

Title	Data	Data Name	No.
Function specifications	Set destination information	`BPL_SetBranchInfo`	1.5

[Format] `Sint32 BPL_SetBranchInfo(Sint32 bstmid, Sint32 nbranch, Sint32 *brtbl)`

[Input]

`bstmid`:	Branch stream ID
`nbranch`:	Number of destinations
`brtbl`:	Branch table

[Output] None

[Function value] Error code

[Function] Assigns destination information (candidate destinations) to a branch stream.

[Remarks]

(a) Assigns the branch stream IDs of branch candidates to the branch table.
To indicate that there are no destinations, specify `BPL_BR_NONE` as a branch table element.

```
brtbl[0] = BSTMID_A;
brtbl[1] = BPL_BR_NONE;  /* No destinations (the end of BP) */
brtbl[2] = BSTMID_B;
nbranch = 3;
```

A destination is specified using the `BPL_SelectBranch` function and a branch number (a position in the branch table).
In this example, the branch processing produces the following results, depending on the destination that is selected:

Selected Destination	Branch Processing (when the branch execution switch of the server function is on)
Branch number 0	Branches to branch stream ID `BSTMID_A`.
Branch number 1	Terminates the branching playback process goes into the no destination state immediately after this branch number is selected.
Branch number 2	Branches to branch stream ID `BSTMID_B`.
Other	(`BPL_SelectBranch` returns the `BPL_ERR_BRNO` error and invalidates the selection.)

(b) The number of streams must satisfy the following conditions:
X +Y≤Z
X: The number of stream keys that are set in `bstmid`
Y: Total number of destination stream keys
Z: Maximum number of streams that can be opened simultaneously (specified using the `STM_Init` function)

Title	Data	Data Name	No.
Function specifications	Get destination information	`BPL_GetBranchInfo`	1.6

[Format] `Sint32 BPL_GetBranchInfo(Sint32 bstmid, Sint32 *nbranch, Sint32 *brtbl, Sint32 nelem)`

[Input]

`bstmid`:	Branch stream ID
`nelem`:	Number of branch table elements

[Output]

`nbranch`:	Number of destinations (0 if no branch candidates)
`brtbl`:	Branch table (a maximum of `nelem` branch candidates are stored from the beginning of the table)

[Function value] Error code

[Function] Gets the destination information that is assigned to a branch stream.

7.2 Branching Playback Processing

Title	Data	Data Name	No.
Function specifications	Set playback start stream	`BPL_SetStart`	2.1

[Format] `Sint32 BPL_SetStart(Sint32 bstmid)`
[Input] `bstmid` : Branch stream ID (`BPL_BR_NONE`: stops branching playback)
[Output] None
[Function value] Error code
[Function] Specify a playback start stream (the branch stream at the beginning of a scenario). To stop branching playback, specify `BPL_BR_NONE` as the branch stream ID.

Title	Data	Data Name	No.
Function specifications	Execute Branching Playback Server	`BPL_ExecServer`	2.2

[Format] `Sint32 BPL_ExecServer(Bool chgsw)`
[Input] `chgsw` : Branch execution switch (ON: branch, OFF: do not branch)
[Output] None
[Function value] Branching playback status
[Function] Executes the Branching Playback Server. When the branch execution switch is ON, performs branching (switches branch streams).

(1) Branching playback state

Constant	Description
`BPL_SVR_COMPLETED`	Branching playback completed.
`BPL_SVR_WAITSEL`	Wait for the selection of a destination.
`BPL_SVR_SELECT`	Destination selected.
`BPL_SVR_NOBRN`	No destinations.

For branching playback states, see Section 3.3, *Changing Branching Playback States*.

Title	Data	Data Name	No.
Function specifications	Select destination	`BPL_SelectBranch`	2.3

[Format] `Sint32 BPL_SelectBranch(Sint32 brno)`
[Input] `brno` : Branch number
[Output] None
[Function value] Error code
[Function] Selects a destination according to a specified branch number.
[Remarks]
(a) Specifying the switch “ON” during the execution of the `BPL_ExecServer` function results in branching (the current stream is switched with the selected destination).
(b) A destination must be selected even when there is only one branch candidate.

Title	Data	Data Name	No.
Function specifications	Get current stream	`BPL_GetCurStm`	2.4

[Format] `Sint32 BPL_GetCurStm(Sint32 nelem, StmHn *stmtbl)`

[Input] `nelem`: Number of elements in the stream handle table (`nelem ≤BPL_STMKEY_MAX)`

[Output] `stmtbl`: Stream handle table

[Function value] Branch stream ID (Negative ID=no corresponding branch streams)

[Function] Gets the current stream (branch stream ID and the stream handle) that is subject to read access.

[Remarks] (a) Stream handles that correspond to stream keys are set in the stream handle table.

Title	Data	Data Name	No.
Function specifications	Get destination stream	`BPL_GetNextStm`	2.5

[Format] `Sint32 BPL_GetNextStm(Sint32 nelem, StmHn *stmtbl)`

[Input] `nelem`: Number of elements in the stream handle table (`nelem ≤BPL_STMKEY_MAX`)

[Output] `stmtbl`: Stream handle table

[Function value] Branch stream ID (Negative ID=no corresponding branch streams)

[Function] Gets a destination stream (branch stream ID and the stream handle).

[Remarks]
(a) Stream handles that correspond to stream keys are set in the stream handle table.
(b) The function value remains negative until a destination is selected (until the `BPL_SelectBranch` function is executed).

Title	Data	Data Name	No.
Function specifications	Get stream group	`BPL_GetStmGrp`	2.6

[Format] `StmGrpHn BPL_GetStmGrp(void)`

[Input] None

[Output] None

[Function value] Stream group handle

[Function] Gets the handle of the stream group that is used by the BPL.

[Remarks] (a) When activated, the BPL opens one stream group.
When the branching playback process is terminated, the stream group handle becomes `NULL`.