ACULAB.COM

Aculab digital telephony software

DPNSS call control API guide

MAN1140 Revision 6.8.7

PROPRIETARY INFORMATION

Document Revision

Rev	Date	By	Detail
5.0	Sep 1998		For version 5.0.1 DPNSS enhanced call drivers
5.0.1	Oct 2001	DJL	API change and review
5.0.2	May 2002	DJL	Minor changes to section 3
5.9.1	July 2002	DJL	Addition of call back when free appendix
5.10	Mar 2003	DJL	Addition of new call-back functions
6.0.0	09.05.05	DJL	Updates for V6 software
6.0.1	05.10.05	DJL	Additional clarification of feature messages added
6.4.0	02.12.05	DJL	Updates for 6.4.0 release
6.4.1	18.01.06	DJL	Small changes/typos following header file review
6.4.2	29.11.06	EC	Addition of signalling message reference information
6.4.3d1	28.08.08	MF	Addition of signalling message reference information
6.4.4	04.10.10	DF	Removed references to withdrawn products
6.4.5	20.10.10	EBJ	Updated to corporate fonts
6.4.6	11.11.10	DF	Updated following QA assessment
6.4.7	16.11.10	EBJ	Removal of Hyperlinks.
6.8.0	06.06.22	DSL	Update page footers.
6.8.6	25.10.22	DSL	Minor corrections.
6.8.7	13.02.23	DSL	Update title page

CONTENTS

1 Introduction

This functional specification describes the interface of a device driver capable of providing the requirements of a ‘Layer 3’ interface to the signalling code (DPNSS) when resident on an Aculab card.

The DPNSS driver has been written to support basic call control and the following extra features:

- Call Diversion Immediate and Busy
- Call Diversion on No Reply
- Diversion validation
- Virtual calls
- Call Hold
- Enquiry Call
- Call Transfer
- Transit working
- Layer 2 channel control
- Non Specified Information (NSI)
- Text
- Trunk Identity
- Conference
- Executive Intrusion
- Sending raw DPNSS SIS messages
- State of destination Enquiry
- Number Presentation Restriction
- Charge Account Codes
- Call back when free
- Call back when next used
- Call back messaging
- Loop Avoidance
- Extension status calls

CAUTION

This document is intended for use in conjunction with the DPNSS Specification BTNR 188 ISSUE 5 (BT Document - Digital Private Network Signalling System N01 (DPNSS1)). Before using these features, the user should be familiar with the Aculab Generic Call Control document (Aculab Call Control Driver Interface Guide) and BTNR 188. (DPNSS1)).

If compliance with BTNR 188 is to be achieved, it is also recommended that the compliance tables within BTNR 188 be adhered to, and that the document BTNR 188T is used as a test specification.

This specification does not presume any particular environment. It is intended for use under various operating systems. The functions are defined as library calls where isolation from the operating system is desired.

1.1 Scope

This functional specification is intended to be of use in the development of applications that make use of the various DPNSS function calls.

This specification describes the initiation and control of an outgoing call, the reception and control of an incoming call, and support of various DPNSS features. The control of timeslots and streams is documented in the switch API guide.

2 Interface Definition (APIs)

The following section describes the interface of the library functions and the device driver. Each function is described in terms of its calling parameters and the values that the function will return. No particular operating system is assumed.

Enhancements to the Aculab API often require extension of the structures used as parameters to Aculab API calls. To eliminate problems associated with this, the following steps **must** be performed:

```
memset(&structure, 0, sizeof(structure));
structure.size = sizeof(structure);
```

In C and C++ programs, these steps can be replaced with the following macro, defined in `acu_type.h`:

```
INIT_ACU_STRUCT(&structure);
```

2.1 feature_xparms - DPNSS feature support function library

DPNSS feature support uses a set of library function calls provided in addition to the generic call control library.

The DPNSS function library enables the application to send and receive instructions/information required to support the features specified at the start of this document.

The additional library function calls are shown below:

dpns_openout	open for outgoing call
dpns_send_overlap	sending overlap digits/information
dpns_incoming_ringing	incoming ringing
dpns_call_accept	accept incoming call
dpns_call_details	get call details
dpns_send_feat_info	send feature information
dpns_disconnect	disconnect call
dpns_release	release call
dpns_getcause	get idle cause
dpns_set_transit	set transit
dpns_send_transit	send transit
dpns_transit_details	transit details
dpns_set_l2_ch	set layer 2 channel
dpns_set_l2_state	set layer 2 state

The DPNSS `feature_xparms` structure is common to most of the above functions. It is used in addition to the parameters used for Basic Call Control with the generic call control library.

```
typedef struct feature_xparms
{
  ACU_INT        msg[MAX_FEAT_MSG]; /* Feature information message   */
  ACU_UCHAR      call_type;         /* Call type - real or virtual   */
  ACU_CHAR       digits[MAXNUM];    /* Feature digits                */
  ACU_CHAR       cli[MAXNUM];       /* Called Line Identity          */
  ACU_CHAR       nsi[MAXNSI];       /* Non Specified Information     */
```

```
  ACU_CHAR        txt[MAXTXT];        /* Text                          */
  ACU_CHAR        tid[MAXTID];        /* Trunk ID                      */
  ACU_UCHAR       clc;                /* Call/Called Line category     */
  ACU_UCHAR       held_clc;           /* Held Calling Line Category    */
  ACU_UCHAR       ipl;                /* Intrusion protection level    */
  ACU_UCHAR       icl;                /* Intrusion capability level    */
  ACU_UCHAR       routes              /* Remaining routes              */
  ACU_UCHAR       transits            /* Remaining transits            */
} FEATURE_XPARMS;
```

msg

This parameter is used to send and receive DPNSS feature messages and may be set to one of the following values, (the corresponding DPNSS identifiers are given in brackets):

2.1.1 DPNSS feature messages

DIB – diversion immediate and busy

Aculab Message	Mnemonic	DPNSS identifier and description
DIVERT_IMMEDIATE	DVT_I	Call divert immediate – used to indicate that the call has been generated following call diversion immediate. The array digits will contain the number from which the calling party has been diverted.
DIVERT_BUSY	DVT_B	Call divert on busy– used to indicate that the call has been generated following busy call diversion. The array digits will contain the number from which the calling party has been diverted
DIVERTING_IMM	DVG_I	Call diverting immediate - Used to indicate that the outgoing call has been generated following call diversion immediate. The array digits must hold the number from which the calling party has been diverted.
DIVERTING_BSY	DVG_B	Call diverting on busy - Used to indicate that the outgoing call has been generated following busy call diversion. The array digits must hold the number from which the calling party has been diverted.
DIVERTED_IMM	DVD_I	Call diverted immediate - An outgoing call has been diverted immediately to another party on the same destination PBX. The array digits will contain the number of the party the call has been diverted to.
DIVERTED_BSY	DVD_B	Call busy diverted – An outgoing call has been diverted on busy to another party on the same destination PBX. The array digits will contain the number of the party the call has been diverted to.

DR – diversion on no reply

Aculab Message	Mnemonic	DPNSS identifier and description
DIVERT_NO_REPLY	DVT_R	Call Divert on no reply– This is received on an outgoing call following EV_OUTGOING_RINGING and before connection. The application may choose to simply ignore this message or divert the outgoing call to the number given in the array digits.
DIVERTING_RNR	DVG_R	Call diverting on no reply - used to indicate that the call has been generated following call diversion on no reply. The array digits will contain the number from which the calling party has been diverted.
DIVERTED_RNR	DVD_R	Call diverted on no reply– An outgoing call has been diverted on no reply to another party on the same destination PBX. The array digits will contain the number of the party the call has been diverted to.

DV – diversion validation

Aculab Message	Mnemonic	DPNSS identifier and description
DIV_VALIDATION	DIV_V	Call Diversion Validation - used for a diversion validation request. The application should respond to the request by releasing the call via the function dpns_disconnect or dpns_release with feature_info.msg set to either ACKNOWLEDGE or REJECT.

HD – call hold

Aculab Message	Mnemonic	DPNSS identifier and description
HOLD_CALL	HOLD_REQ	Call hold request - The application is requested to place its party on hold (i.e. disconnect the speech channel). The application must respond via the function dpns_send_feat_info with the feature_info msg element set to either ACKNOWLEDGE or REJECT.
HOLD_ACK	ACK	Hold Acknowledge
HOLD_REJECT	REJ	Hold Reject
HOLD_NOT_SUPPORTED	SNU	Hold not supported
RECONNECT_CALL	RECON	Reconnect held call

EN – enquiry call

Aculab Message	Mnemonic	DPNSS identifier and description
ENQUIRY	ENQ	Enquiry Call – used to indicate an enquiry call. The element held_clc will be set to the calling line category of the party placed on hold before enquiry call setup.

TR – call transfer

Aculab Message	Mnemonic	DPNSS identifier and description
TRANSFER_O	TRFR	Transfer call originating
TRANSFER_T	TRFR	Transfer call terminating
TRANSFERRED	TRFD	Call transferred - the remote party has transferred a connected call.
TRANSFERRED_INFO		Call transfer information – Following the TRANSFERRED message the Calling Line Identity and Calling Line Category of the transferred party will be set in the CLI and CLC elements.

EI – executive intrusion

Aculab Message	Mnemonic	DPNSS identifier and description
INTRUSION_REQUEST	EI_R	Intrusion request - Indicates executive intrusion request generated by remote PBX. The icl element is set to the Intrusion Capability Level of the requesting party.
PV_INTRUSION	EI_PVR	Intrusion prior validation – Indicates intrusion prior validation request generated by the remote PBX. The icl element is set to Intrusion Capability Level of the requesting party.
INTRUSION_ACK	ACK	Intrusion acknowledge – Used to indicate acknowledge of intrusion request generated by the application.

Aculab Message	Mnemonic	DPNSS identifier and description
INTRUDING	EI_I	Intruding - Used to indicate successful intrusion on remote party.
IPL_REQUEST	IPL_R	Intrusion protection level request - Remote PBX intrusion protection level request.
IPL_RESPONSE	IPL	Intrusion protection level response – Remote PBX intrusion protection level response. The ipl element is set to the Intrusion Protection Level of the responding party.
INTRUSION_WITHDRAW	EI_W	Intrusion withdraw - Remote PBX intrusion withdraw.
WITHDRAW_ACK	ACK	Intrusion withdraw acknowledge () - Remote PBX intrusion withdraw acknowledge.
WITHDRAW_NOT_SUPPORTED	SNU	Withdraw not supported – Withdraw not supported by remote PBX.

AO – add on

Aculab Message	Mnemonic	DPNSS identifier and description
ADD_ON_VALIDATION	AD_V	Add on validation () - The application is requested to validate an add-on request. The application must respond with either ADD_ON_ACK or ADD_ON_REJ.
ADD_ON_ACK	ACK	Add on acknowledge ()
ADDED_ON	AD_O	Added on () - Remote PBX has formed a conference including the application party.
ADD_ON_REJECT	REJ	Add on reject ()
ADD_ON_NOT_SUPPORTED	SNU	Add on not supported () - Indicates that the remote PBX does not support the add-on feature. Received in response to ADD_ON_VALIDATION
ADD_ON_CLEARDOWN	AC_CDC	Add on clear down () - Sent by remote PBX to instruct application to clear down a conference (disconnecting all parties)
TWO_PARTY_O	TWP	Two party () - Sent by remote PBX to indicate two party call following add on (application designated as originating party). The cli and clc parameters are set to the connected party details.
TWO_PARTY_T	TWP	Two party () - Sent by remote PBX to indicate two party call following add on (application designated as terminating party). The cli and clc parameters are set to the connected party details.

CBF – call back when free

Aculab Message	Mnemonic	DPNSS identifier and description
CBWF_REQUEST	CBWF-R	Call Back When Free Request

Aculab Message	Mnemonic	DPNSS identifier and description
CBWF_CANCEL	CBWF-C	
CBWF_FREE_NOTIFY	CBWF-FN	Call Back When Free - Free Notify
RING_OUT	RO	Ring Out
CALL_BACK_COMPLETE	CBC	Call Back Complete
CBWF_CALL_SETUP_IMMEDIATE	CBWF-CSUI	Call Back When Free Call Setup Immediate
CBWF_CALL_SETUP_DELAYED	CBWF-CSUD	Call Back When Free Call Setup Delayed

CBM – call back messaging

Aculab Message	Mnemonic	DPNSS identifier and description
CALL_BACK_MESSAGE_REQ	CBM-R	Call Back Messaging Request
CALL_BACK_MESSAGE_CAN	CBM-C	Call Back Messaging Cancel

ES – extension status calls

Aculab Message	Mnemonic	DPNSS identifier and description
EXTENSION_STATUS_CALL	EST	Extension Status Call - indicates that the call being established is an Extension-Status Call.

CC – call charging

Aculab Message	Mnemonic	DPNSS identifier and description
CHARGE_REQUEST	CH-CR	Charge reporting – cost request
CHARGE_UNITS_USED	CH-UU	Charge reporting – units used
CHARGE_ACTIVATE	CH-ACT	Charge reporting - activate
CHARGE_ACCOUNT_REQUEST	CH-ACR	Charge Account Code Request
CHARGE_ACCOUNT_CODE	CH-AC	Charge Account Code

NPR – number presentation restriction

Aculab Message	Mnemonic	DPNSS identifier and description
NPR_A_PARTY_SUFFIX_B	NPR-A	Number presentation restriction - A party (restriction domain)

LA – loop avoidance

Aculab Message	Mnemonic	DPNSS identifier and description
LOOP_AVOIDANCE	LA	Used to indicate the number of further transits or routes are left before a call terminates. The transits parameter is compulsory and the routes parameter is optional. Both can be a value from 0 –25. A value of 0xFF (255) in the routes parameter will signal that this parameter is not to be used.

CBF – call back when next used request

Aculab Message	Mnemonic	DPNSS identifier and description
CBWNU_REQUEST	CBWNU-R	Call Back When Next Used Request

Other feature messages

Aculab Message	Mnemonic	DPNSS identifier and description
NO_MSG		Default to no instruction
ACKNOWLEDGE	ACK	Acknowledge - Used to acknowledge a feature requested by the application.
REJECT	REJ	Reject (REJ) - Used to reject a feature requested by the application.
STATE_OF_DEST_FREE	SOD_F	State of destination free (SOD_F) - Remote party is free or in the ringing state.
STATE_OF_DEST_BUSY	SOD_B	State of destination free (SOD_B) – used to indicate that the remote party is busy.
STATE_OF_DEST_REQ	SOD_R	State of destination free (SOD_R) – used to find out if diversion bypass is allowed.
DIV_BYPASS	DIV_BY	Diversion Bypass (DIV_BY)
RECONNECT		Reconnect held party.
NIGHT_SERVICE_DIVERT	NS-DVT	Night Service Divert (NS-DVT)
FEAT_NOT_SUPPORTED		Feature not supported - Feature requested by the application is not supported by the destination PBX.
DPNSS_RAW		Send raw DPNSS message from txt field. - Used to indicate to the driver, and firmware, that the txt field contains raw DPNSS information and should be passed unparsed.

The msg element is an array, which can hold up to MAX_FEAT_MSG feature instructions. The default setting for msg is NO_MSG. When sending a message, the application should begin with the first msg element. All remaining elements should be set to NO_MSG.

Later sections of this document refer to sending and receiving feature messages. Feature messages are sent by the application to the driver and vice versa using feature_xparms.msg.

call_type

This element is used to indicate call type (real or virtual). call_type is valid for both incoming and outgoing calls and should always be set to REAL or VIRTUAL.

REAL – Only used if the ts element has been set to -1. The device driver will use the first available **real** channel.

VIRTUAL - Is only used if the ts element has been set to -1. The device driver will use the first available **virtual** channel.

digits

Is an array of IA5 digits used in conjunction with feature instructions during call control.

cli

Is an array of IA5 digits used for Calling/Called Line Identity. It is valid for both incoming and outgoing calls.

nsi

Is an array of IA5 characters used to send and receive Non-Specified-Information.

txt

Is an array of IA5 characters used to send and receive text. This parameter can be used to send 'raw' DPNSS strings in Initial Service Request Messages.

tid
Is an array of IA5 characters used to send and receive the trunk identity string.

clc
Is used to send and receive Calling/Called Line. `category.clc` may be set to one of the following values:

`NO_CLC`	Default value
`ORDINARY`	`DPNSS CLC_ORD`
`DECADIC`	`DPNSS CLC_DEC`
`DASS2`	`DPNSS CLC_DASS2`
`PSTN`	`DPNSS CLC_PSTN`
`MF5`	`DPNSS CLC_MF5`
`OPERATOR`	`DPNSS CLC_OP`
`NETWORK`	`DPNSS CLC_NET`
`CONFERENCE`	`DPNSS CLC_CONF`

If no CLC is specified, the driver will default to `ORDINARY (DPNS CLC_ORD)`.

held_clc
Used in conjunction with Enquiry Call to send and receive the Calling Line Category of a held party. `held_clc` may be set to one of the following values:

`NO_CLC`	Default value
`ORDINARY`	`DPNSS CLC_ORD`
`DECADIC`	`DPNSS CLC_DEC`
`DASS2`	`DPNSS CLC_DASS2`
`PSTN`	`DPNSS CLC_PSTN`
`MF5`	`DPNSS CLC_MF5`
`OPERATOR`	`DPNSS CLC_OP`
`NETWORK`	`DPNSS CLC_NET`
`CONFERENCE`	`DPNSS CLC_CONF`

ipl
Is used to send and receive Intrusion Protection Levels. Refer to BTNR 188 Section 10 Paragraph 2.2.1 for the valid range of values.

icl
Is used to send and receive Intrusion Capability Levels. Refer to BTNR 188 Section 10 Paragraph 2.2.1 for the valid range of values.

Parameters not used in the `feature_xparms` structure must be initialised to their default values.

routes
Is used signal the number of further routes that call is allowed to attempt. Refer to BTNR 188 Section 38 for more details. Can have a value from 0 – 25. A value of 0xff (255) means that the parameter will be omitted or has been omitted on receipt. This parameter is optional for loop avoidance.

Parameters not used in the `feature_xparms` structure must be initialised to their default values.

Transits
Is used signal the number of further transits that call is allowed to attempt. Refer to

BTNR 188 Section 38 for more details. Can have a value from 0 – 25. This parameter is mandatory for loop avoidance.

Parameters not used in the `feature_xparms` structure must be initialised to their default values.

2.2 dpns_openout() - DPNSS open for outgoing call

This function allows an application to initiate an outgoing call. The function registers the outgoing call requirement with the device driver, which if satisfied with the calling parameters, will return a unique call identifier, the `handle`. The call `handle` must be used in all successive call control related operations on the driver.

Synopsis

```
ACU_ERR dpns_openout (DPNS_OUT_XPARMS * outdetailsp);


typedef struct dpns_out_xparms
{
      ACU_ULONG                 size;
      ACU_CALL_HANDLE           handle;
      ACU_PORT_ID               net;
      ACU_INT                   ts;
      ACU_INT                   cnf;
      ACU_INT                   sending_complete;
      ACU_CHAR                  destination_addr[MAXNUM];
      ACU_CHAR                  originating_addr[MAXNUM];
      ACU_ACT                   app_context_token;
      ACU_EVENT_QUEUE           queue_id;
      union                     uniquex unique_xparms;
      FEATURE_XPARMS            feature_info;
   } DPNS_OUT_XPARMS;

  typedef struct feature_xparms
    {
      ACU_INT                   msg[MAX_FEAT_MSG];
      ACU_UCHAR                 call_type;
      ACU_CHAR                  digits[MAXNUM];
      ACU_CHAR                  cli[MAXNUM];
      ACU_CHAR                  nsi[MAXNSI];
      ACU_CHAR                  txt[MAXTXT];
      ACU_CHAR                  tid[MAXTID];
      ACU_UCHAR                 clc;
      ACU_UCHAR                 held_clc;
      ACU_UCHAR                 ipl;
      ACU_UCHAR                 icl;
    } FEATURE_XPARMS;
```

Call input parameters

The dpns_openout function takes a pointer, outdetailsp, to a structure dpns_out_xparms. dpns_out_xparms has the same format as the out_xparms structure described in the Generic Call Control specifications, but with the addition of the feature_info structure.

dpns_out_xparms() must be initialised with the values as defined for call_openout() in the gGeneric call control API specification.

Feature xparm input parameters

In addition to the Generic Call Control input parameters, the feature_info parameters may be set to the following values when used with dpns_openout():

Msg

The valid msg parameter values for this call include:

NPR_A_PARTY_SUFFIX_B

NIGHT_SERVICE_DIVERTING

DIVERTING_IMM

DIVERTING_BSY

DIVERTING_RNR

DIV_BYPASS

ENQUIRY

DIV_VALIDATION

INTRUSION_REQUEST

PV_INTRUSION

CALL_BACK_MESSAGE_REQ

EXTENSION_STATUS_CALL

CBWF_REQUEST

CBWF_CANCEL

CBWF_FREE_NOTIFY

CBWF_CALL_SETUP_IMMEDIATE

CBWF_CALL_SETUP_DELAYED

CALL_BACK_MESSAGE_CAN

DPNSS_RAW

For descriptions of the feature_xparms parameters and values, please refer to section 2.1

Return Values

handle

If successful, this will contain a unique (non zero) call identifier, which must be used in all successive call related operations on the driver.

On successful completion, a value of zero is returned otherwise a negative value will be returned indicating the type of error.

Signalling messages transmitted

This will transmit at least an ISRMI or ISRMC. If there is a lot of information to transmit then this may be followed by one or more SSRMI and optionally an SSRMC. For further information, refer to BTNR 188.

2.3 dpns_send_overlap() - DPNSS sending overlap digits/information

This function may be used to send the destination address of an outgoing call more digits or feature information. The function may also be used any time that a valid outgoing call handle is available and the state of that handle is `EV_WAIT_FOR_OUTGOING`. The outgoing call handle would have been obtained with either the `call_openout` or `dpns_openout` functions.

Synopsis

```
ACU_ERR dpns_send_overlap(DPNS_OVERLAP_XPARMS *overlapp);

typedef struct dpns_overlap_xparms

{
      ACU_ULONG               size;
      ACU_CALL_HANDLE         handle;
      ACU_INT                 sending_complete;
      ACU_CHAR                destination_addr[MAXNUM];
      FEATURE_XPARMS          feature_info;
    } DPNS_OVERLAP_XPARMS;

typedef struct feature_xparms

{
      ACU_INT                 msg[MAX_FEAT_MSG];
      ACU_UCHAR               call_type;
      ACU_CHAR                digits[MAXNUM];
      ACU_CHAR                cli[MAXNUM];
      ACU_CHAR                nsi[MAXNSI];
      ACU_CHAR                txt[MAXTXT];
      ACU_CHAR                tid[MAXTID];
      ACU_UCHAR               clc;
      ACU_UCHAR               held_clc;
        ACU_UCHAR             ipl;
      ACU_UCHAR               icl;
    } FEATURE_XPARMS;
```

Call input parameters

The `dpns_send_overlap()` function takes a pointer; overlap, to a structure of type `dpns_overlap_xparms`. The `dpns_overlap_xparms` structure has the same format as the `overlap_xparms` structure, described in the Generic Call Control specifications, with the addition of the DPNSS `feature_info` structure. The `dpns_overlap_xparms()` must be initialised with the values specified for `call_send_overlap()` in the Generic Call Control specifications.

Feature xparm input parameters

In addition to the Generic Call Control input parameters, the `feature_info` parameters may be set to the following values when used with `dpns_send_overlap()`:

msg

The valid `msg` parameters for this call include:

```
DPNSS_RAW
NIGHT_SERVICE_DIVERTING
DIVERTING_IMM
DIVERTING_BSY
DIVERTING_RNR
ENQUIRY
DIV_VALIDATION
INTRUSION_REQUEST
PV_INTRUSION
```

For descriptions of the `feature_xparms` parameters and values, please refer to Section 2.1

Return values

On successful completion, a value of zero is returned; otherwise, a negative value will be returned indicating the type of error.

Signalling messages transmitted

This will transmit an SSRMI and/or SSRMC. For further information, refer to BTNR 188.

2.4 dpns_call_details() - DPNSS get call details

This function is used to read the details of an incoming/outgoing DPNSS call connected through the device driver.

Synopsis

```
ACU_ERR dpns_call_details (DPNS_DETAIL_XPARMS * detailsp);
typedef struct dpns_detail_xparms
    {
      ACU_ULONG               size;
      ACU_CALL_HANDLE         handle;
      ACU_LONG                timeout;
      ACU_INT                 valid;
      ACU_INT                 stream;
      ACU_INT                 ts;
      ACU_INT                 calltype;
      ACU_INT                 sending_complete;
      ACU_CHAR                destination_addr[MAXNUM];
      ACU_CHAR                originating_addr[MAXNUM];
      ACU_CHAR                connected_addr[MAXNUM];
      ACU_CHAR                redirected_addr[MAXNUM];
      union  uniquex          unique_xparms;
      FEATURE_XPARMS          feature_info;
    } DPNS_DETAIL_XPARMS;


typedef struct feature_xparms
```

```
{
  ACU_INT                 msg[MAX_FEAT_MSG];
  ACU_UCHAR               call_type;
  ACU_CHAR                digits[MAXNUM];
  ACU_CHAR                cli[MAXNUM];
  ACU_CHAR                nsi[MAXNSI];
  ACU_CHAR                txt[MAXTXT];
  ACU_CHAR                tid[MAXTID];
  ACU_UCHAR               clc;
  ACU_UCHAR               held_clc;
  ACU_UCHAR               ipl;
  ACU_UCHAR               icl;
} FEATURE_XPARMS;
```

Call input parameters

The `dpns_call_details()` function takes a pointer, `detailsp`, to a structure `dpns_detail_xparms`. The `dpns_detail_xparms` structure has the same format as the `detail_xparms` structure, described in the Generic Call Control specifications, with the addition of the DPNSS feature_info structure. In most instances, the `dpns_detail_xparms` must be initialised with the values specified for `call_details()` in the Generic Call Control specifications. The exceptions are ***feature_information*** and ***app_context_token*** , which are not used with DPNSS

Return Values

In addition to the information elements described for `call_details()` in the Generic Call Control Specification, the structure `feature_info` may contain the following information:

Feature parameters

msg

All `msg` parameter are valid for this call. For descriptions of the `feature_xparms` parameters and values, please refer to section 2.1

Call parameters

redirect_addr –a null terminated string of IA5 digits containing the redirected number.

On successful completion, a value of zero is returned otherwise a negative value will be returned indicating the type of error.

2.5 dpns_incoming_ringing() - DPNSS incoming ringing

This function may be used to optionally send the ringing message to the network.

The function `dpns_incoming_ringing` may be used after an incoming call has been detected but before the call has been accepted. Use of the function will stop any further destination address digits being received.

Synopsis

```
ACU_ERR dpns_incoming_ringing(DPNS_INCOMING_RING_XPARMS *inringp);

typedef struct dpns_incoming_ring_xparms
    {
      ACU_ULONG                 size;
      ACU_CALL_HANDLE           handle;
      FEATURE_XPARMS            feature_info;
    } DPNS_INCOMING_RING_XPARMS;

    typedef struct feature_xparms
    {
      ACU_INT                   msg[MAX_FEAT_MSG];
      ACU_UCHAR                 call_type;
      ACU_CHAR                  digits[MAXNUM];
      ACU_CHAR                  cli[MAXNUM];
      ACU_CHAR                  nsi[MAXNSI];
      ACU_CHAR                  txt[MAXTXT];
      ACU_CHAR                  tid[MAXTID];
      ACU_UCHAR                 clc;
      ACU_UCHAR                 held_clc;
      ACU_UCHAR                 ipl;
      ACU_UCHAR                 icl;
    } FEATURE_XPARMS;
```

Call input parameters

The `dpns_incoming_ringing()` function takes a pointer, `inringp`, to a structure `dpns_incoming_ring_xparms`. The structure must be initialised with the following values before invoking the function.

The input parameter `handle` uniquely identifies the call that will send the incoming ringing message.

Feature xparm input parameters

The `feature_info` structure is used to generate feature call information/instructions.

msg

The valid `msg` parameter values for this call include:

```
DPNSS_RAW
NIGHT_SERVICE_DIVERT
NIGHT_SERVICE_DIVERTED
DIVERT_IMMEDIATE
DIVERT_BUSY
DIVERTED_IMM
DIVERTED_BSY
INTRUSION_ACK
```

For descriptions of the `feature_xparms` parameters and values, please refer to section 2.1

Return Values

On successful completion, a value of zero is returned otherwise a negative value will be returned indicating the type of error.

Signalling messages transmitted

This will transmit a NAM. For further information, refer to BTNR 188.

2.6 dpns_send_feat_info() - DPNSS send feature info

This function is used to send DPNSS feature information to the network following `call_incoming_ringing()` or `dpns_incoming_ringing()` on an incoming call and `EV_OUTGOING_RINGING` on an outgoing call.

Synopsis

```
ACU_ERR dpns_send_feat_info(DPNS_FEATURE_XPARMS *featurep);


typedef struct dpns_feature_xparms
{
      ACU_ULONG               size;
      ACU_CALL_HANDLE         handle;
      FEATURE_XPARMS          feature_info;
    } DPNS_FEATURE_XPARMS;


typedef struct feature_xparms
    {
      ACU_INT                 msg[MAX_FEAT_MSG];
      ACU_UCHAR               call_type;
      ACU_CHAR                digits[MAXNUM];
      ACU_CHAR                cli[MAXNUM];
      ACU_CHAR                nsi[MAXNSI];
      ACU_CHAR                txt[MAXTXT];
      ACU_CHAR                tid[MAXTID];
      ACU_UCHAR               clc;
      ACU_UCHAR               held_clc;
      ACU_UCHAR               ipl;
```

```
    ACU_UCHAR               icl;
  } FEATURE_XPARMS;
```

Call input parameters

The `dpns_send_feat_info()` function takes a pointer, `featurep`, to a structure `dpns_feature_xparms`. The structure must be initialised with the following values before invoking the function.

The input parameter `handle` uniquely identifies the call that will send the incoming ringing message.

Feature xparm input parameters

The `feature_info` structure is used to generate DPNSS feature information/instructions.

msg

The valid *msg* parameter values for this call include:

```
        DPNSS_RAW
        ADD_ON_CLEARDOWN
        ADD_ON_VALIDATION
        ADD_ON_ACK
        ADD_ON_REJECT
        ADDED_ON
        TWO_PARTY_O
        TWO_PARTY_T
        HOLD_CALL
        HOLD_ACK
        HOLD_REJECT
        TRANSFER_O
        TRANSFER_T
        RECONNECT_CALL
        CALL_BACK_COMPLETE
        DIVERT_NO_REPLY
        DIVERTED_RNR
        STATE_OF_DEST_FREE
        RING_OUT
        ACKNOWLEDGE
        REJECT
        CHARGE_UNITS_USED
CHARGE_ACCOUNT_REQUEST
CHARGE_ACCOUNT_CODE
CHARGE_ACTIVATE
IPL_REQUEST
IPL_RESPONSE
INTRUSION_WITHDRAW
INTRUSION_REQUEST
INTRUSION_ACK
```

```
WITHDRAW_ACK
```

Other feature parameters used by this call, include:

nsi

txt

tid

ipl

icl

digits

For descriptions of the other `feature_xparms` parameters and values, please refer to section 2.1

Return Values

On successful completion, a value of zero is returned otherwise a negative value will be returned indicating the type of error.

Signalling messages transmitted

This will transmit an EEMC or an LLMC (only with CHARGE_UNITS_USED). For further information, refer to BTNR 188.

2.7 dpns_call_accept() - DPNSS accept incoming call

This function is used to send an incoming call connection message to the calling party.

Synopsis

```
ACU_ERR dpns_call_accept(DPNS_CALL_ACCEPT_XPARMS *call_acceptp);

typedef struct dpns_call_accept_xparms
    {
      ACU_ULONG                size;
      ACU_CALL_HANDLE          handle;
      FEATURE_XPARMS           feature_info;

    } DPNS_CALL_ACCEPT_XPARMS;

typedef struct feature_xparms
    {
      ACU_INT                  msg[MAX_FEAT_MSG];
      ACU_UCHAR                call_type;
      ACU_CHAR                 digits[MAXNUM];
      ACU_CHAR                 cli[MAXNUM];
      ACU_CHAR                 nsi[MAXNSI];
      ACU_CHAR                 txt[MAXTXT];
      ACU_CHAR                 tid[MAXTID];
      ACU_UCHAR                clc;
      ACU_UCHAR                held_clc;
      ACU_UCHAR                ipl;
      ACU_UCHAR                icl;
```

```
} FEATURE_XPARMS;
```

Call input parameters

The `dpns_call_accept()` function takes a pointer, `call_acceptp`, to a structure `dpns_call_accept_xparms`. The structure must be initialised with the following values before invoking the function.

handle

The input parameter `handle` uniquely identifies the connected call.

Feature xparm input parameters

The `feature_info` structure is used to generate DPNSS feature instructions/information. The following values can be used by `dpns_call_accept()`:

Msg

The valid *msg* parameter values for this call include:

```
DPNSS_RAW
CHARGE_ACTIVATE
CHARGE_ACCOUNT_REQUEST
INTRUDING
```

Other feature parameters used by this call, include:

nsi

txt

tid

For definitions of the `feature_xparms` parameters and values, please refer to section 2.1

Return Values

On successful completion a value of zero is returned otherwise a negative value will be returned indicating the type of error.

Signalling messages transmitted

If the application has not sent ringing, then this will first transmit a NAM. Otherwise, it will send a CCM. For further information, refer to BTNR 188.

2.8 dpns_getcause() - DPNSS get idle cause

This function can be used to read the clearing cause when an incoming or outgoing call goes to `EV_IDLE`. The returned clearing cause will only be valid at `EV_IDLE`.

Synopsis

```
ACU_ERR dpns_getcause(DPNS_CAUSE_XPARMS *causep);

typedef struct dpns_cause_xparms
    {
      ACU_ULONG                 size;
      ACU_CALL_HANDLE           handle;
      ACU_INT                   cause;
      ACU_INT                   raw;
      FEATURE_XPARMS            feature_info;
    } DPNS_CAUSE_XPARMS;


typedef struct feature_xparms
    {
      ACU_INT                   msg[MAX_FEAT_MSG];
      ACU_UCHAR                 call_type;
      ACU_CHAR                  digits[MAXNUM];
      ACU_CHAR                  cli[MAXNUM];
      ACU_CHAR                  nsi[MAXNSI];
      ACU_CHAR                  txt[MAXTXT];
      ACU_CHAR                  tid[MAXTID];
      ACU_UCHAR                 clc;
      ACU_UCHAR                 held_clc;
      ACU_UCHAR                 ipl;
      ACU_UCHAR                 icl;
    } FEATURE_XPARMS;
```

Input Parameters

The `dpns_getcause()` function takes a pointer, `causep`, to a structure `dpns_cause_xparms`. The structure of `dpns_cause_xparms` is the same as the `cause_xparms structure`, described in the Generic Call Control specifications, with the addition of the `feature_info` structure. `dpns_cause_xparms` must be initialised with the values described for `call_getcause()` in the Generic Call Control specifications.

Return Values

In addition to the return values described in the Generic Call Control specifications, the `feature_info` may contain the following:

Msg

The valid *msg* parameter values for this call can include:

```
ACKNOWLEDGE
REJECT
DIVERT_IMMEDIATE
DIVERT_BUSY
```

Other feature parameters used by this call, include:

nsi

txt

For definitions of the `feature_xparms` parameters and values, please refer to section 2.1

On successful completion, a value of zero is returned otherwise a negative value will be returned indicating the type of error.

2.9 dpns_disconnect() - DPNSS disconnect call

This function can be used to disconnect an incoming or outgoing call currently routed through the driver. If the `dpns_disconnect()` function is successful, the driver will start the disconnect procedure and will return immediately to the calling process.

When the call has been disconnected, the event `EV_IDLE` will be generated. The `dpns_release()` function must then be used to give back the `handle` to the driver.

Synopsis

```
ACU_ERR dpns_disconnect(DPNS_CAUSE_XPARMS *causep);

typedef struct dpns_cause_xparms
    {
      ACU_ULONG                 size;
      ACU_CALL_HANDLE           handle;
      ACU_INT                   cause;
      ACU_INT                   raw;
      FEATURE_XPARMS            feature_info;
    } DPNS_CAUSE_XPARMS;

typedef struct feature_xparms
    {
      ACU_INT                   msg[MAX_FEAT_MSG];
      ACU_UCHAR                 call_type;
      ACU_CHAR                  digits[MAXNUM];
      ACU_CHAR                  cli[MAXNUM];
      ACU_CHAR                  nsi[MAXNSI];
      ACU_CHAR                  txt[MAXTXT];
      ACU_CHAR                  tid[MAXTID];
      ACU_UCHAR                 clc;
      ACU_UCHAR                 held_clc;
      ACU_UCHAR                 ipl;
      ACU_UCHAR                 icl;
    } FEATURE_XPARMS;
```

Input Parameters

The dpns_disconnect function takes a pointer, causep, to a structure dpns_cause_xparms. dpns_cause_xparms is the same as the cause_xparms structure, described in the Generic Call Control specifications, with the addition of the feature_info structure. dpns_cause_xparms must be initialised with the values described for call_getcause() in the Basic Call Control specifications.

Feature xparm parameters

The following feature_info elements may be used by the application:

Msg

The valid *msg* parameter values for this call include:

```
DPNSS_RAW
ACKNOWLEDGE
REJECT
NIGHT_SERVICE_DIVERTED
DIVERTED_IMM
DIVERTED_BSY
STATE_OF_DEST_FREE
STATE_OF_DEST_BUSY
CHARGE_UNITS_USED
```

Other feature parameters used by this call, include *nsi and txt.* or definitions of the feature_xparms parameters and values, please refer to section 2.1

Return Values

On successful completion, a value of zero is returned otherwise a negative value will be returned indicating the type of error.

NOTE

If there is a call in progress when dpns_disconnect is invoked, the driver will initiate the disconnect procedure and will immediately return control to the calling process.

Signalling messages transmitted

This will transmit a CRM/CIM. For further information, refer to BTNR 188.

2.10 dpns_release() - DPNSS release call

This function must be used to relinquish ownership of a call handle in response to call termination EV_IDLE, or any error condition that may cause the application to abandon the call. If the dpns_release() function is successful, the driver will disconnect the call and the call handle will be closed. The handle may no longer be used by the application.

Synopsis

```
ACU_ERR dpns_release (DPNS_CAUSE_XPARMS * causep);

typedef struct dpns_cause_xparms
    {
      ACU_ULONG                 size;
      ACU_CALL_HANDLE           handle;
      ACU_INT                   cause;
```

```
    ACU_INT                 raw;
    FEATURE_XPARMS          feature_info;
  } DPNS_CAUSE_XPARMS;
typedef struct feature_xparms
  {
    ACU_INT                 msg[MAX_FEAT_MSG];
    ACU_UCHAR               call_type;
    ACU_CHAR                digits[MAXNUM];
    ACU_CHAR                cli[MAXNUM];
    ACU_CHAR                nsi[MAXNSI];
    ACU_CHAR                txt[MAXTXT];
    ACU_CHAR                tid[MAXTID];
    ACU_UCHAR               clc;
    ACU_UCHAR               held_clc;
    ACU_UCHAR               ipl;
    ACU_UCHAR               icl;
  } FEATURE_XPARMS;
```

Input Parameters

The function `dpns_release()` takes a pointer, `causep`, to a structure `dpns_cause_xparms`. `dpns_cause_xparms` is the same as the `cause_xparms` structure described in the Generic Call Control specifications, with the addition of the `feature_info` structure. `dpns_cause_xparms` must be initialised with the values described for `call_getcause()` in the Generic Call Control specifications.

Feature xparm parameters

The following `feature_info` elements may be used by the application:

nsi

txt

NOTE

If there is a call in progress when `dpns_release` is invoked, the calling process will block in the driver until the driver has disconnected the call. Control will then be returned to the application. The `feature_info` elements `nsi` and `txt` are only valid if the call is not in idle state

Return Values

On successful completion, a value of zero is returned otherwise a negative value will be returned indicating the type of error.

Signalling messages transmitted

If the call is still connected this will transmit a CRM. For further information, refer to BTNR 188.

2.11 dpns_set_transit() - DPNSS set transit

This function may be used to invoke DPNSS transit working for both incoming and outgoing calls. Refer to BTNR 188 for further details of Transit functionality.

Synopsis

```
ACU_ERR dpns_set_transit(ACU_CALL_HANDLE handle);
```

Input Parameters

handle

The input parameter `handle` uniquely identifies the call to be placed in transit state.

Return Values

On successful completion, a value of zero is returned, the event `EV_DPNS_TRANSIT` will be raised by the driver. If the call attempt is unsuccessful, a negative value will be returned indicating the type of error.

2.12 dpns_transit_details() - DPNSS transit details

This function is used to read a DPNSS transit message received from the network. `dpns_transit_details()` should only be called following `dpns_set_transit()`, call transfer, or two party working following conference.

Synopsis

```
ACU_ERR dpns_transit_details(DPNS_TRANSIT_XPARMS *transitp);

typedef struct dpns_transit_xparms
  {
    ACU_ULONG                 size;
    ACU_CALL_HANDLE           handle;
    ACU_LONG                  timeout;
    ACU_INT                   valid;
    ACU_CHAR                  trans_msg[TRANSIT_MSG_LENGTH];
  } DPNS_TRANSIT_XPARMS;
```

Input Parameters

The `dpns_transit_details()` takes a pointer, `transitp`, to a structure `dpns_transit_xparms`. The structure must be initialised with the following values before invoking the function.

handle

The input parameter `handle` is used to uniquely identify the call.

timeout

This parameter is ignored for this call.

Return Values

trans_msg

The ASCII string trans_msg contains the DPNSS message, which is to be forwarded to the destination party. It is important that this string is not modified before forwarding to the destination party.

valid

The return value `valid` is a Boolean, which indicates whether the details returned are valid, or not.

On successful completion, a value of zero is returned otherwise a negative value will be returned indicating the type of error.

2.13 dpns_send_transit() - DPNSS send transit

This function is used to send a DPNSS transit message received from the network via `dpns_transit_details()`.

Synopsis

```
ACU_ERR dpns_send_transit(DPNS_TRANSIT_XPARMS *transitp);

typedef struct dpns_transit_xparms
    {
      ACU_ULONG               size;
      ACU_CALL_HANDLE         handle;
      ACU_LONG                timeout;
      ACU_INT                 valid;
      ACU_CHAR                trans_msg[TRANSIT_MSG_LENGTH];
    } DPNS_TRANSIT_XPARMS;
```

Input Parameters

The function `dpns_send_transit()` takes a pointer, `transitp`, to a structure `dpns_transit_xparms`. The structure must be initialised with the following values before invoking the function.

handle

The input parameter `handle` is used to uniquely identify transit message destination call.

trans_msg

The input parameter `trans_msg` must contain the unaltered ASCII string received from `dpns_transit_details()`.

timeout & valid

These parameters are no longer used but are retained for backward compatibility.

Return Values

On successful completion, a value of zero is returned otherwise a negative value will be returned indicating the type of error.

Signalling messages transmitted

This will transmit an EEMC. For further information, refer to BTNR 188.

2.14 dpns_set_l2_ch() - DPNSS set layer 2 channel

This function is used to enable and disable a DPNSS channel at Layer 2 (the data link layer).

It is recommended that this function not be used during call processing.

Synopsis

```
ACU_ERR dpns_set_l2_ch(DPNS_L2_XPARMS *dpns_l2_parms);

typedef struct dpns_l2_xparms
    {
      ACU_ULONG                size;
      ACU_PORT_ID              net;
      ACU_INT                  channel;
      ACU_UCHAR                state;
      ACU_LONG                 timeout;
    } DPNS_L2_XPARMS;
```

Input Parameters

The function `dpns_set_l2_ch()` takes a pointer, `l2_parms`, to a structure `dpns_l2_xparms`. The structure must be initialised with the following values before calling the function. Note that the `timeout` parameter is not used in this function.

net

The input parameter `net` must contain the number of the network port on which the DPNSS layer 2 channel is to be set.

channel

The input parameter `channel` must contain the number of the DPNSS channel which is to be set.

state

The `state` parameter is used to either enable or disable a channel and must be set to one of the following values:

`DPNS_L2_ENABLE`	Enable DPNSS layer 2 channel.
`DPNS_L2_DISABLE`	Disable DPNSS layer 2 channel.

timeout

This parameter is no longer used but is retained for backward compatibility.

Return Values

On successful completion, a value of zero is returned otherwise a negative value will be returned indicating the type of error.

2.15 dpns_l2_state() - DPNSS Layer 2 State

This function is used to return the current state of a DPNSS Layer 2 channel.

Synopsis

```
ACU_ERR dpns_l2_state(DPNS_L2_XPARMS *dpns_l2_parms);

typedef struct dpns_l2_xparms
    {
      ACU_ULONG               size;
      ACU_PORT_ID             net;
      ACU_INT                 channel;
      ACU_UCHAR               state;
      ACU_LONG                timeout;
    } DPNS_L2_XPARMS;
```

Input Parameter

The `dpns_l2_state()` takes a pointer, `l2_parms`, to a structure `dpns_l2_xparms`. The structure must be initialised with the following values before calling the function.

net

The input parameter `net` must contain the number of the network port on which the DPNSS channel is to be examined.

channel

The input parameter `channel` must contain the number of the DPNSS channel that is to be examined. It will have a value depending upon the barred channels/time slots in the output timeslot vector `validvector` returned by `call_signal_info()`.

timeout

Is not valid for this call.

Return Values

On successful completion, a value of zero is returned otherwise a negative value will be returned indicating the type of error.

state

The return value `state`, within `l2_parms`, will contain the current state of the DPNSS layer 2 channel and will be set to one of the following values:

`DPNS_L2_ENABLED`	DPNSS channel is enabled.
`DPNS_L2_DISABLED`	DPNSS channel is disabled.

3 DPNSS feature call control

Important Notice

It is recommended that the user be familiar with BTNR 188 Issue 5 and the Aculab V6 Call Control API Guide before proceeding to use the DPNSS Call Control API.

The following sections describe the function calls required to support the specified DPNSS features. When describing the library function calls, only the parameters required to support a specific feature are given. For a full list of the possible input parameters for a library function call, refer to section 2 of this document.

Each of the features described in the following sections must be enabled via the command line parameters detailed in Appendix A.

3.1 set_feat_msg() - sending and receiving DPNSS feature messages

DPNSS feature messages are sent and received using the `FEATURE_XPARMS` structure described in section 2 of this document. The `FEATURE_XPARMS` structure is common to most of the library functions.

The *`msg`* element is an array (within `FEATURE_XPARMS`), which can hold up to `MAX_FEAT_MSG` feature instructions. The default setting for *`msg`* is `NO_MSG`.

To send a feature message the application must set the first empty element in the array `msg` to the desired value.

The following routine can be used to set the first available *`msg`* element within the `FEATURE_XPARMS` structure.

```
int set_feat_msg (FEATURE_XPARMS *feature_xparms, ACU_INT feat_msg)
{
    int index;

    for (index = 0; index < MAX_FEAT_MSG; index ++)
    {
        if (feature_xparms->msg[index] == NO_MSG)
        {
            feature_xparms->msg[index] = feat_msg;
            return 0;
        }
    }
    return -1;
}
```

Feature messages can be sent using the following functions:

`dpns_openout()`

`dpns_send_overlap()`

`dpns_incoming_ringing()`

`dpns_call_accept()`

`dpns_send_feat_info()`

`dpns_disconnect()`

`dpns_release()`

To send feature information following incoming or outgoing ringing the application should use the function `dpns_call_accept()` on call connection, and `dpns_send_feat_info()` at any other time before call clearing.

The device driver uses the same method for sending feature messages to the

application. It will always place messages starting at location 0 of the array *msg*. The application should read the array *msg* until an empty element (NO_MSG) or end of array is encountered.

Feature messages are received using the function dpns_call_details().

If the event mechanism for call control is used (described in the V6 Call Control API Guide) the application can call the dpns_call_details() function when the following events occur:

EV_INCOMING_CALL_DET,

EV_INCOMING_DETAILS,

EV_OUTGOING_RINGING,

EV_CALL_CONNECTED

An EV_INCOMING_DETAILS event may occur at any point between incoming call detection/ generation, and call clearing.

3.2 Call diversion immediate/busy (BTNR 188 section 11)

Call diversion immediate and busy diversion are available for incoming and outgoing calls.

3.2.1 Incoming call diversion to another PBX

To instruct an incoming call to divert on busy or divert immediate the application must use the function dpns_incoming_ringing() with the feature_info structure msg element set to DIVERT_IMMEDIATE or DIVERT_BUSY. The feature_info digits array must be set to the number of the party the call is to be diverted to.

Following dpns_incoming_ringing() the calling party will clear the call. The calling party may then attempt to establish the diverted call.

3.2.2 Outgoing call diversion to another PBX

After attempting to establish an outgoing call (call_openout(), dpns_openout()) the remote party may initiate call diversion immediate/busy. When remote diversion occurs the driver will clear the call and generate an EV_IDLE event.

The *msg* element of feature_info returned by dpns_getcause() will be set to either DIVERT_IMM or DIVERT_BSY. The digits element of feature_info will be set to the address of the party the call is to be diverted to.

3.2.3 Incoming call diversion on the same PBX

The application may divert an incoming call to another party without the use of another DPNSS link. The application can inform the calling party of 'on PBX' diversion via the function

dpns_incoming_ringing().

The *msg* element of feature_info (in dpns_incoming_ring_xparms) should be set to DIVERTED_BSY or DIVERTED_IMM with the digits array set to the number of the party the call has been diverted to.

3.2.4 Outgoing call diversion on the same PBX

During outgoing call setup, the destination PBX may divert a call to another party on the same PBX. If this occurs the driver will set the msg element of feature_info to DIVERTED_BSY or DIVERTED_IMM with the array digits set to the number of the party the call has been diverted to.

This information may be obtained via the library function dpns_call_details(), which

should be called following `EV_INCOMING_DETAILS`.

3.2.5 Incoming call diverting

If an incoming call has been diverted from another party, the *`msg`* element of `feature_info` will be set to `DIVERTING_IMM` or `DIVERTING_BSY`. The `digits` array will contain the number of the party the call has been diverted from.

The application can obtain this information via `dpns_call_details()` following `EV_INCOMING_DETAILS` or `EV_INCOMING_CALL_DET`.

3.2.6 Outgoing call diverting

If an outgoing call has been established following call diversion, the application can notify the destination party. When `dpns_openout()` is called, the `msg` element of `feature_info` must be set to `DIVERTING_IMM` or `DIVERTING_BSY`. The `digits` element of `feature_info` must be set to the number of the party the call has been diverted from.

3.3 Call diversion no reply (BTNR 188 section 11)

Ring No Reply (RNR) diversion is available for both incoming and outgoing calls.

3.3.1 Incoming call diversion to another PBX

To instruct an incoming call to RNR divert to another PBX the application must use the function `dpns_send_feat_info()` with the `feature_info` structure `msg` element set to `DIVERT_NO_REPLY`. The `feature_info` *`digits`* array must be set to the number of the party the call is to be diverted to.

If the calling PBX ignores the diversion request, or call diversion fails, no state change will occur. If the call diversion request is successful then the calling PBX will clear the call.

RNR diversion should only take place following `call_incoming_ringing()`, `dpns_incoming_ringing` and before call connection (`call_accept()`, `dpns_call_accept()`).

3.3.2 Outgoing call diversion to another PBX

After EV_OUTGOING_RINGING the called party may initiate RNR diversion. On receiving RNR diversion the driver will set the feature_info *msg* element to DIVERT_NO_REPLY and the *digits* element to the number of the party the call is to be diverted to. The application may check for RNR diversion information via dpns_call_details() following EV_INCOMING_DETAILS.

The application may choose to ignore the diversion information or attempt to establish a new call to the number supplied in feature_info digits. If the diversion is successful, the application should connect the calling party to the diversion call and clear the original call (call_disconnect(), dpns_disconnect()). Refer to BTNR 188 Section 11 for further details.

3.3.3 Incoming call diversion on the same PBX

The application may divert an incoming call to another party without the use of another DPNSS link. The application can inform the calling party of 'on PBX' RNR diversion via the function

dpns_send_feat_info(). The *msg* element of feature_info should be set to DIVERTED_RNR. The *digits* array must set to the number of the party the call has been diverted to.

3.3.4 Outgoing call diversion on the same PBX

EV_OUTGOING_RINGING the destination PBX may divert a call to another party on the same PBX. If this occurs the driver will set the *msg* element of feature_info to DIVERTED_RNR with the array digits set to the number of the party the call has been diverted to. This information may be obtained via the library function dpns_call_details(), which should be called following EV_INCOMING_DETAILS.

3.3.5 Incoming call diverting

If an incoming call has been diverted from another party, the *msg* element of feature_info will be set to DIVERTING_RNR. The digits array will contain the number of the party the call has been diverted from.

The application can obtain this information via dpns_call_details() following EV_INCOMING_DETAILS or EV_INCOMING_CALL_DET.

3.3.6 Outgoing call diverting

If an outgoing call has been established following call diversion, the application can notify the destination party. When dpns_openout() is called the msg element of feature_info must be

set to DIVERTING_RNR. The *digits* element of feature_info must be set to the number of the party the call has been diverted from.

3.4 Diversion validation (BTNR 188 section 11)

Diversion validation is available for both incoming and outgoing calls. Diversion validation should only be used with virtual calls.

3.4.1 Incoming diversion validation

When a diversion validation call is detected, the driver will set the `feature_info msg` element to `DIV_VALIDATION`. The application must respond to a diversion validation request by clearing the call. This is done using `dpns_disconnect()` with the *`msg`* element of `feature_info` set to either `ACKNOWLEDGE` or `REJECT`.

The application may obtain the `DIV_VALIDATION` message via `dpns_call_details()` following `EV_INCOMING_CALL_DET` or `EV_INCOMING_DETAILS`.

3.4.2 Outgoing diversion validation

The application can generate a diversion validation request by setting the *`msg`* element of `feature_info` to `DIV_VALIDATION`. The request should be initiated by using library function `dpns_openout()`.

The destination PBX will respond by clearing the call. The application can read the diversion validation response by checking the *`msg`* element of `feature_info` in `dpns_cause_xparms` after an `EV_IDLE`. The *`msg`* information element should be set to either `ACKNOWLEDGE` or `REJECT`. If the destination PBX did not understand the request, the *`msg`* element will not be set to `ACKNOWLEDGE` or `REJECT`.

3.5 Call hold (BTNR 188 section 12)

The application or remote party may initiate call hold.

3.5.1 Application initiated call hold

Call hold may only be initiated following call connection, i.e. after the `EV_CALL_CONNECTED` event.

To initiate call hold set the `feature_info` element *`msg`* to `HOLD_CALL`. The request can then be sent using the function `dpns_send_feat_info()`.

The application will receive the response by calling `dpns_call_details()` after an `EV_INCOMING_DETAILS` *`event`*. The *`msg`* element of `dpns_detail_xparms` will be either `HOLD_ACK` or `HOLD_REJ`. If the hold request has been accepted, an `EV_DPNS_HOLDING` event is raised. The call will remain in this state until either the application requests reconnection, or the holding or held party clears.

If the application wishes to reconnect the held party, the *`msg`* element of `feature_info` is set to `RECONNECT_CALL`, and the request initiated via `dpns_send_feat_info()`. The destination PBX will reconnect its held party. The `EV_CALL_CONNECTED` event will now be raised.

Should either the holding or held party clear during at `EV_DPNS_HOLDING` normal call clearing applies.

3.5.2 Remote initiated call hold

Call hold may only be initiated following call connection EV_CALL_CONNECTED.

When a remote call hold request is received, the driver will set the *msg* element of feature_info to HOLD_CALL. The application can obtain the hold request via dpns_call_details() following EV_INCOMING_DETAILS.

The application must respond to the request with the *msg* element of feature_info set to either HOLD_ACK or HOLD_REJ.

If the hold request is acknowledged, an EV_DPNS_HELD event will be generated. If the hold request is rejected no state change will occur.

A call will remain in EV_DPNS_HELD until either party clears or the remote party instructs the application to reconnect.

If the remote party requests reconnection the driver will set the msg element of feature_info to RECONNECT_CALL. The EV_CALL_CONNECTED will be generated. The application should reconnect its party to the traffic channel.

If either party clears, normal call clearing applies.

3.6 Enquiry call (BTNR 188 section 13)

Enquiry call is supported for both incoming and outgoing calls.

3.6.1 Outgoing enquiry call

Following call hold the application can make an enquiry call. To inform the remote party of an enquiry call the feature_info *msg* element is set to ENQUIRY, and the held_clc element set to the calling line category of the held party. An outgoing call can then be established using the dpns_openout function.

3.6.2 Incoming enquiry call

If an incoming call is an enquiry call, the *msg* element of feature_info will be set to ENQUIRY, and held_clc will be set to the calling line category of the held party. This information can be obtained via dpns_call_details() following EV_INCOMING_DETAILS or EV_INCOMING_CALL_DET.

3.7 Call transfer (BTNR 188 section 13)

Application controlled Call Transfer uses two DPNSS channels. It can be initiated after the application has placed a call on hold and established an enquiry call. The enquiry call may be in ringing or connected state.

The remote party in a call may transfer a call to the application.

3.7.1 Application initiated call transfer

The application may transfer (connect) an enquiry call and a held call (each using a separate channel). To initiate call transfer the application must set the *msg* element of feature_info to

TRANSFER_O or TRANSFER_T. TRANSFER_O is used to designate a party as the new originating party and TRANSFER_T is used to designate a party as the new terminating party. For further details on call transfer, refer to BTNR 188 Section 13.

The transfer request is sent via dpns_send_feat_info(), and is required for both the enquiry, and held calls.

Following call transfer, the enquiry Call and the held Call will be get the EV_DPNS_TRANSIT event. The application then operates as a transit PBX for the remainder of the call. Refer to section 3.8 for Transit working details.

3.7.2 Remote party initiated call transfer

A remote party may transfer a call to the application. When remote transfer occurs the driver sets the `feature_info` element, *`msg`*, to `TRANSFERRED`. The application can obtain this information via `dpns_call_details()` following `EV_INCOMING_DETAILS`.

Following the `TRANSFERRED` message the driver may send the `feature_info` *`msg`* `TRANSFERRED_INFO`. The `feature_info clc` element will be set to the calling line category of the transferred party and the *`cli`* element will be set to the calling line identity of the transferred party.

3.8 DPNSS transit working

Transit working is a requirement of BTNR 188.

Following certain call scenarios; the application may no longer be directly in control of a call. For example if the application receives an incoming call, and makes an outgoing call, it can transfer the two parties (refer to section 3.6). The application has “dropped out” of the call giving control to the transferred parties.

Following this event the application need only act as a Transit PBX. Transit working changes the syntax analysis and processing required by the DPNSS signalling software on the Aculab card. The application need only pass the messages from one party to another without recognising or acting on the message contents.

The device driver enters a Transit working state when the application is required to work as a Transit PBX. This may be following call transfer, or by the application directly making a transit-working request.

Once Transit working has been established, all messages received must be passed transparently between the source and destination links via the application.

When a call receives the event `EV_DPNS_TRANSIT`, it will remain in Transit State until either the application, or one of the parties connected in Transit, clears the call. Normal call clearing applies thereafter.

When a Transit message is received from the network, the driver will generate an `EV_DPNS_IN_TRANSIT` event. This Transit message can be collected using the `dpns_transit_details()` function.

The application can send a Transit message using the `dpns_send_transit()` function.

Returning to the Call Transfer example described at the start of this section. When two parties are transferred, both calls will get the `EV_DPNS_TRANSIT` event. The application is then required to read Transit messages (using `dpns_transit_details()`) from one party and pass them unchanged to the other, and vice versa. The structure `dpns_transit_xparms`, read using `dpns_transit_details()`, is transmitted using `dpns_send_transit()`. The only parameter within `dpns_transit_xparms`, which requires changing, is the *`handle`*, which must be changed from the receiving call handle to the transmitting call handle. The contents of the message `trans_msg` remain unchanged.

3.9 Call back when free (CBWF) - BTNR 188 section 9

CBWF offers a user who meets a busy extension the possibility of having the call completed automatically when the called extension and a transmission path across the network become free. CBWF is available for both incoming and outgoing calls.

CBWF request, cancel, and free notifications should only be used with virtual calls. To use this functionality the firmware switch (`-fCBF`) should be applied.

3.9.1 Outgoing request

The application may generate a CBWF request by setting the *msg* element of feature_info to CBWF_REQUEST. The request should be initiated by using the library function dpns_openout() to make a virtual call.

The destination PBX will respond by clearing the call. The application can check the response by checking the clearing *cause* and feature_info.msg. The clearing cause will indicate whether the request has been acknowledged (0x14) or rejected (0x19). Any other clearing cause denotes failure. The feature_info.msg field will contain the current state of the called extension. Depending whether the called extension is free or busy, feature_info.msg will be set to either STATE_OF_DEST_FREE or STATE_OF_DEST_BUSY.

The application may obtain this information via dpns_call_details() following EV_REMOTE_DISCONNECT or EV_DETAILS.

3.9.2 Incoming request

An incoming virtual call may contain the CBWF_REQUEST. This would normally follow an unsuccessful call attempt. The application must respond to a CBWF_REQUEST by clearing the call (dpns_disconnect()) with the raw clearing cause set to either 0x14 (ACK) or 0x19 (REJ). In addition, feature_info.msg must be set to either STATE_OF_DEST_FREE or STATE_OF_DEST_BUSY depending on the state of dialled extension.

The application may obtain this information via dpns_call_details() following EV_INCOMING_CALL_DET or EV_INCOMING_DETAILS.

3.9.3 Outgoing free notify

A CBWF_FREE_NOTIFY is sent to indicate that the called party is now available to proceed with the call back. Of course, this should only be sent if there is a CBWF request registered against this extension.

This is done by setting feature_info.msg to CBWF_FREE_NOTIFY and making a virtual call to the party that requested the call back. The far end will disconnect the call with either a clearing cause of 0x14 (ACK) or 0x18 (FNR). In addition, the feature_info.msg will contain either STATE_OF_DEST_FREE or STATE_OF_DEST_BUSY depending on the state of the requesting extension.

The application may obtain this information via dpns_call_details() following EV_REMOTE_DISCONNECT or EV_DETAILS.

3.9.4 Incoming free notify

Once the called extension is ready to complete the call back, it will send a CBWF_FREE_NOTIFY. The application will receive this notification via a virtual call containing the CBWF_FREE_NOTIFY message in feature_info.msg. The application may obtain this information via dpns_call_details() following EV_INCOMING_CALL_DET or EV_INCOMING_DETAILS.

The application must respond to the free notify by disconnecting the call with clearing cause 0x14 or 0x18 and including the state of destination in feature_info.msg. If the Free Notify was acknowledged, the application should proceed with the call setup sequence.

3.9.5 Outgoing cancel

The application may generate a request to cancel an existing CBWF instruction. Again this is done by setting feature_info.msg to CBWF_CANCEL and making a virtual

call using dpns_openout().

The destination PBX will respond by clearing the call. The application can determine the response by looking at the clearing cause. If the call was cleared with ACK (0x14) then the CBWF_REQUEST has been cleared from the PBX. If there was no such CBWF_REQUEST registered at the PBX then the clearing cause will be “Facility Not Registered” (0x18). Any other clearing cause denotes failure.

The application may obtain this information via dpns_call_details() following EV_REMOTE_DISCONNECT or EV_DETAILS.

3.9.6 Incoming cancel

An incoming virtual call may contain the CBWF_CANCEL request in feature_info.msg if so then the application should check to see if it has a CBWF_REQUEST registered against that extension. If there is a request, then disconnect the call with clearing cause (0x14) and delete the CBWF_REQUEST from its records. If there is no such CBWF_REQUEST registered at that extension then clear the call with cause (0x18).

The application may obtain this information via dpns_call_details() following EV_INCOMING_CALL_DET or EV_INCOMING_DETAILS.

3.9.7 Outgoing call setup

Once the FREE_NOTIFY has been received and the requesting extension is free the application should initiate the call setup sequence. An outgoing call containing either CBWF_CALL_SETUP_IMMEDIATE or CBWF_CALL_SETUP_DELAYED is made to the requesting extension. Once the application receives EV_OUTGOING_RINGING then the RING_OUT feature_info.msg needs to be sent using dpns_send_feat_info(). In response the application will receive an EV_DETAILS event and feature_info.msg will have been set to CALL_BACK_COMPLETE. This signifies that the call back has been completed and the call can be treated as a simple call from this point on.

3.9.8 Incoming call setup

To complete the call back an incoming call will be received by the application. At EV_INCOMING_CALL_DET use dpns_call_details() to examine feature_info.msg this will contain either CBWF_CALL_SETUP_IMMEDIATE or CBWF_CALL_SETUP_DELAYED. In response the application should call dpns_incoming_ringing(); the far end will then send a RING_OUT message, which can be obtained via dpns_call_details() following an EV_DETAILS event. At this point, the application must use dpns_send_feat_info() to transmit the feature_info.msg CALL_BACK_COMPLETE.

Now accept the call using and treat the call as a normal call from this point on.

3.10 Add on/conference (BTNR 188 section 13)

Unless stated otherwise:

- All feature information messages are sent in feature_info.msg using dpns_send_feat_info() (Refer to section 2.5). All responses are received via dpns_call_details() following EV_INCOMING_DETAILS. Feature messages are received in feature_info.msg (refer to section 2.3).
- Call clearing is processed as Basic Call clearing.

3.10.1 Application controlled add on/conference

3.10.1.1 Conference establishment

Refer to BTNR 188 section 13 subsection 2.3.9

Following establishment of an enquiry call the application may form a conference. A

conference is established using an Add On request for both the Enquiry (refer to section 3.4) and Held (refer to section 3.5) calls. The application initiates an Add On request with the feature message ADD_ON_VALIDATION. The driver will respond with the following:

- ADD_ON_ACK - Application can proceed with conference establishment. The *cli* and *clc* information elements of feature_info contain the *CLI* and *CLC* of the remote party. The application may proceed to form a three party conference. The Enquiry and Held parties are informed of Conference establishment by sending the feature message ADDED_ON. The call enters state EV_DPNS_CONFERENCE.
- ADD_ON_REJECT - Remote PBX has rejected the Add On request. The application must abandon conference establishment.
- ADD_ON_NOT_SUPPORTED - Remote PBX does not support call conference. The application must abandon conference establishment.
- No response - If no response is received within a given time (suggested 5 seconds) the application should abandon conference establishment. Timer maintenance is the responsibility of the application.

NOTE

It is the application's responsibility to provide the relevant voice channel switching during call conference.

3.10.1.2 Active conference

Once a conference is established, the following feature messages may be received:

ADD_ON_CLEARDOWN - Application should clear both DPNSS conference parties using Basic Call clearing.

Refer to BTNR 188 Section 13 Subsection 2.3.12.

The application may send the following feature messages:

- TWO_PARTY_O/TWO_PARTY_T - If, following conference establishment, either of the two remote parties clears, the conference shall be cleared. The application may either clear or stay connected to the remaining call. If the application is to remain connected, TWO_PARTY_O or TWO_PARTY_T must be sent to the remaining party. TWO_PARTY_O indicates a return to two party call with the remaining party designated as the originating party. TWO_PARTY_T indicates a return to two party call with the remaining party designated as the terminating party. The call will return to EV_CALL_CONNECTED. If TWO_PARTY_O/T is sent following Call Hold the call will remain in EV_DPNS_HOLDING.

  Refer to BTNR 188 Section 13 Subsection 2.3.13.
- TRANSFER_O/TRANSFER_T - The application may "drop out" of a conference and transfer the Held and Enquiry calls. Call transfer following conference is initiated by sending the feature message TRANSFER_O or TRANSFER_T to the two remaining DPNSS parties. TRANSFER_O is used to designate a party as the new originating party and TRANSFER_T is used to designate a party as the new terminating party. Following call transfer the state of the remaining calls will change to EV_DPNS_TRANSIT. The application will work as a transit PBX for the remainder of the call. Refer to section 3.7 for transit working details.

  Refer to BTNR 188 Section 13 Subsection 2.3.11.

- HOLD_CALL - The application may split an established conference. Conference split enables the application to remain connected to one of the remote parties whilst the other is placed on hold. To initiate conference split the application must first place one of the parties on hold. Call hold is initiated by sending the feature message HOLD_CALL.

  If the driver acknowledges the hold request with HOLD_ACK the application may proceed and send feature message TWO_PARTY_T to both the held and connected parties. The held call will get an EV_CALL_HELD event and the connected call will get EV_CALL_CONNECTED.

  If the HOLD_CALL request is rejected (feature message HOLD_REJECT) the conference split must be abandoned. If the call_hold feature is not supported by the remote PBX (feature message HOLD_NOT_SUPPORTED) the conference split may proceed. The remote party will not be given any indication of call hold.

  Refer to BTNR 188 Section 13 Subsection 2.3.13.

3.10.2 Remote add on/conference

The remote party in an established call may include the application party in a conference. The application may receive an Add On request in call events EV_CALL_CONNECTED and EV_DPNS_HELD.

3.10.2.1 Remote conference establishment

When the driver receives an Add On request from the network the ADD_ON_VALIDATION feature message is sent to the application. The application may respond with the following:

- ADD_ON_ACK - If the application responds with ADD_ON_ACK the remote PBX will proceed with conference establishment. When all three (conference) parties are connected the application is sent the ADDED_ON feature message. The call will get the EV_DPNS_CONFERENCE event.
- ADD_ON_REJECT - Application has rejected the conference request.

  Refer to BTNR 188 Section 13 Subsection 2.3.9.

3.10.2.2 Active remote conference

After EV_DPNS_CONFERENCE, the application may receive the following feature messages:

- HOLD_REQ - Remote PBX has requested call hold. The application must respond with feature message HOLD_REJECT, or HOLD_ACK. If hold request is acknowledged the application will get EV_DPNS_HELD. If the hold request is rejected the call will remain in EV_DPNS_CONFERENCE.

  If the Hold Request is acknowledged the application will get the EV_DPNS_HELD event. The application may then receive the TWO_PARTY_O/T feature message (refer to next paragraph).

  Refer to BTNR 188 Section 13 Subsection 2.3.13.
- TWO_PARTY_O - The remote PBX has returned to a two party call following conference. The application is designated as the originating end. The remote party details are given in the *cli* and *clc* fields of feature_info when the TWO_PARTY_O feature message is received. If the call is held it will not change from EV_DPNS_HELD otherwise it will receive EV_CALL_CONNECTED.

  Refer to BTNR 188 section 13 subsection 2.3.13.

- TWO_PARTY_T - As TWO_PARTY_O with the application designated as the terminating end.

  The application may generate the following feature message:

  ADD_ON_CLEARDOWN - Once a remote party has established a conference the application may clear down all parties involved. To do this the application must send the feature *msg* ADD_ON_CLEARDOWN. On receiving the Add On Cleardown request the remote PBX will initiate call clearing.

  Refer to BTNR 188 section 13 subsection 2.3.12.

If the application wishes to clear from conference, Basic Call clearing applies.

3.11 Executive intrusion (BTNR 188 section 10)

Unless stated otherwise, all feature information messages are sent in feature_info.msg using dpns_send_feat_info() (Refer to section 2.5). All responses will generate an EV_INCOMING_DETAILS, after which, the details can be collected using dpns_call_details(). Feature messages are received in feature_info.msg.

Call clearing is processed as Basic Call clearing.

3.11.1 Application controlled intrusion without prior validation

Refer to BTNR 188 Section 10 Subsection 2.3.1.

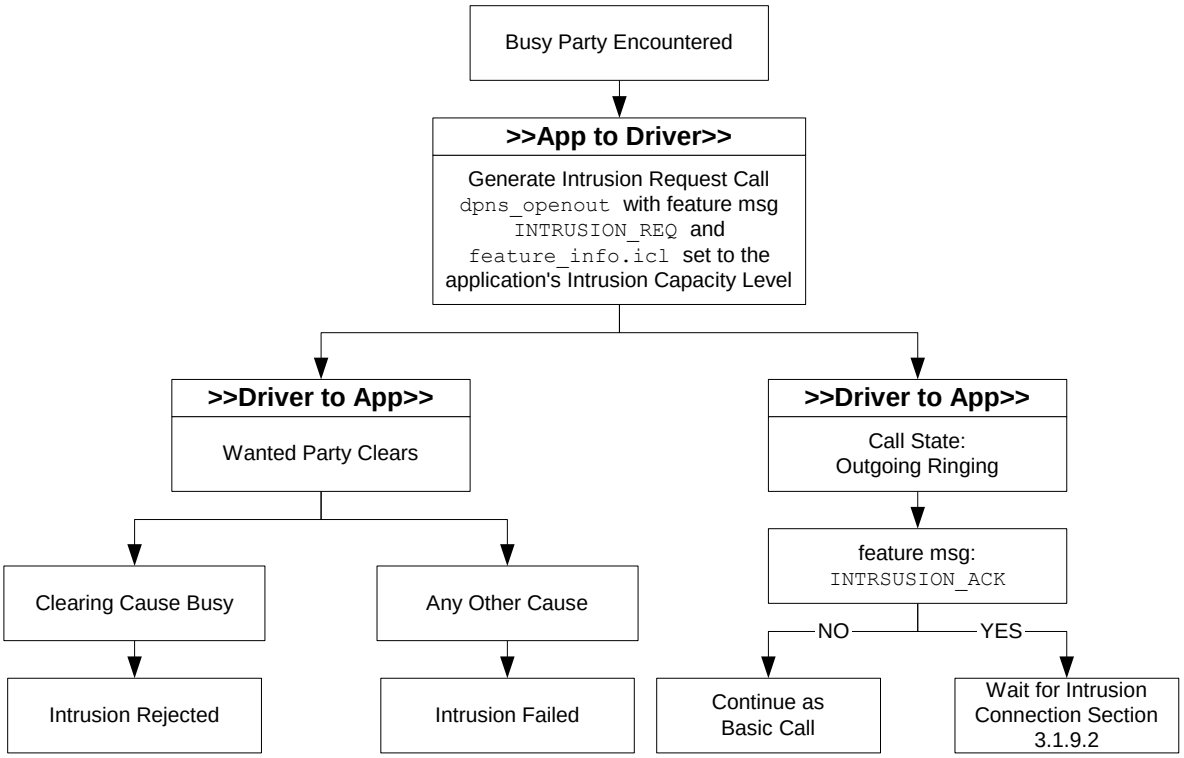
3.11.1.1 Intrusion request

If a busy remote party is encountered (on establishing an outgoing call) the application can request Executive Intrusion. The application initiates an Intrusion Request using dpns_openout() with the *msg* element of dpns_out_xparms.feature_info set to INTRUSION_REQUEST. The *icl* field of dpns_out_xparms.feature_info must be set to the Intrusion Capability Level of the Intruding party. The destination_addr of dpns_out_xparms must be set to the address of the wanted party. The *cli* field of dpns_out_xparms must be set to the address of the requesting party.

In response to the Intrusion Request, the application will receive one of the following:

- OUTGOING_RINGING - If EV_OUTGOING_RINGING is encountered the application can determine the result of the Intrusion Request by calling dpns_call_details(). If the feature message INTRUSION_ACK is present the application can proceed and wait for Intrusion connection. If INTRUSION_ACK is not present the application must assume that the wanted party has become free and has been called by the remote PBX. The call will proceed as Basic Call (wait for connection).
- IDLE / REMOTE_DISCONNECT - If the remote party clears with cause LC_NUMBER_BUSY the Intrusion Request is not allowed. Receipt of any of clearing cause indicates that the Intrusion Request has failed.

The following diagram summarises the sequence of events:

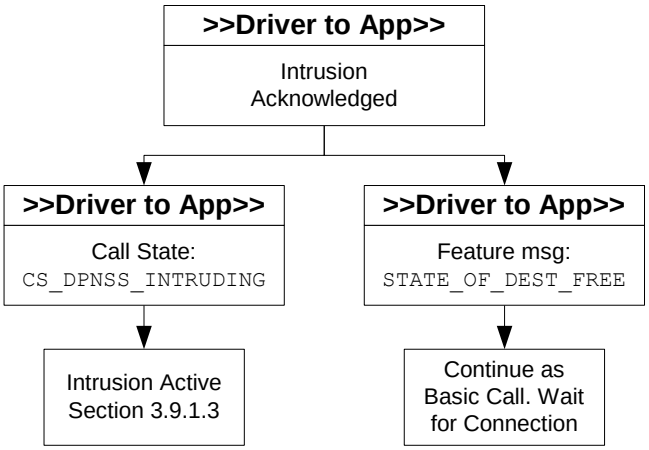


3.11.1.2 Intrusion connection

As stated in the previous section, receipt of feature message `INTRUSION_ACK` indicates success of the Intrusion Request. Subsequently one of the following events may occur:

- `INTRUDING` - If `EV_DPNS_INTRUDING` is encountered the application is intruding on the wanted party. The application can confirm this by calling `dpns_call_details()` (`feature_info.msg` is set to `INTRUDING`).
- `STATE_OF_DEST_FREE` - If the wanted party has cleared and is called by the remote PBX the driver will generate feature message `STATE_OF_DEST_FREE`. The call continues as Basic Call (wait for call connection).

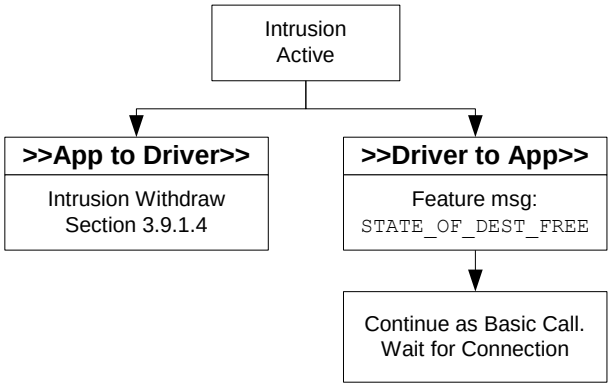
The following diagram summarises the sequence of events:



3.11.1.3 Intrusion active

Once Intrusion is active, the Intruding party may withdraw (refer to section 3.9.1.4) or the wanted party may clear. If the wanted party clears and is subsequently called by the remote PBX, the application is sent feature message `STATE_OF_DEST_FREE`. The call continues as Basic Call (wait for remote party to answer).

The diagram below summarises the sequence of events:



3.11.1.4 Intrusion withdraw

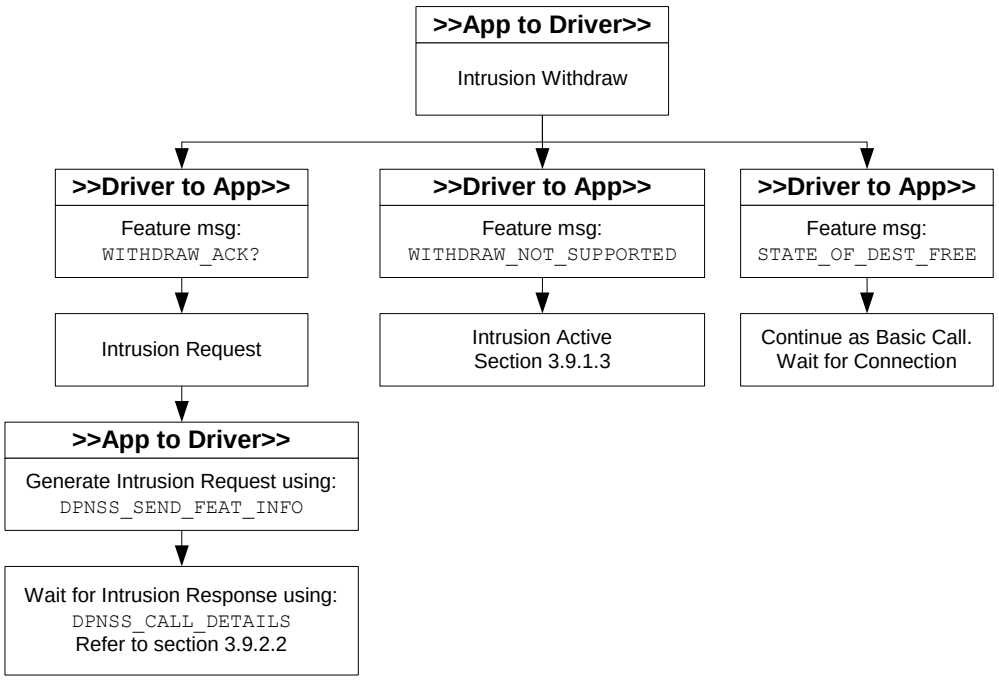
The application may temporarily withdraw from Intrusion without clearing the call.

Refer to BTNR 188 section 10 subsection 2.3.3.

To invoke Intrusion Withdraw the application must send the feature message `INTRUSION_WITHDRAW`. The driver will respond with one of the following feature messages:

- `WITHDRAW_ACK` - The application's party is no longer intruding. The application may re-enter Intrusion by sending the feature message `INTRUSION_REQUEST` (via `dpns_send_feat_info()`). The *`icl`* must again be set to the Intrusion Capability Level of the Intruding party. The Intrusion Request responses are described in section 3.9.2.2.
- `WITHDRAW_NOT_SUPPORTED` - Executive Intrusion Withdraw is not supported. The application remains Intruding.
- `STATE_OF_DEST_FREE` - If the wanted party has cleared and is called by the remote PBX the driver will generate feature message `STATE_OF_DEST_FREE`. The call continues as Basic Call.

The following diagram summarises the sequence of events:



3.11.2 Application controlled intrusion with prior validation

Refer to BTNR 188 section 10 subsection 2.3.2.

Executive Intrusion with prior validation can be used on every Basic Call setup. This facility enables the remote PBX to validate Intrusion levels during call setup if the remote party is busy.

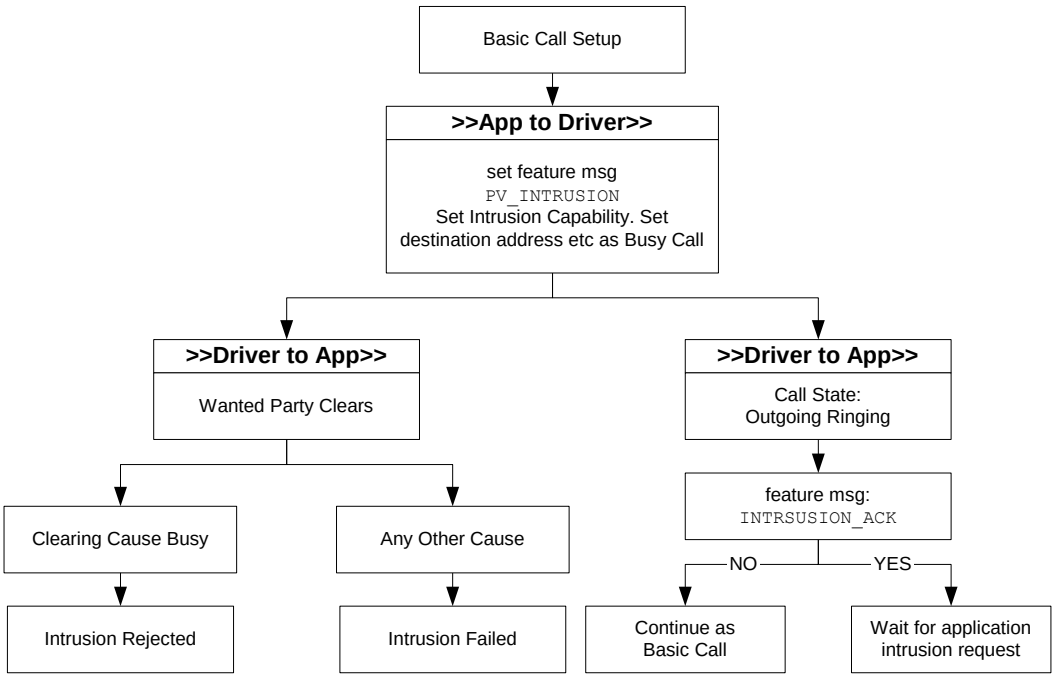
3.11.2.1 Prior validation intrusion request

The application initiates a Prior Validation Intrusion Request using `dpns_openout` with the feature message `PV_INTRUSION` and `feature_info.icl` set to the Intrusion Capability Level of the application's party. All other parameters within `dpns_out_xparms` are set as they would be for Basic Call setup.

The application may receive the following responses:

- `EV_OUTGOING_RINGING` - If the call attempt is successful the driver will respond with `EV_OUTGOING_RINGING`. The application should establish the status of the called party by examining the feature message obtained via `dpns_call_details()`. The feature message `INTRUSION_ACK` indicates that the called party is busy and Intrusion can be established. If `INTRUSION_ACK` is not present the wanted is currently free and ringing (the call continues as Basic Call).
- `IDLE / REMOTE_DISCONNECT` - If the remote party clears with cause `LC_NUMBER_BUSY` the called party is busy and Intrusion not allowed.

The diagram below summarises the sequence of events:



3.11.2.2 Prior validation intrusion establishment

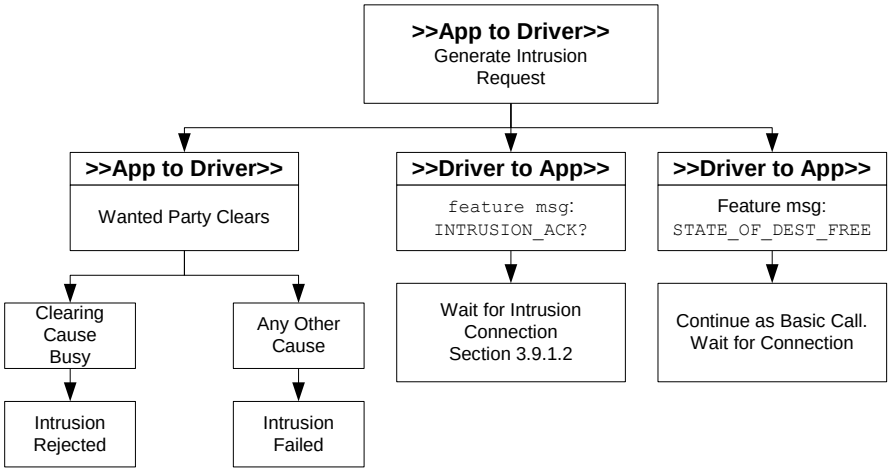
Following receipt of `INTRUSION_ACK` on `EV_OUTGOING_RINGING` the application can request Intrusion.

Intrusion is requested by sending the feature message `INTRUSION_REQUEST`. The *`feature info icl`* element must be set to the Intrusion Capability Level of the Intruding party.

In response to the Intrusion Request, the application will receive one of the following responses:

- `INTRUSION_ACK` - If the feature message `INTRUSION_ACK` is received the application waits for Intrusion connection.
- `STATE_OF_DEST_FREE` - If the wanted party has cleared and is called by the remote PBX the driver will generate feature message `STATE_OF_DEST_FREE`. The application can return to Basic Call and wait for call connection.
- `IDLE / REMOTE_DISCONNECT` - If the remote party clears with cause `LC_NUMBER_BUSY` the Intrusion Request is not allowed. Receipt of any other clearing cause indicates that the Intrusion Request has failed.

The following diagram summarises the sequence of events:



3.11.3 Network controlled intrusion without prior validation

Refer to BTNR 188 section 10 subsection 2.3.1.

A remote party may Intrude on an application-controlled party.

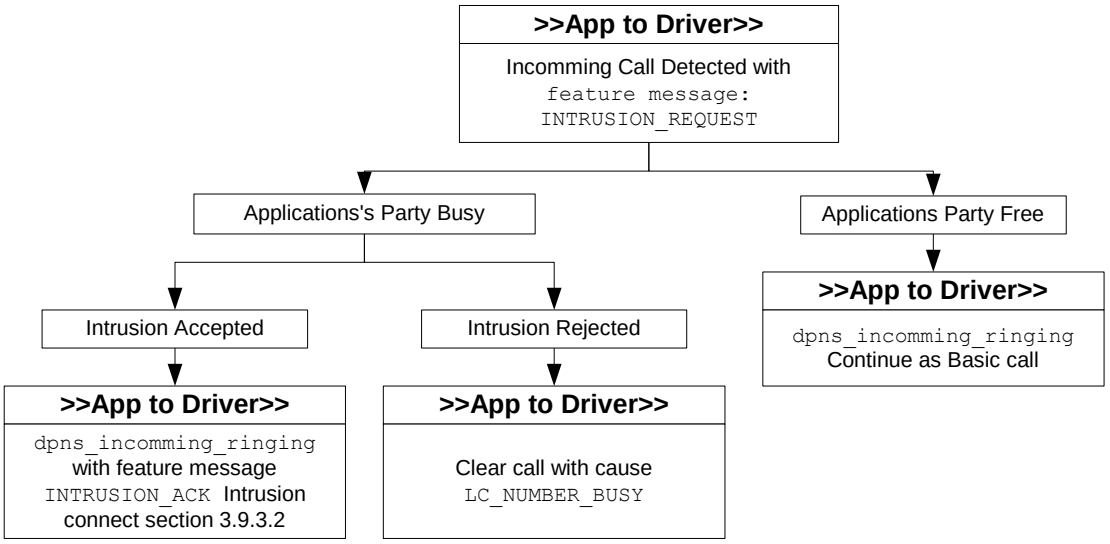
3.11.3.1 Remote intrusion request without prior validation

When a remote party requests Executive Intrusion the feature message `INTRUSION_REQUEST` is sent to the application. `feature_info.icl` is set to the Intrusion Capability Level of the Intruding party.

The application may respond to the Intrusion Request with one of the following:

- `dpns_incoming_ringing()` - To acknowledge the Intrusion Request the application must use `dpns_incoming_ringing()` with `feature_info.msg` set to `INTRUSION_ACK`. If the application's party has become free, `INTRUSION_ACK` must not be set (the call will then continue as Basic Call).
- `call_disconnect()\call_release()` - To reject the Intrusion request the application releases the call using `call_disconnect()` or `call_release()` with clearing cause `LC_NUMBER_BUSY`.

The following diagram summarises the sequence of events:

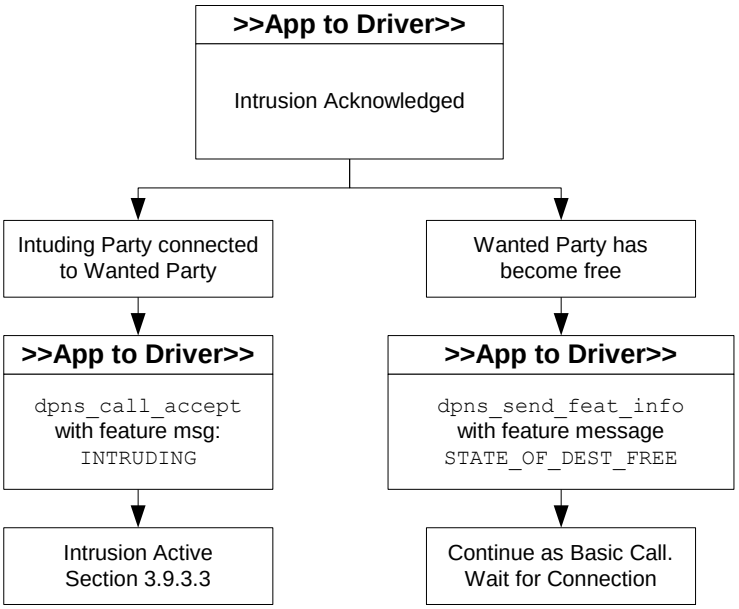


3.11.3.2 Intrusion connection

After Intrusion acknowledge, the application may use the following:

- `dpns_call_accept()` - On Intrusion accept the application must connect the wanted party to the intruding party. The application informs the intruding party of Intrusion connection by sending feature message `INTRUDING` via `dpns_call_accept()`.
- `dpns_send_feat_info()` - The application may ring the wanted party if it becomes free before intrusion connection. The application informs the intruding party of call ringing by sending feature message `STATE_OF_DEST_FREE` via `dpns_send_feat_info()`. The call continues as Basic Call (wait for call connection).

The following diagram summarises the sequence of events:



3.11.3.3 Intrusion active

Once the intruding party is connected to the wanted party, the intruding party may request withdraw, or the wanted may hang up.

The application may send the following feature message:

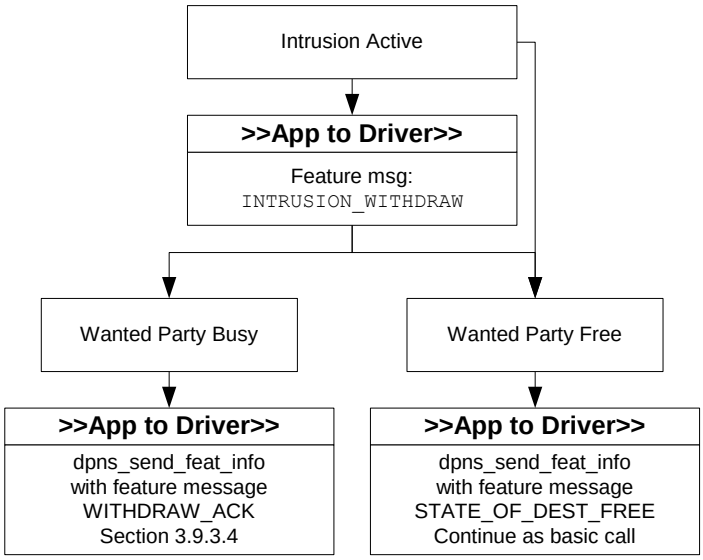
- `STATE_OF_DEST_FREE` - The wanted party is called by the application on call clearing. The application informs the intruding party of wanted party ringing with feature message `STATE_OF_DEST_FREE` sent via `dpns_send_feat_info()`.

The application may receive the following feature message:

- `INTRUSION_WITHDRAW` - The application may respond with feature message `WITHDRAW_ACK` or `STATE_OF_DEST_FREE`. `STATE_OF_DEST_FREE` is used to indicate wanted party ringing following call clearing (the call continues as Basic Call). Section 3.9.3.4 describes the actions taken following `WITHDRAW_ACK`.

Refer to BTNR 188 section 10 subsection 2.3.3.

The following diagram summarises the sequence of events:

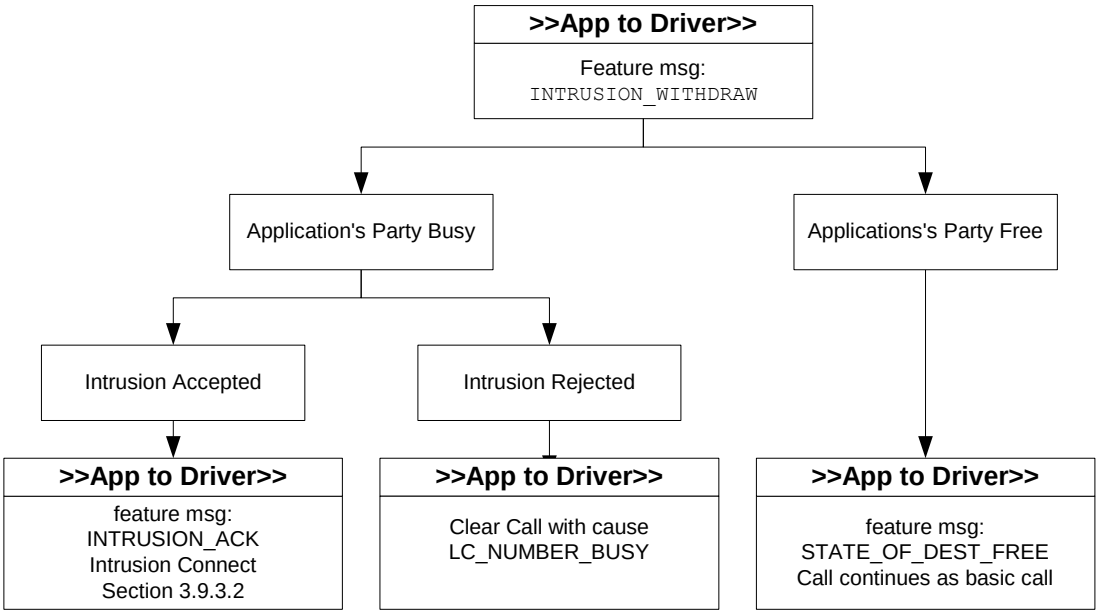


3.11.3.4 Intrusion withdraw

The application may receive a further Intrusion Request feature message `INTRUSION_REQ` via `dpns_call_details()`. The application responds with one of the following:

- `dpns_send_feat_info()` - To acknowledge the Intrusion Request the application must use `dpns_send_feat_info()` with `feature_info.msg` set to `INTRUSION_ACK`. If the application's party has become free `STATE_OF_DEST_FREE` must be sent in place of `INTRUSION_ACK` (the call will then continue as Basic Call).
- `call_disconnect{}\call_release{}` - To reject the Intrusion request the application releases the call using `call_disconnect()` or `call_release{}` with clearing cause `LC_NUMBER_BUSY`.

The following diagram summarises the sequence of events:



3.11.4 Network Controlled Intrusion With Prior Validation

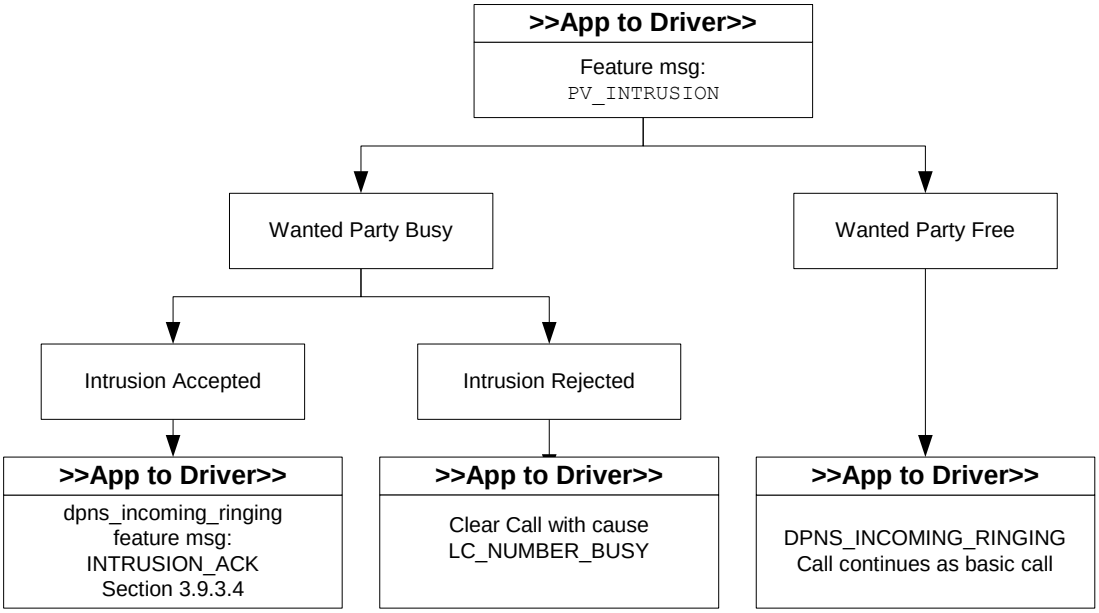
The application may receive an Intrusion Prior Validation and remote party's Intrusion Capability Level during incoming call setup. The Intrusion Capability Level is used by the application if the wanted party is found to be busy.

3.11.4.1 Intrusion Request

When an Intrusion Request with Prior Validation, is received (during incoming call setup) the application is sent feature message `PV_INTRUSION`. The application may respond with the following:

- `dpns_incoming_ringing()` - If the wanted party is busy the application may respond with `dpns_incoming_ringing()` with feature message `INTRUSION_ACK`. If the wanted party is free the call continues as Basic Call (feature message `INTRUSION_ACK` is not sent).
- `call_disconnect()/call_release()` - If the wanted party is busy and Intrusion is not possible the application clears the call with cause `LC_NUMBER_BUSY`.

The following diagram summarises the sequence of events:



3.11.5 Incoming protection request

If a third party wishes to intrude on a remote party connected to the application via DPNSS, the application will be requested to provide its Intrusion Protection Level. The application's Intrusion Protection Level is used by the remote PBX to determine if Intrusion can proceed. An Intrusion Protection Level Request will only be received when an application controlled call is connected to a remote party.

When an Intrusion Protection Level request is received the driver sends feature message `IPL_REQUEST` (received via `dpns_call_details`). The application must respond with its Intrusion Protection Level. This is sent via feature message `IPL_RESPONSE` with `feature_info.ipl` set to the protection level of the application's party. The Intrusion Protection Level response is sent via `dpns_send_feat_info`.

The application may simply choose to ignore feature message `IPL_REQUEST`. In this case, the remote Intrusion request is abandoned.

3.11.6 Outgoing protection request

If remote party requests Intrusion on an application party that is busy on another call

the Intrusion Protection Level of the party currently connected to the application must be determined.

To request the Intrusion Protection Level the application must send the feature message `IPL_REQUEST` via `dpns_send_feat_info`. The protection level request is only valid on call connection. The application may receive the following responses:

`IPL_RESPONSE` - If the feature message `IPL_RESPONSE` is received, the `ipl` element of `feature_info` is set to the Intrusion Protection Level of the party currently connected. The application uses the received Intrusion Protection Level of the remote party with that of the Intruding and its own party to determine if Intrusion can proceed.

3.12 Extension Status Calls

Refer to BTNR 188 Section 20

The Extension Status Call supplementary service offers the capability of determining, on request, the status of an extension.

Extension Status Call is available for both incoming and outgoing calls. Just one message is used in this supplementary service: `EXTENSION_STATUS_CALL`.

To use this functionality the firmware switch (`-fES`) should be applied.

3.12.1 Application Initiated Extension Status Call

An application may request the status of another extension by setting a `feature_info.msg` to `EXTENSION_STATUS_CALL` and making a virtual call to that extension.

If the called extension is free then it will disconnect the call with raw clearing cause 0x14 (ACK). If the called extension is busy then the clearing cause will be 0x08 (BY). If the called extension has diversion enabled then the relevant diversion information will be included when the call gets disconnected. Any other clearing cause denotes failure.

This information can be obtained after an `EV_REMOTE_DISCONNECT` or `EV_IDLE` by using `dpns_call_details()`.

3.12.2 Remote Initiated Extension Status Call

An incoming virtual call may contain the `feature_info.msg EXTENSION_STATUS_CALL`. The application must respond to this request by clearing the call.

If the called extension is free, it must use `dpns_disconnect` with a raw clearing cause of 0x14 (`ACK`), if the called extension is busy then it should send 0x08 (`BY`). The extension has diversion enabled then the application should include the relevant diversion information when clearing the call.

This information can be obtained after an `EV_INCOMING_CALL_DET` or `EV_DETAILS` event by using `dpns_call_details()`.

3.13 DPNSS Call Back Messaging

Refer to BTNR 188 Section 36

The Call Back Messaging supplementary service allows a caller to indicate to the called party that the calling party wishes to be called back.

Call Back Messaging is available for both incoming and outgoing calls.

Two messages are used in this supplementary service, `CALL_BACK_MESSAGE_REQ` and `CALL_BACK_MESSAGE_CAN`. To use this functionality the firmware switch (`-fCBM`) should be applied.

3.13.1 Application Initiated Call Back Request

Requesting a call back would normally take place after encountering a busy extension, no reply or by a message centre wishing to contact the called party.

In order to register a request for Call Back Messaging the application must set a `feature_info.msg` to `CALL_BACK_MESSAGE_REQ` and make a virtual call to that extension. Of course, the application needs to include the relevant *`CLI`* when making the call.

If the request is successful then the far end will disconnect the call with clearing cause 0x14 (`ACK`). Any other clearing cause denotes failure. This information can be

obtained after an EV_REMOTE_DISCONNECT or EV_IDLE by using dpns_call_details().

3.13.2 Application Initiated Call Back Cancel

In order to cancel a previously registered Call Back Message Request the application must set a feature_info.msg to CALL_BACK_MESSAGE_CAN and make a virtual call to that extension.

If the request is successful then the far end will disconnect the call with clearing cause 0x14 (ACK). Any other clearing cause denotes failure. This information can be obtained after an EV_REMOTE_DISCONNECT or EV_IDLE by using dpns_call_details().

3.13.3 Remote Initiated Call Back Request

An incoming virtual call may contain the feature_info.msg CALL_BACK_MESSAGE_REQ. The application must respond to this request by clearing the call. If the application wishes to accept the request then it must use dpns_disconnect() with a raw clearing cause of 0x14 (ACK), if the request is to be rejected then it should send 0x19 (REJ).

If the application has accepted the Call Back Message request then it needs to store the *CLI* of the requesting party.

This information can be obtained after an EV_INCOMING_CALL_DET or EV_DETAILS event by using dpns_call_details().

3.13.4 Remote Initiated Call Back Cancel

An incoming virtual call may contain the feature_info.msg CALL_BACK_MESSAGE_CAN. The application must respond to this request by clearing the call. If the application wishes to accept the request then it must use dpns_disconnect() with a raw clearing cause of 0x14 (ACK), if the request is to be rejected then it should send 0x19 (REJ).

This information can be obtained after an EV_INCOMING_CALL_DET or EV_DETAILS event by using dpns_call_details().

3.14 Charge Reporting

Refer to BTNR 188 Section 40

The Charge Reporting supplementary service allows details of call cost and associated information to be passed between the parties involved in a call.

Charge Reporting is available for both incoming and outgoing calls.

To use all of this functionality the firmware switch (-fCC) should be applied.

3.14.1 Application Initiated Charge Activation

An application may activate call charging when accepting a call by setting a feature_info.msg to CHARGE_ACTIVATE and calling dpns_call_accept(). Alternatively, the application may choose to activate charging after the call has been connected. In this case, the application must call dpns_send_feat_info() with a feature_info.msg field set to CHARGE_ACTIVATE.

3.14.2 Remote Initiated Charge Activation

When an outgoing call is accepted the far end may activate charging. Here the `feature_info.msg` field will contain `CHARGE_ACTIVATE`. Alternatively, the far end may activate charging after connection; again, the `feature_info.msg` field will contain `CHARGE_ACTIVATE`.

This information can be obtained after an `EV_CALL_CONNECTED` or `EV_DETAILS` event by using `dpns_call_details()`.

3.14.3 Application Initiated Account Code Indication

The application can send Account code details after the call has been connected. Usually account code details would be sent in response to an account code request.

To send an account code the application must set `feature_info.msg` to `CHARGE_ACCOUNT_CODE` and copy the account code string into `feature_info.digits` and use `dpns_send_feat_info()`.

3.14.4 Remote Initiated Account Code Indication

The application may receive account code details from the far end during an established call. The `feature_info.msg` will include `CHARGE_ACCOUNT_CODE` and the `feature_info.digits` field will contain the account code.

This information can be obtained after an `EV_DETAILS` event by using `dpns_call_details()`.

3.14.5 Application Initiated Account Code Request

The application can request an account code at two points. This can be either when an incoming call is accepted with `dpns_call_accept` or during an established call using `dpns_send_feat_info`.

The `feature_info.msg` element needs to be set to `CHARGE_ACCOUNT_REQUEST`. When making one of these calls.

The far end should now respond with an account code.

3.14.6 Remote Initiated Account Code Request

An account code can be requested by the far end either at call connect or during an established call. If the far end has requested an account code one of the `feature_info.msg` elements will be set to `CHARGE_ACCOUNT_REQUEST`.

This information can be obtained after either an `EV_DETAILS` or `EV_CALL_CONNECTED` event by using `dpns_call_details()`. Once obtained, the application should respond to this request by sending an account code.

3.14.7 Application Initiated Call Cost Details

The application can send unsolicited call cost details when it is disconnecting a call. To do this the `feature_info.msg` field should be set to `CHARGE_UNITS_USED` and a string containing the number of units used should be copied into `feature_info.digits`. Once this is done, the application makes a call to `dpns_disconnect` to disconnect the call.

3.15 DPNSS layer 2

The application has the ability to enable or disable DPNSS channels at Layer 2 (the data link layer). This is done using the `dpns_set_l2_ch()` function (see Section 2.13).

The application can also read a channel's Layer 2 state using the `dpns_l2_state` function (see Section 2.14).

3.16 DPNSS non specified information

Refer to BTNR 188 section 15

At any point during a call, the application or remote party may generate Non Specified Information (NSI) as defined in BTNR 188 Section 15.

NSI is sent and received via the `nsi` array located within the `feature_info` structure.

The `nsi` structure element is an array of IA5 characters with the following format:

```
Supplementary String Suffix*NSI Id*NSI String
```

The supplementary string suffix is used to determine if the NSI string is informative, optional, or mandatory. It should be set to a value as defined in BTNR 188 Section 5 Table 1. The NSI Id is the manufacturers identifier and should be set to one of the values specified in BTNR 188 Section 15.

Example of sending an NSI string:

`Mitel Id = C` (Specified in BTNR 188 Section 15)

`String Suffix = Z` Mandatory for branching, transit and end PBX (BTNR 188 Section 5 Table 1)

```
NSI String = "NSI STRING"
```

The application should copy `Z*C*NSI STRING` into the `feature_info.nsi` field.

There are two different modes of operation for passing incoming NSI strings to the application:

The default behaviour is to remove the leading `'*'`, SIS suffix (if one is present), and trailing `'#'` from the string. When there are multiple NSI strings present in one message, the driver will concatenate them together.

The same DPNSS message `"*58Z*C*NSI STRING#"` is received by the application as `"C*NSI STRING"`. Here the first '*' and SIS suffix `'Z'` have been removed, as has the trailing `'#'`.

The `-s3,1` command line option allows the application to collect the complete NSI string. With this enabled the leading `'*'` and any SIS suffix present will be passed to the application. Also the trailing `'#'` on each NSI string will be included, allowing the application to easily parse a message containing multiple NSI strings.

So the string from our example would be received as `"*Z*C*NSI STRING#"`.

The `-fNS*` command line option allows the receipt of NSI strings bearing any manufacturer code.

The `-s4,1` command line options allows the application to receive messages which contain multiple identical SIS strings in the same protocol message.

With both these options enabled, the application would now be able to receive a DPNSS message containing two NSI strings each bearing a different manufacturer code.

3.17 DPNSS text

Refer to BTNR 188 section 16

Text may be sent and received at any point during a call. Text is sent and received via the IA5 `txt` array located within the `feature_info` structure.

The Text Type parameter is appended to the end of the string.

For example to send the text message "`Aculab`" as a name the string "`Aculab*1`" is copied into the `txt` array. This is valid for both incoming and outgoing text messages.

3.18 Trunk ID

Refer to BTNR 188 section 16

This string is used in conjunction with a CLC to identify a trunk.

Trunk Identity is sent and received via the IA5 *`tid`* array within the `feature_info` structure.

To send PBX identifier "1", trunk group identifier "2", and trunk member "3" the *`tid`* is set to "1*2*3". This is valid for both incoming and outgoing calls.

3.19 Sending DPNSS raw messages

This feature has been added to allow the application writer to use facilities not provided by the Aculab API but that are available by sending messages to the switch. Please note that the Aculab API will not provide any means of retrieving any non-supported responses that the switch makes.

The feature can only be accessed when using the following functions:

```
dpns_openout
dpns_call_accept
dpns_send_overlap
dpns_incoming_ringing
dpns_send_feat_info
dpns_disconnect
```

By setting one of the message fields to `DPNSS_RAW`, whatever is in the text field will be sent out as part of the DPNSS Layer 3 message.

Example

```
DPNS_OUT_XPARMS outdetails;

INIT_ACU_STRUCT(&outdetails);

strcpy(outdetails.destination_addr, "12345");
strcpy(outdetails.originating_addr, "54321");
strcpy(outdetails.feature_info.txt, "*19*L#);

outdetail.ts                = -1;
outdetails.feature_info.msg[0]   = DPNSS_RAW;

dpns_openout(&outdetails);
```

This will result in the DPNSS Loop Avoidance message (LA *19*L#) being sent in the initial setup message. Hence, the loop avoidance message can now be supported, although no addition for this has been made to the Aculab API.

CAUTION

As this feature bypasses all Aculab parsing of messages the application should take the utmost care when using this facility. Also it is the

responsibility of the user to ensure that any features invoked through this facility exist and are supported through the PBX to which the Aculab equipment is to be interfaced.

3.20 Charge account codes

This feature has been added to enable an application to send and receive Charge Account code requests and information. Requests for an Account Code may be sent in a call connect message via the `dpns_call_accept` API call or the `dpns_send_feat_info` message after reaching the connected state. This is achieved by setting a feature message element to `CHARGE_ACCOUNT_REQUEST`. To send an Account Code in response to a message then the message should be set to `CHARGE_ACCOUNT_CODE` and the actual Account Code should be placed into the feature digits field. A call to `dpns_call_details` will reveal whether an Account Code or Account Code request has been received. The message will be contained in one of the feature message fields and the Account Code will be contained within the digits field.

Example 1

```
DPNS_CALL_ACCEPT_XPARMS accept_parms;

INIT_ACU_STRUCT(&accept_parms);

accept_parms.handle                  = the_handle;
accept_parms.feature_info.msg[0]     = CHARGE_ACCOUNT_REQUEST;

dpns_call_accept(&accept_parms);
```

Example 2

```
DPNS_FEATURE_XPARMS feature;

INIT_ACU_STRUCT(&feature);

feature.handle                       = the_handle;
feature.feature_info.msg[0]          = CHARGE_ACCOUNT_CODE;
strcpy(feature.feature_info.digits, "1968");

dpns_send_feat_info(&feature);
```

Appendix A: Command Line Switches

DPNSS features are enabled when the device driver is loaded during system initialisation.

Feature switches available are:

`-fDIB` - Enable immediate and busy diversion*

`-fDR` - Enable diversion on no reply*

`-fDV` - Enable diversion validation*

`-fHD` - Enable call hold*

`-fNSx` - Enable Non Specified Information. Where ‘x’ is the PBX manufacturer’s identity as

defined in BTNR 188 section 15

`-fNS*` - Enable receipt of NSI messages from any PBX manufacturer

`-fEN` - Enable enquiry call*

`-fTR` - Enable call transfer*

`-fEI` - Enable executive intrusion*

`-fAO` - Enable add on*

`-fCBF` - Enable call back when free*

`-fCBN` - Enable call back when next used*

`-fCBM` - Enable call back messaging*

`-fES` - Enable extension status calls*

`-fCBM` - Enables Call Back Messaging

`-fES` - Enables support for Extension Status Call's

`-fCC` - Enables support for Call Charging*

`-fNPR` - Enables protocol support for Number Presentation Restriction*

`-fFQ` - Enables driver feature queue mechanism

See the notes distributed with the Aculab DPNSS firmware for more details.

* DPNSS feature messages are detailed in section 2.1.1

Appendix B: Error Codes

The following lists the error codes returned by the call control system. Some errors are internal to the driver occurring only during initialisation and will never be seen by the application.

ERR_HANDLE	-The handle supplied is invalid
ERR_COMMAND	-The command specified is invalid or was not expected
ERR_NET	-The network OUTLET number specified is invalid
ERR_PARM	-Inconsistency in the call parameters
ERR_RESPONSE	-Application failed to respond within response time
ERR_NOCALLIP	-call_details issued with no call in progress
ERR_CFAIL	-Command failed. Error detected during the execution of the current command
ERR_TSBAR	-The specified timeslot is barred from use or an illegal timeslot number has been specified
ERR_TSBUSY	-The specified timeslot is in use or there are no free timeslots.
ERR_SERVICE	-The specified service octet or associated additional information octet is unsupported or is invalid
ERR_BUFF_FAIL	-The driver has run out of data buffer resources. This error should never be seen during normal operation

Appendix C: Feature Details Queuing

V6 now provides the option of having the driver queue all feature details before they are passed to the application. Without this is was possible for the application to miss some information if it did not collect the details quickly enough.

With this option enabled, the driver will store a set of feature details with every event. The application must use `dpns_call_details()` to collect these details after every call control event, except `EV_IDLE`, `EV_WAIT_FOR_INCOMING`, and `EV_DPNS_IN_TRANSIT`.

If `dpns_call_details()` is not used at these times then details may appear uncoordinated with the latest call control events. This slight change in API semantics is due to fact that details, which, in the past, could have been overwritten, will still be waiting for collection.

At the `EV_IDLE` event, details should be collected using the `dpns_getcause()` function.

This functionality is enabled with the addition of the –FFQ command line switch.

Contact us

Phone

+44 (0)1908 273800 (UK)
+1(781) 352 3550 (USA)

Email

Info@aculab.com

Sales@aculab.com

Support@aculab.com

Socials

Certificate number IS 722024
ISO 27001:2013

Certificate number FS722030
ISO 9001:2015