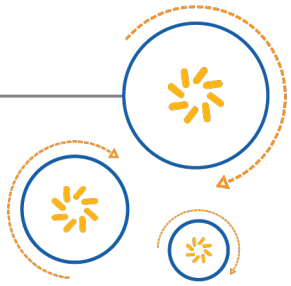
Qualcomm Technologies International, Ltd.



L2CAP Automatic Connections

Application Note

80-CT432-1 Rev. AH

October 17, 2017

Qualcomm Technologies International, Ltd. (formerly known as Cambridge Silicon Radio Limited) is a company registered in England and Wales with a registered office at: Churchill House, Cambridge Business Park, Cowley Road, Cambridge, CB4 0WZ, United Kingdom.
Registered Number: 3665875 | VAT number: GB787433096

Revision history

Revision	Date	Description
1	DEC 2008	Initial release. Alternative document number CS-00124983-AN.
2	JUL 2010	Updated to match code for 2010 SDK releases. Updated to latest style guidelines.
3	JUL 2011	Updated to the latest CSR™ style.
4	JAN 2012	Updated to the latest CSR style.
5	APR 2014	Updated to the latest CSR style.
6	SEP 2016	Updated to conform to QTI standards; No technical content was changed in this document revision
7	APR 2017	Updated for TP L2CAP auto-connect API. Added to Content Management System.
AH	OCT 2017	Updated to use Agile document numbers.

Contents

Tables

Figures

1 BlueStack L2CAP automatic connections interface

The goal of the L2CAP auto-connect system is to simplify the L2CAP channel establishment procedure for the application writer. L2CAP auto-connect provides a vastly superior and much more abstract interface to the traditional connect/configuration scheme that older versions of BlueStack L2CAP used.

The auto-connect system does not change the way BlueStack L2CAP or any other protocol interoperates with peer devices so auto-connect can safely be used with legacy and future Bluetooth devices from QTIL and other vendors.

2 L2CAP operation - overview

To understand why and how the L2CAP auto-connect system is superior to the traditional L2CAP connect/configure method, compare Figure 2-1 and Figure 2-2.

The message sequence charts (MSCs) show a typical example of the simplest, successful L2CAP channel setup.

The first figure shows the traditional L2CAP sequence for setting up and configuring a channel. In this case, the application drives L2CAP directly on both the master and the slave side, and L2CAP does not do much more than translate the application messages into over-the-air L2CAP frames and vice versa.

NOTE The application must be able to handle a large number of L2CAP primitives, and must include a negotiation algorithm to deal with discrepancies between the local and peer L2CAP configuration parameters.

The second figure shows both the master and slave using the auto-connect system. Only the initial connection request/response action is needed from the application. The rest of the configuration is taken care of by auto-connect.

NOTE This method is simpler for the application as auto-connect takes care of the complex negotiation including the flow and error control mode fallback algorithms mandated in the specification for Bluetooth 3.0 or later.

If the application developer does not want to use the auto-connect system, the legacy connect/configuration scheme can be used. The system used is selected dynamically when the application registers itself with L2CAP.

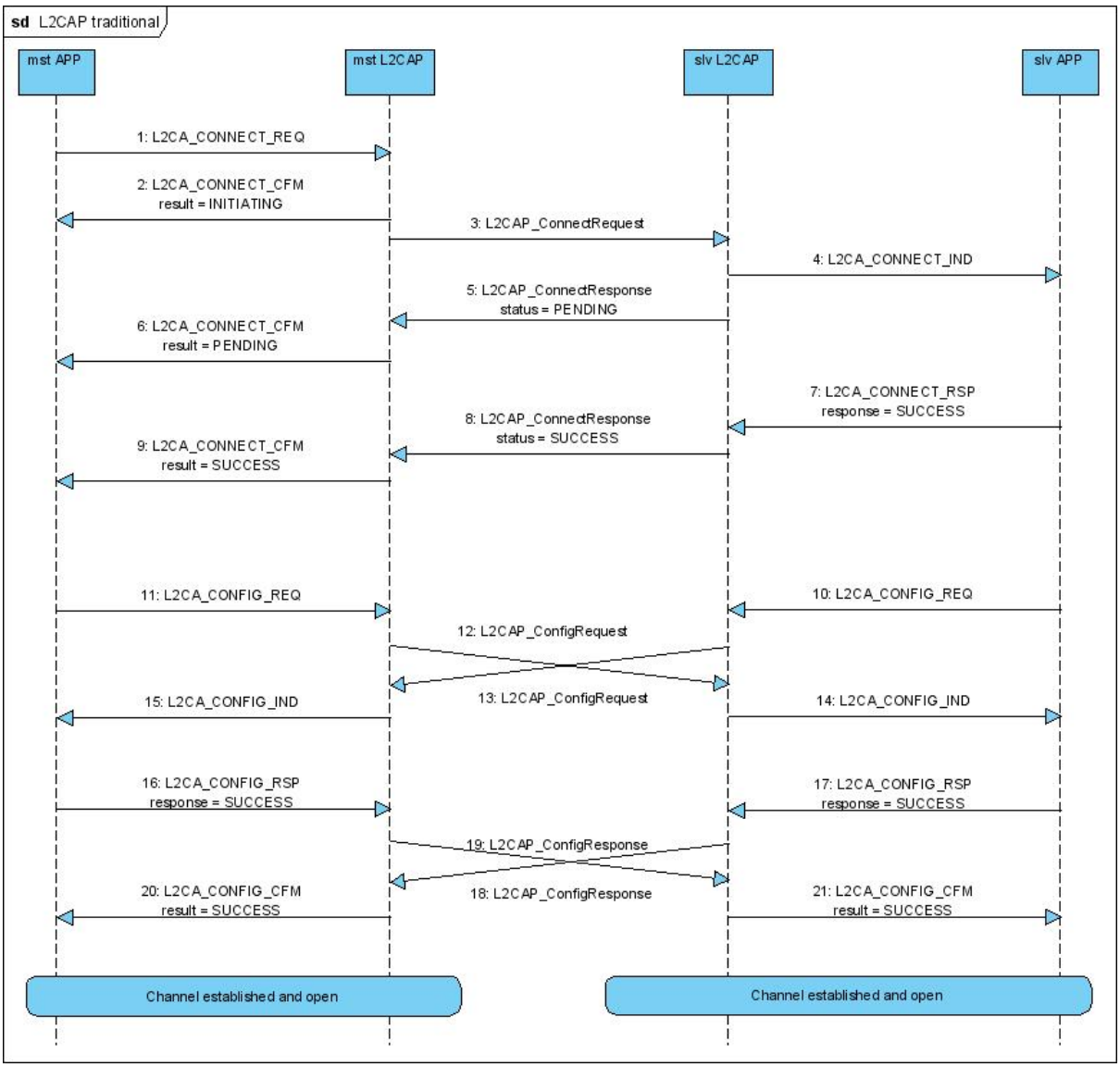


Figure 2-1 Traditional L2CAP channel establishment

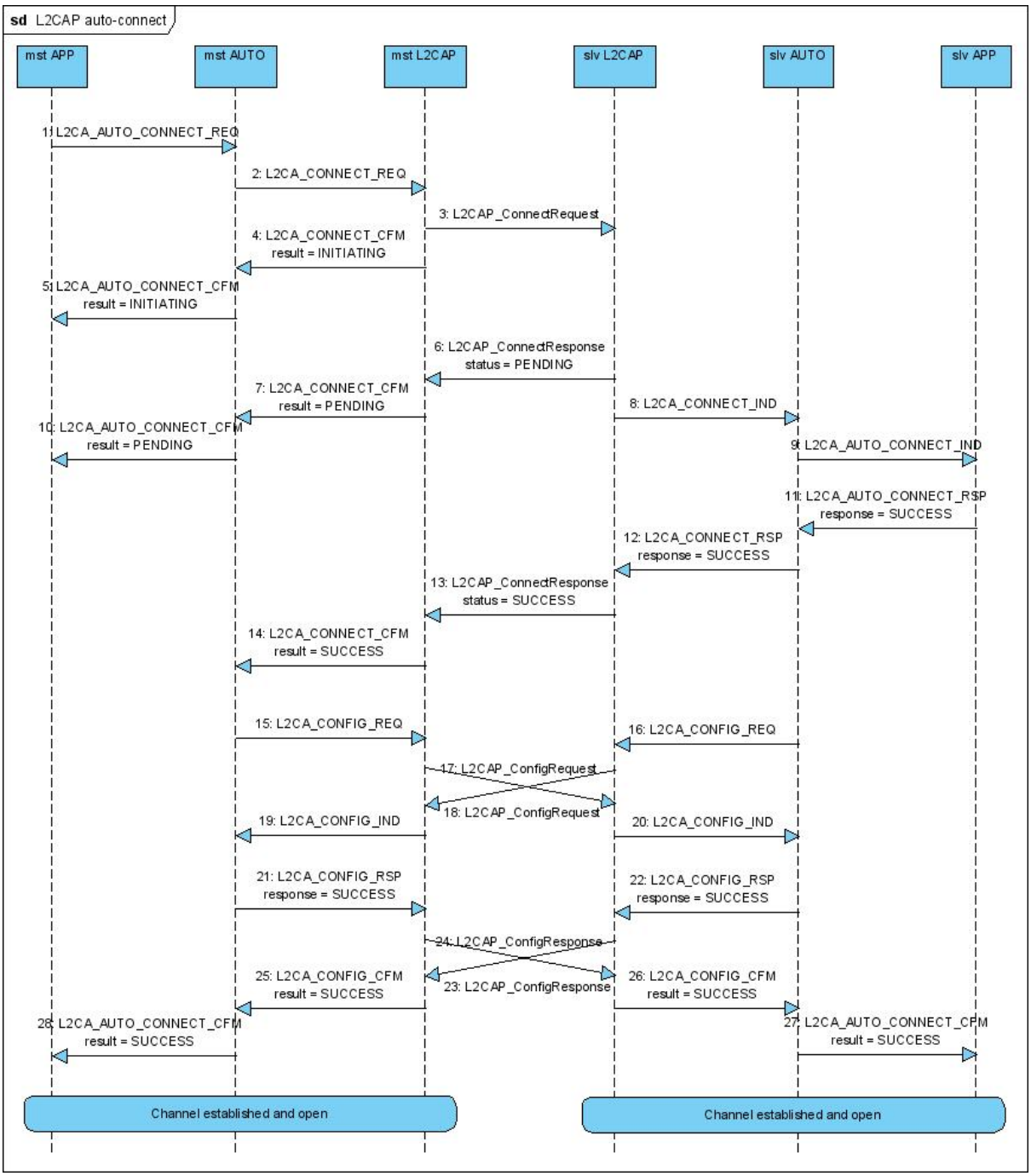


Figure 2-2 L2CAP auto-connect channel establishment

3 L2CAP auto-connect API

L2CAP auto-connect now support connections for:

- BR/EDR
- Bluetooth low enrgy technology

The auto-connect API is now known as **transport aware(TP) L2CAP auto-connect**. Additional transport related information is now carried as well as the other information required to to complete the connection procedure.

NOTE While deprecated, legacy L2CAP auto-connect APIs continue to be supported.

3.1 Transport Aware L2CAP auto-connect API

The TP L2CAP auto-connect interface is defined in the `l2cap_prim.h` header file and consists of four primitives:

- `L2CA_AUTO_TP_CONNECT_REQ`
- `L2CA_AUTO TP_CONNECT_IND`
- `L2CA_AUTO_TP_CONNECT_RSP`
- `L2CA_AUTO TP_CONNECT_CFM`

These APIs are one to one replacements of the legacy L2CAP auto-connect APIs. An application that is aware of auto-connect TP API must not use the legacy L2CAP auto connect APIs.

However, the application must register itself with L2CAP to use transport aware L2CAP auto APIs. To do this the following primitives that have been modified:

- `L2CA_REGISTER_REQ`
- `L2CA_REGISTER_CFM`
  See Auto-connect API – request and Auto-connect API – confirm

3.1.1 TP auto-connect API – request

The L2CAP TP auto-connect interface is defined in the `l2cap_prim.h` header file.

The L2CA_AUTO_TP_CONNECT_REQ primitive is used to initiate a new L2CAP channel.

Table 3-1 L2CA_AUTO_TP_CONNECT_IND structure

Name	Type	Description
type	l2cap_prim_t	Primitive identity **NOTE** This is always set to L2CA_AUTO_TP_CONNECT_REQ.
cid	L2ca_cid_t	CID to reconfigure. **NOTE** Set to 0 to create a new connection .
psm_local	psm_t	Local PSM (from the L2CA_REGISTER_REQ).
tp_addrt	TP_BD_ADDR_T	Bluetooth address of peer device including address type and transport.
psm_remote	psm_t	PSM to connect to on peer device.
con_ctx	uint16_t	Connection context. Application supplied number that is neither used nor modified by BlueCore.
remote_control	l2ca_controller_t	Reserved for future use. Must be set to 0.
local_control	l2ca_controller_t	Reserved for future use. Must be set to 0.
conftab_length	uint16_t	Number of uint16_ts in conftab.
conftab	uint16_t	This is a pointer to the Configuration Key/Value pair table for more information, see Basics of L2CAP configuration tables.

When cid in the auto-connect request is set to 0, a new L2CAP channel creation is attempted to the peer specified by tp_addrt. The tp_addrt defines the Bluetooth device address, address type and transport values along with psm information for the peer.

The con_ctx value is not used by BlueStack and can be used by the application. This number is returned in subsequent L2CAP primitives regarding this connection, for example inL2CA_AUTO_TP_CONNECT_CFM and L2CA_DISCONNECT_IND messages.

The conftab is a Key/Value pair encoded table that is used to pass configuration options to L2CAP, for more information,See Basics of L2CAP configuration tables. It is legal to pass the NULL pointer, otherwise the pointer must point to an array of uint16_ts, and the length specified in conftab_length must be set to the number of uint16_ts in the conftab.

3.1.2 TP auto-connect API – indication

The L2CAP TP auto-connect interface is defined in the l2cap_prim.h header file.

L2CA_AUTO_TP_CONNECT_IND is used to indicate to the application on the responder side when a peer has initiated a connection.

Table 3-2 L2CA_AUTO_TP_CONNECT_IND structure

Name	Type	Description
type	l2cap_prim_t	Primitive identity **NOTE** This is always set to L2CA_AUTO_TP_CONNECT_IND.
Phandle	phandle_t	Target application handle.
cid	L2ca_cid_t	Assigned CID. .
reg_ctx	uint16_t	Registration context.
Identifier	l2ca_identifier_t	Signal identifier – to be returned in L2CA_AUTO_CONNECT_RSP.
tp_addrt	TP_BD_ADDR_T	Bluetooth address of peer device including address type and transport.
psm_local	psm_t	Local PSM connected to.
local_control	l2ca_controller_t	Reserved for future use. Must be set to 0.
Flags	l2ca_conflags_t	Connection flags as defined in l2cap_prim.h header file.
conftab_length	uint16_t	Number of uint16_ts in conftab.Reserved for future use.
conftab	uint16_t	This is a pointer to the Configuration Key/Value pair table. for more information,See Basics of L2CAP configuration tables . Reserved for future use

There are no special parameters for the auto-connect indication. L2CAP expects a L2CA_AUTO_TP_CONNECT_RSP which allows the application to either accept or reject the incoming connection indication.

3.1.3 TP auto-connect API – response

The L2CAP TP auto-connect interface is defined in the l2cap_prim.h header file.

L2CA_AUTO_TP_CONNECT_RSP is the response that is sent after the application has received an L2CA_AUTO__TP_CONNECT_IND.

Table 3-3 : L2CA_AUTO_TP_CONNECT_RSP structure

Name	Type	Description
type	l2cap_prim_t	Primitive identity **NOTE** This is always set to L2CA_AUTO_CONNECT_RSP.
cid	L2ca_cid_t	L2CAP channel identifier. Must be the same CID as the L2CA_AUTO_CONNECT_IND.
Identifier	l2ca_identifier_t	Signal identifier – return the value from L2CA_AUTO_CONNECT_REQ.

Table 3-3 : L2CA_AUTO_TP_CONNECT_RSP structure (cont.)

Name	Type	Description
Response	l2ca_conn_result_t	Result code. Use L2CA_CONNECT_SUCCESS to accept a connection. Use L2CA_CONNECT_REJ_RESOURCES to reject a connection.
con_ctx	uint16_t	Connection context. Application supplied number that is not used or modified by BlueStack.
conftab_length	uint16_t	Number of uint16_ts in conftab.
conftab	uint16_t	This is a pointer to the Configuration Key/Value pair table. for more information,See Basics of L2CAP configuration tables.

The auto-connect response must always be sent when a L2CA_AUTO_CONNECT_IND has been received by the application. The response must contain the same values for identifier and cid as the request.

The con_ctx is an opaque number that can be used for application specific purposes. This value is returned in subsequent L2CAP related messages, for example L2CA_AUTO_CONNECT_CFM and L2CA_DISCONNECT_IND.

The conftab must point to the BlueStack Key/Value Pair table.

The conftab_length value identifies the number of unit16s that make up the BlueStack Key/Value Pair table. Basics of L2CAP configuration tables describes the table encoding in more detail. It is legal to pass the NULL pointer. For Bluetooth low energy transport the application can use the conftab to mention the Rx credits and MTU that can be sent to the remote device.

3.1.4 TP auto-connect API – confirm

The L2CAP TP auto-connect interface is defined in the l2cap_prim.h header file.

L2CA_AUTO_TP_CONNECT_CFM event is sent to the application on the initiating side, to signal that the auto-connect procedure has been:

- Initiated successfully (INITIATING)
- Executing for completion (PENDING)
  or
- Completed (SUCCESS/FAILURE)

It is also used to indicate the auto-connect setup is complete on the responding side.

Table 3-4 L2CA_AUTO_TP_CONNECT_CFM structure

Name	Type	Description
type	l2cap_prim_t	Primitive identity – always set to L2CA_AUTO_CONNECT_CFM.
Phandle	phandle_t	Target application handle.
cid	L2ca_cid_t	Assigned CID.
reg_ctx	uint16_t	Registration context.

Table 3-4 L2CA_AUTO_TP_CONNECT_CFM structure (cont.)

Name	Type	Description
con_ctx	uint16_t	Connection context – returned from the request/response primitive..
tp_addrt	TP_BD_ADDR_T	Bluetooth address of peer device including address type and transport.
psm_local	psm_t	Local PSM number for this connection.
local_control	l2ca_controller_t	Reserved for future use. Must be set to 0.
result	l2ca_conn_result_t	Result code, Auto-connect API – confirm.
config	L2CA_TP_CONFIG_T	L2CAP configuration results for this channel.
ext_feats	uint32_t	Peer extended features.
Flags	l2ca_conflags_t	Connection flags as defined in l2cap_prim.h header file.
config_ext_len	uint16_t	Reserved for future use. Must be set to 0.
config_ext	uint16_t	Reserved for future use. Must be set to 0.

L2CA_AUTO_TP_CONNECT_CFM is used on the initiator and responder side of a connection to indicate the status of a connection.

The additional/modified parameters compared to the legacy APIs are:

- flags which has link information that is applicable to both the transport and abides by the usage of similar primitives, such as, L2CA_CONNECT_CFM_T and ATT_CONNECT_CFM_T.
- config which is used to indicate the l2cap link configuration parameters for either transport. A modified data type ,L2CA_TP_CONFIG_T, has been added to carry this information as it requires additional parameters for Bluetooth low energy based connections. Receive-Credits, MTU_IN and MTU_OUT are part of L2CA_TP_CONFIG.

3.2 Legacy L2CAP auto-connect API

The L2CAP auto-connect interface is defined in the l2cap_prim.h header file and consists of four primitives:

- L2CA_AUTO_CONNECT_REQ
- L2CA_AUTO_CONNECT_IND
- L2CA_AUTO_CONNECT_RSP
- L2CA_AUTO_CONNECT_CFM

However, before the auto-connect API can be used the application must register itself with L2CAP. To accomplish this the following primitives have been modified to support auto-connect:

- L2CA_REGISTER_REQ
- L2CA_REGISTER_CFM

3.2.1 L2CAP auto-connect API - registration

The L2CAP auto-connect interface is defined in the `l2cap_prim.h` header file.

Before any L2CAP functionality can be used, the application must register itself and the PSM that is used for the channel. The registration is done using the primitive structure `L2CA_REGISTER_REQ` described in the table below.

Table 3-5 L2CA_REGISTER_REQ primitive structure

Name	Type	Description
`type`	`l2ca_prim_t`	Primitive identity. **NOTE** This is always set to `L2CA_REGISTER_REQ`.
`psm_local`	`psm_t`	Local PSM to register.
`phandle`	`phandle_t`	Application handle.
`mode_mask`	`uint16_t`	L2CAP Flow and Error Control modes to support.
`flags`	`uint16_t`	Register flags.
`reg_ctx`	`uint16_t`	Registration context.
`connection_oriented`	`STREAM_BUFFER_SIZES_T`	Reserved. Shall be zeroed.
`connectionless`	`STREAM_BUFFER_SIZES_T`	Reserved. Shall be zeroed.

`mode_mask`: is a bitmask that defines the L2CAP Flow and Error Control modes that the application can support. The bitmask uses mode bitmask values, described in the table below.

Table 3-6 L2CAP Flow and error control modes

Mode Identifier		Mode Bitmask		Mode Description
`L2CA_FLOW_MODE_BASIC`	(`0x00`)	`L2CA_MODE_MASK_BASIC`	(`0x01`)	Basic mode
`L2CA_FLOW_MODE_ENHANCED_RETRANS`	(`0x03`)	`L2CA_MODE_MASK_ENHANCED_RETRANS`	(`0x08`)	Enhanced Retransmission mode
`L2CA_FLOW_MODE_STREAMING`	(`0x04`)	`L2CA_MODE_MASK_STREAMING`	(`0x10`)	Streaming mode

The `mode_mask` elements can be ORed together, for example:

```
(L2CA_MODE_MASK_ENHANCED_RETRANS | L2CA_MODE_MASK_BASIC)
```

In this example `mode_mask` would enable both Basic and Enhanced Retransmission mode. For Bluetooth low energy transport `mode_mask` is set to `0`.

`flags`: is a bitfield used to enable support for reception of Unicast Connectionless Data (UCD) and/or (BCD). `flags` additionally describe:

- The application's intention to use the transport aware APIs
- To indicate that the current registration is for Bluetooth low energy transport.

The flags element uses the flag bitmask values, described in the table below.

Table 3-7 L2CAP flag definitions

Name	Value	Description
`L2CA_REGFLAG_ALLOW_RX_UCD`	`0x0001`	Enable UCD receive support for this registered PSM
`L2CA_REGFLAG_ALLOW_RX_BCD`	`0x0002`	Enable BCD receive support for this registered PSM
`L2CA_REGFLAG_USE_TP_PRIMS`	`0x0004`	Enable to indicate that application shall use transport aware APIs
`L2CA_REGFLAG_USE_LE_TP`	`0x0008`	Enable to indicate that registration being done on Bluetooth low energy transport

NOTE the LECOC feature is only available with the L2CAP transport aware API interface. Therefore, it is mandatory for the application to set `flag` to `L2CA_REGFLAG_USE_TP_PRIMS` while indicating the transport registration of Bluetooth low energy PSM

The flags elements can be ORed together, for example:

1. `(L2CA_REGFLAG_ALLOW_RX_UCD | L2CA_REGFLAG_ALLOW_RX_BCD)`
   In this case flags would allow reception of both UCD and BCD.
   or
2. `(L2CA_REGFLAG_USE_TP_PRIMS | L2CA_REGFLAG_USE_LE_TP)`
   In this case the application indicates that it communicates using TP APIs and current registration is for Bluetooth low energy transport.

`reg_ctx`: allows the application to define an opaque registration context, which is ignored by L2CAP but which is returned in subsequent upstream primitives regarding this registration.

3.2.2 Auto-connect API – request

The L2CAP auto-connect interface is defined in the `l2cap_prim.h` header file.

The L2CA_AUTO_CONNECT_REQ primitive is used to initiate a new L2CAP channel.

Table 3-8 L2CA_AUTO_CONNECT_REQ structure

Name	Type	Description
type	l2cap_prim_t	Primitive identity. **NOTE** This is always set to L2CA_AUTO_CONNECT_REQ.
cid	l2ca_cid_t	CID to reconfigure. **NOTE** To create a new connection set to 0.
psm_local	psm_t	Local PSM (from the L2CA_REGISTER_REQ).
bd_addr	BD_ADDR_T	Bluetooth address of peer device.
psm_remote	psm_t	PSM to connect to on peer device.
con_ctx	uint16_t	Connection context. Application supplied number that will neither be used nor modified by BlueStack.
remote_control	l2ca_controller_t	Reserved for future use. Must be set to 0.
local_control	l2ca_controller_t	Reserved for future use. Must be set to 0.
conftab_length	uint16_t	Number of uint16_ts in conftab.
conftab	uint16_t*	This is a pointer to the Configuration Key/Value pair table. See section Basics of L2CAP configuration tables.

When the cid in the auto-connect request is set to 0, a new L2CAP channel creation is attempted to the peer specified by the bd_addr and psm values.

The con_ctx value is not used by BlueStack and can be used by the application. This number is returned in subsequent L2CAP primitives regarding this connection, for example the L2CA_AUTO_CONNECT_CFM and L2CA_DISCONNECT_IND.

The conftab is a special Key/Value pair encoded table that is used to pass configuration options to L2CAP, see Basics of L2CAP configuration tables. It is legal to pass the NULL pointer, otherwise the pointer must point to an array of uint16_ts, and the length specified in conftab_length must be set to the number of uint16_ts in the conftab.

3.2.3 Auto-connect API – indication

The L2CAP auto-connect interface is defined in the l2cap_prim.h header file.

L2CA_AUTO_CONNECT_IND is used on the responder side when a peer wants to initiate a connection.

Table 3-9 L2CA_AUTO_CONNECT_IND structure

Name	Type	Description
type	l2cap_prim_t	Primitive identity – always set to L2CA_AUTO_CONNECT_IND.
phandle	phandle_t	Target application handle.
cid	l2ca_cid_t	Assigned CID.

Table 3-9 L2CA_AUTO_CONNECT_IND structure (cont.)

Name	Type	Description
reg_ctx	uint16_t	Registration context.
identifier	l2ca_identifier_t	Signal identifier – to be returned in L2CA_AUTO_CONNECT_RSP.
bd_addr	BD_ADDR_T	Peer Bluetooth address.
psm_local	psm_t	Local PSM connected to.
local_control	l2ca_controller_t	Reserved for future use. Will always be 0.

There are no special parameters for the auto-connect indication; L2CAP expects a L2CA_AUTO_CONNECT_RSP which allows the application to either accept or reject the connection.

3.2.4 Auto-connect API – response

The L2CAP auto-connect interface is defined in the l2cap_prim.h header file.

L2CA_AUTO_CONNECT_RSP is the auto-connect response that is sent after the application has received an L2CA_AUTO_CONNECT_IND.

Table 3-10 L2CA_AUTO_CONNECT_RSP structure

Name	Type	Description
type	l2cap_prim_t	Primitive identity – always set to L2CA_AUTO_CONNECT_RSP.
cid	l2ca_cid_t	L2CAP channel identifier. Must be the same CID as the L2CA_AUTO_CONNECT_IND.
identifier	l2ca_identifier_t	Signal identifier – return the value from L2CA_AUTO_CONNECT_REQ.
response	l2ca_conn_result_t	Result code. Either use L2CA_CONNECT_SUCCESS to accept connection, or reject with L2CA_CONNECT_REJ_RESOURCES.
con_ctx	uint16_t	Connection context. Application supplied number that is neither used nor modified by BlueStack.
conftab_length	uint16_t	Number of uint16_ts in the conftab.
conftab	uint16_t*	A pointer to the configuration Key/Value-pair table. See section Basics of L2CAP configuration tables.

The auto-connect response must always be sent when a L2CA_AUTO_CONNECT_IND has been received by the application. The response must contain the same values for identifier and cid as the request.

The con_ctx is an opaque number that can be used for application specific purposes. This value is returned in subsequent L2CAP related messages, for example L2CA_AUTO_CONNECT_CFM and L2CA_DISCONNECT_IND.

The conftab must point to the BlueStack Key/Value Pair table.

The conftab_length value identifies the number of unit16s that make up the BlueStack Key/Value Pair table. Section Basics of L2CAP configuration tables describes the table encoding in more detail. It is legal to pass the NULL pointer.

3.2.5 Auto-connect API – confirm

The L2CAP auto-connect interface is defined in the l2cap_prim.h header file.

L2CA_AUTO_CONNECT_CFM is used on the initiating or master side to signal that the auto-connect procedure has been initiated (INITIATING), is running (PENDING) or has been completed (SUCCESS).

It is also used to indicate the auto-connect setup is complete on the responding (slave) side.

Table 3-11 L2CA_AUTO_CONNECT_CFM structure

Name	Type	Description
type	l2cap_prim_t	Primitive identity – always set to L2CA_AUTO_CONNECT_CFM.
phandle	phandle_t	Target application handle.
cid	l2cap_cid_t	Channel Identifier.
reg_ctx	uint16_t	Registration context.
con_ctx	uint16_t	Connection context – returned from the request/response primitive.
bd_addr	BD_ADDR_T	Peer Bluetooth address.
psm_local	psm_t	Local PSM number for this connection.
result	l2ca_conn_result_t	Result code, see Table 3-13.
config	L2CA_CONFIG_T	L2CAP configuration results for this channel.
ext_feats	uint32_t	Peer extended features.

L2CA_AUTO_CONNECT_CFM is used both on the initiator and responder side of a connection to indicate the status of a connection.

On the initiator side it is used during connection-setup to inform the application of progress within the connection setup procedure. On the responder side the primitive is used to indicate that the connection has been established (or has failed).

The config parameter contains, upon successful channel establishment, the L2CAP configuration that auto-connect was able to achieve. As some configuration parameters are asynchronous, the config structure describes a mixture of local and remote options that the application must adhere to, see the table below.

Table 3-12 L2CA_AUTO_CONNECT_CFM config results

Config Member Name	Description	Local/Remote Option
mtu	Maximum transmission unit	Remote, not negotiated.
flush_to	Flush timeout	Local, peer may modify.
qos	Quality of Service specification (legacy)	Local, peer may modify.

Table 3-12 L2CA_AUTO_CONNECT_CFM config results (cont.)

Config Member Name	Description	Local/Remote Option
flow	Flow and Error Control	Remote, the mode is shared and the rest is negotiated.
fcs	Frame Check Sequence	Shared and negotiated.
flowspec	Extended flow specification	Local, peer may modify.
ext_window	Extended window size	Remote, local device may modify.

For normal L2CAP operations with the Qualcomm® BlueCore™ family of ICs, only the mtu, flush_to, and flow members should be used. The remaining members are either not fully supported, or require a host based system to work properly.

The table below lists the possible Error Codes that auto-connect can pass to the application in the result field of an L2CA_AUTO_CONNECT_CFM primitive.

Table 3-13 L2CA_AUTO_CONNECT_CFM Result error codes

Error Code Definition	Note
L2CA_CONNECT_SUCCESS	Channel setup complete. This is not an error.
L2CA_CONNECT_PENDING	Peer channel setup in progress. This is not an error.
L2CA_CONNECT_REJ_PSM	Peer rejects; invalid or unsupported PSM.
L2CA_CONNECT_REJ_RESOURCES	Peer rejects; insufficient resources.
L2CA_CONNECT_NOT_READY	Local device/controller is not ready.
L2CA_CONNECT_FAILED	Connection failed (page timeout).
L2CA_CONNECT_KEY_MISSING	Link key missing.
L2CA_CONNECT_TIMEOUT	L2CAP signal timeout.
L2CA_CONNECT_INVALID_CONFTAB	conftab is invalid.
L2CA_CONNECT_CONFTAB_EXHAUSTED	conftab negotiation with peer failed.
L2CA_CONNECT_CONFTAB_UNSUPPORTED	Peer does not support required extended feature.
L2CA_CONNECT_INITIATING	Local channel setup in progress. This is not an error.
L2CA_CONNECT_RETRYING	Connection attempt failed, retrying. This is not an error.
L2CA_CONNECT_PEER_ABORTED	Peer closed connection during auto-connect.
L2CA_CONNECT_OUT_OF_MEM	Not enough memory available to complete connection.

NOTE *Local* options are locally generated from the conftab, but may be modified by the peer during the L2CAP configuration phase.
Remote options originate from the peer device
Shared options are always the same on both the local device and the peer device.

A few other error codes exist, but these are purely for on-host features and are never enabled when BlueStack runs in the BlueCore firmware. The unavailable error codes include `L2CA_CONNECT_REJ_CONTROL_ID` and `L2CA_CONNECT_PHYS_FAILED`.

Additional error codes are avilable for L2CAP over Bluetooth low energy connections. These are described in the L2CAP primitive documentation.

4 Basics of L2CAP configuration tables

Central to the L2CAP auto-connect interface is the BlueStack Key/Value Pair configuration table. The table allows applications to specify, in a compact manner, the L2CAP configuration options that are important to that particular application.

NOTE Parameters that the application does not care about can usually be omitted.

A Key/Value Pair table (known as a `conftab`) is an array of `uint16_ts`. The system uses an encoding scheme that allows for four different types of `uint16_ts`:

- **Separators**
  All `conftabs` must begin with a separator. The separators allow for complex configurations and multi-point fallback negotiation, although this feature is rarely used. Most BlueStack applications only ever use the start separator in a `conftab` table.
  Key/Value Pairs always go between two separators, and are commonly referred to as a block.
- **Conftab Terminator**
  The conftab terminator is a special type of separator that marks the end of all readable blocks. It is best practice to always terminate a `conftab` table with the terminator, although it is not strictly necessary.
- **Keys**
  Keys tell L2CAP what specific parameter the application wants to specify. Keys must be unique (within a block), and they must be followed by at least one value.
  Keys have a special sub-encoding that apart from the unique number for the key also contains the encoding format of the values that follow.
- **Values**
  All keys are followed by one (or more) values. The value field specifies what L2CAP should negotiate for the named key. Four types of values exist:
  - Minimum values
  - Maximum values
  - Exact values
  - Ranges
- Values can be either 16-bit or 32-bit long.

If auto-connect is passed the NULL pointer as the `conftab` parameter, or the `conftab` Key/Value Pair table does not contain any keys (or keys for an L2CAP option is missing), a set of default L2CAP parameters are used. The default values are described in the table above.

4.1 L2CAP configuration table separator encoding

Start separators

Start Separators are encoded as a `uint16_t` where the 8 Most Significant (MS) bits are fixed, see table below.

Table 4-1 Separator encoding

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Value	1	0	0	0	0	0	0	0	R	R	R	R	R	R	R	R

The 8-bit value indicated by the Rs in the table above is used to refer to another `conftab` block that must be read first. This provides a basic inheritance system. This is an advanced `conftab` feature and should not be used by normal applications.

In general, QTIL recommends the R value is always set to zero, that is the separator has the value `1000 0000 0000 0000` or `0x8000` which is the normal Start Separator.

Termination separator encoding

The Termination Separator encoding is a completely fixed `uint16_t`, see the table below.

Table 4-2 Termination separator encoding

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Value	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0

The table above shows the termination separator has the fixed value of `1111 1111 0000 0000` or `0xff00`.

NOTE It is not strictly necessary to include the termination separator. However, QTIL recommends that it is used to ensure maximum forward compatibility with new versions of BlueStack.

4.1.1 L2CAP configuration table Key encoding

Keys are used to both uniquely identify keys and to store the encoding format of the values that follow each key.

L2CAP `#defines`, in the `l2cap_prim.h` header file, specify keys for each parameter that the application can set; including the unique key number and the encoding format of the values.

Keys use the format shown in the table below.

Table 4-3 Key encoding

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Value	0	0	0	0	S	S	T	T	K	K	K	K	K	K	K	K

Where:

- SS is the size encoding. This sets the length of the one or more values that follow this key. Only two values are defined:
  - 00: The value(s) is 16 bit wide, that is a value will use up one uint16_t.
  - 01: The value(s) is 32 bit wide, that is a value will use up two uint16_ts,

NOTE 10 and 11 are reserved values.

- TT is the type encoding. The type sets how L2CAP should interpret the value(s) given to it. The types supported are:
- 00: The value is an exact number. Only one value follows this key, and only the exact value given for this key is to be accepted.
- 01: The value is a **minimum**. Only one value follows this key. L2CAP is free to accept and negotiate for any suitable value given this minimum. That is, the actual value used may be higher than or equal to (but not lower) than what the application asked for.
- 10: The value is a **maximum**. Only one value follows this key. L2CAP is free to accept and negotiate for any value which is less than or equal to the specified value.
- 11: Two values are used to specify a **range** which can be accepted. A key that specifies a range always have two values following the key. The values can be either *lower-upper* bounds or *upper-lower* bounds depending on whether the lower or higher value goes first. The ordering is significant in the sense that the latter value is the preferred, that is if a *lower-upper* range is specified, L2CAP will try to negotiate for the higher number, but if an *upper-lower* range is used, L2CAP will try to negotiate for the lower number.
- KKKK KKKK is the unique key number. The key numbers are defined in the l2cap_prim.h header file.

4.1.2 L2CAP configuration table conftab examples

conftab examples indicating how the scheme is used.

NOTE The examples are not necessarily tied to L2CAP.

Example 1

Conftab	0x8000	0xff00
Note	Start separator	Termination separator

Example 2

Conftab	0x8000	0x0001	0x1234	0xff00
Note	Start separator	Key (that is, SS = 00, TT = 00, KK…K = 00000001)	16-bit, exact	Termination separator

Example 3

Conftab	0x8000	0x0102	0x1234	0x0203	0x5678	0xff00
Note	Start separator	Key (that is, SS = 00, TT = 01, KK…K = 00000010)	Value: 16-bit minimum	Key (i.e. SS = 00, TT = 10, KK…K = 00000011)	Value: 16-bit maximum	Termination separator

Example 4

Conftab	0x8000	0x0705	0x1111	0x2222	0x3333	0x4444	0xff00
Note	Start separator	Key (i.e. SS = 01, TT = 10, KK…K = 00000101)	Value, 32-bit range lower, MS	Value: 32-bit range lower, LS	Value: 32-bit range, higher, MS	Value: 32-bit range higher, LS	Termination separator

Description of the examples

- **Example 1:** Shows an empty `conftab`. Only the start separator (`0x8000`) and termination separator (`0xff00`) are used.
- **Example 2:** Shows a `conftab` table with one key present， `0x0001`, which denotes that one 16-bit exact value follows, in the example `0x1234`.
- **Example 3:** Shows a `conftab` table with two keys present. The first key, `0x0102`, denotes a 16-bit minimum value with key number 2. The value for this key is `0x1234`. The second key, `0x0103`, denotes a 16-bit maximum value with key number 3. In the example this maximum value is `0x5678`.
- **Example 4:** Shows a `conftab` table with one key present, `0x0705`, which denotes a 32-bit range parameter for each of two following values, which in the example are `1111 2222` and `3333 4444`.

NOTE The ordering of two values can be seen to form a lower-higher range, in this case L2CAP tries to maximize the actual value during the negotiation. If the range is higher to lower the L2CAP tries to minimize the actual value during negotiation.

5 L2CAP parameter negotiation

After a `conftab` has been passed to L2CAP either through a `L2CA_AUTO_CONNECT_REQ` or a `L2CA_AUTO_CONNECT_RSP` and the initial channel establishment is completed, the L2CAP configuration stage starts.

The configuration is two-way, asynchronous and possibly simultaneous. To describe the underlying method, the method is described in two parts, a request and response part.

NOTE The configuration stage cannot complete unless both parts are completed.t

L2CAP auto-connect always initiates the *request* part immediately and deals with the generation of `L2CAP_ConfigRequest` signal frames. The *response* part receives and decodes `L2CAP_ConfigRequests` sent from the peer, and replies with an `L2CAP_ConfigResponse`.

5.1 L2CAP request parameter negotiation

The *request* method consists of the following steps:

1. Read, parse and perform basic sanity checks of the user-supplied `conftab`. The `conftab` is turned into a set of internal values that describe the required options and a similar structure that describes the preferred options.

   NOTE These internal structures allow exact, minimum, maximum and ranged values to be defined.

2. If the peer has previously responded to an `L2CAP_ConfigRequest`, put this reply into a peer options structure, taking the `L2CAP_ConfigResponse` error code into account.
   If no information from the peer is available, generate a default peer options structure using the preferred options read in from the `conftab`.
3. For each option in the combined preferred options, required options and peer options, see if an L2CAP configuration option can or shall be used:
4. If option is required, select optimal option value based on preferred, required and peer option values. If peer value is outside legal boundary, Fail, otherwise prepare optimal value for transmission in `L2CAP_ConfigRequest`.
5. If option is preferred and peer supports option, select optimal value as before. If peer option is outside legal boundary, ignore option completely.
6. If not failed, send `L2CAP_ConfigRequest`.
7. If the optimal value generation in step 5 failed, auto-connect tries the next `conftab` block if one is available. If no more blocks are available (which is the normal case when only one block is specified), auto-connect fails and the channel is disconnected. The application is informed of this in a `L2CA_AUTO_CONNECT_CFM` with the result code `L2CA_CONNECT_CONFTAB_EXHAUSTED`.

5.2 L2CAP response parameter negotiation

The *response* method consists of a similar set of steps:

1. Read, parse and perform basic sanity checks of the user-supplied `conftab`. The `conftab` is turned in to a set of internal values that describe the *required* options and a similar structure that describes the *preferred* options.

   NOTE These internal structures allow exact, minimum, maximum and ranged values to be defined.

2. Set default result code to Success.
3. For each option in the combined preferred, required and peer options, see if an L2CAP option can or shall be used:
4. If peer option is not supported, set overall result code to `Failed due to unacceptable parameters`.
5. If peer option is required, select optimal option value based on preferred, required and peer option values.
   If peer value is outside legal boundary, set overall result code to `Failed due to unacceptable parameters`. Unconditionally prepare the optimal value for transmission in the `L2CAP_ConfigResponse`.
6. Unconditionally send `L2CAP_ConfigResponse` with result code based on results.

   The two configuration methods run in a loop that only stops on full failure or full completion.

   NOTE There are built-in safeguards to avoid becoming stuck in the configuration loop, and there is error detection code to deal with invalid peer behavior as mandated by the specification for Bluetooth 3.0 or later.

6 conftab keys defined by L2CAP

Table 6-1 lists all possible L2CAP auto-connect keys based on the `l2cap_prim.h` header file.

NOTE The prefix `L2CA_AUTOPT_` has been omitted from the defined names of the L2CAP conftab Keys to improve readability, for example, the full defined name for `MTU_IN` is `L2CA_AUTOPT_MTU_IN`.

Table 6-1 L2CAP auto-connect conftab keys

Key Define	Key Value	Format Type, Value	Description	Defaults
MTU_IN	0x0001	16-bit, Exact	Locally accepted MTU	672
MTU_OUT	0x0102	16-bit, Minimum	Minimum MTU that peer must accept	48
FLUSH_IN	0x0703	32-bit, Range	Flush timeout (µs) peer must use	0xffffffff to 0xffffffff
FLUSH_OUT	0x0704	32-bit, Range	Flush timeout (µs) applied on outgoing traffic	0xffffffff to 0xffffffff
QOS_SERVICE	0x0005	16-bit exact	QoS service type (n/a)	Best effort
QOS_RATE_IN	0x0706	32-bit, Range	QoS peer token rate (n/a)	0x00000000 to 0xffffffff
QOS_RATE_OUT	0x0707	32-bit, Range	QoS outgoing token rate (n/a)	0x00000000 to 0xffffffff
QOS_BUCKET_IN	0x0708	32-bit, Range	QoS peer token bucket (n/a)	0x00000000 to 0xffffffff
QOS_BUCKET_OUT	0x0709	32-bit, Range	QoS outgoing token bucket (n/a)	0x00000000 to 0xffffffff
QOS_PEAK_IN	0x070a	32-bit, Range	QoS/flowspec peer peak bandwidth (n/a)	0x00000000 to 0xffffffff
QOS_PEAK_OUT	0x070b	32-bit, Range	QoS/flowspec outgoing peak bandwidth (n/a)	0x00000000 to 0xffffffff
QOS_LATENCY_IN	0x070c	32-bit, Range	QoS/flowspec peer access latency (n/a)	0x00000000 to 0xffffffff
QOS_LATENCY_OUT	0x070d	32-bit, Range	QoS/flowspec outgoing access latency (n/a)	0x00000000 to 0xffffffff
QOS_DELAY_IN	0x070e	32-bit, Range	QoS peer delay variation (n/a)	0x00000000 to 0xffffffff
QOS_DELAY_OUT	0x070f	32-bit, Range	QoS outgoing delay variation (n/a)	0x00000000 to 0xffffffff

Table 6-1 L2CAP auto-connect conftab keys (cont.)

Key Define	Key Value	Format Type, Value	Description	Defaults
FS_SDU_SIZE_IN	0x0310	16-bit, Range	Flowspec incoming SDU size (n/a)	0 to *672*
FS_SDU_SIZE_OUT	0x0311	16-bit, Range	Flowspec outgoing SDU size (n/a)	0 to *672*
FLOW_MODE	0x0012	16-bit, Exact	Flow and Error Control (F&EC) mode (*special*)	Basic mode
FLOW_WINDOW_IN	0x0313	16-bit, Range	F&EC local incoming window size	1 to *5*
FLOW_WINDOW_OUT	0x0314	16-bit, Range	F&EC outgoing (peer) window size	1 to *63*
FLOW_MAX_RETX_IN	0x0315	16-bit, Range	F&EC peer maximum retransmit	0 to *255*
FLOW_MAX_RETX_OUT	0x0316	16-bit, Range	This option is now ignored. It is set by the peer.	0
FLOW_MAX_PDU_IN	0x0317	16-bit, Range	F&EC incoming maximum accepted PDU size	48 to *895*
FLOW_MAX_PDU_OUT	0x0318	16-bit, Range	F&EC outgoing maximum PDU size	48 to *895*
FCS	0x0019	16-bit, Exact	Frame Check Sequence (CRC) (*special*)	On
FS_SERVICE	0x001a	16-bit, Exact	Extended flow specification service mode (*n/a*)	Best effort
EXT_FEATS	0x0420	32-bit, Exact	Peer cached extended features (*special*)	N/A
DISABLE_RECONF	0x0021	16-bit, Exact	L2CAP reconfiguration disable (*special*)	False (that is, allow reconfigurations)
SECURITY_PROTOCOL	0x0022	16-bit, Exact	Reserved. Shall not be used.	N/A
SECURITY_CHANNEL	0x0023	16-bit, Exact	Reserved. Shall not be used.	N/A
CHANNEL_PRIORITY	0x0024	16-bit, Exact	Reserved. Shall not be used.	N/A
CREDITS_IN	0x0025	16-bit, Exact	Credits initially extended to the remote device while creating connection	0
CONN_FLAGS		16-bit, Exact	Holds the l2cap connection flags parameters indicating the connection options to bring up the link.	L2CA_CONNECTION_BR_EDR for BR/EDR transport. L2CA_CONNECTION_LE_MASTER_DIRECTED for Bluetooth low energy transport.

The table above lists all assigned L2CAP auto-connect `conftab` keys and the value encoding. Most keys are simple exact/minimum/maximum values or ranges that can be entered directly into a `conftab`.

If the conftab is left out entirely (for example, by passing the NULL pointer as the `conftab` parameter), or part of an L2CAP configuration option is missing, the default values listed in the **Defaults** column of the table are used.

When the Default sets a range for a particular key the preferred value is shown in *italics*.

The keys that are marked *n/a* are either not fully supported, or require a host system to work properly. These features are normally not available when BlueStack runs embedded on the BlueCore firmware.

Keys that require special values to work are identified in the **Description** column, that is, (*special*) appears at the end of the description:

- L2CA_AUTOPT_FLOW_MODE
  This 16-bit exact value is a combination of two 8-bit values:
- The most significant octet is the preferred F&EC mode
- The least significant octet is the allowed fallback F&EC modes.

The preferred mode is set using the appropriate L2CA_FLOW_MODE definition, the fallback modes by a bitmask that is created by ORing together the allowed modes using the L2CA_FLOW_MASK definitions.
Table 3-6 lists both the possible values for the *preferred* mode (in the **Mode Identifier** column) and the *fallback* modes (in the **Mode Bitmask** column).
For example:
To request enhanced retransmission mode, and to allow fallback to basic mode, the value is:

```
0x0301
as
0x03 = L2CA_FLOW_MODE_ENHANCED_RETRANS and 0x01 = L2CA_FLOW_MASK_BASIC.
```

To use and allow only basic mode, the value is:

`0x0000`
as
`0x00` = `L2CA_FLOW_MODE_BASIC` and `0x00` does not allow any fallback modes.

`L2CA_AUTOPT_FCS`
Set to either:

- `1` (on, default)
- `0` (off).

NOTE On always takes precedence, so it is not possible to guarantee that the FCS is turned off. This is mandated by the specification for Bluetooth 3.0 or later.

- `L2CA_AUTOPT_EXT_FEATS`
  It is possible for the application to cache the peer's L2CAP extended features bitmask and pass it in to auto-connect using this key. This allows auto-connect to skip the `L2CAP_GetInfoRequest` step during connection setup. If this key is left out, auto-connect automatically queries the peer for the supported features.
- `L2CA_AUTOPT_DISABLE_RECONF`
  This key controls whether L2CAP should prohibit the peer from initiating L2CAP

reconfigurations. If the value is set to 0xffff, all such reconfigurations are warded off. The application is not told of these reconfiguration attempts if this feature is used.

6.1 Examples of typical L2CAP conftabs

Typical L2CAP conftab examples tailored for both legacy applications and applications that must use the new L2CAP flow and error control modes.

Example 1

Table 6-2 L2CAP basic mode example conftab

conftab	Description
0x8000	Start separator
0x0001 0x02a0	Incoming (local) MTU = 672
0x0102 0x0030	Remote (outgoing) MTU >= 48
0x0703 0xffff 0xffff 0xffff 0xffff	Incoming (peer) flush timeout [0xffffffff - 0xffffffff] = 0xffffffff (infinite)
0x0704 0xffff 0xffff 0xffff 0xffff	Outgoing (local) flush timeout [0xffffffff – 0xffffffff] = 0xffffffff (infinite)
0x0021 0xffff	Ward off all reconfiguration attempts
0xff00	Termination separator

Example 2

Table 6-3 L2CAP enhanced retransmission mode example conftab

conftab	Description
0x8000	Start separator
0x0001 0x02a0	Incoming (local) MTU = 672
0x0102 0x0030	Remote (outgoing) MTU >= 48
0x0012 0x0301	Flow control mode 0x03 = Prefer ERTM, 0x01 = Allow basic
0x0313 0x0001 0x0002	Flow control mode, incoming window size [1 – 2], prefer the higher value
0x0021 0xffff	Ward off all reconfiguration attempts
0xff00	Termination separator

Example 3

Table 6-4 L2CAP streaming mode with multiple fallback modes

conftab	Description
0x8000	Start separator
0x0012 0x0419	Flow control mode 0x04 = Prefer SM, 0x19 = Allow basic, ERTM and SM
0x0704 0x0000 0x0000 0x0000 0x0000	Outgoing (local) flush timeout [0x0 – 0x0] = 0 (immediate – no retry)
0x0021 0xffff	Ward off all reconfiguration attempts
0xff00	Termination separator

Document references

Document	Reference
Logical Link Control and Adaption Protocol Specification	Bluetooth Core Specification Addendum 1

Terms and definitions

Term	Definition
API	Application Programming Interface
Auto-connect	BlueStack L2CAP automatic connection interface
BCD	Broadcast Connectionless Data
BlueCore	Group term for Qualcomm's range of Bluetooth wireless technology ICs
BlueStack	Qualcomm's implementation of the core protocols; L2CAP, RFCOMM, SDP, etc.
Bluetooth	Set of technologies providing audio and data transfer over short-range radio connections
CID	L2CAP channel identifier
Conftab	BlueStack key/value-pair configuration table
ERTM	Enhanced retransmission mode
F&EC	Flow and Error Control
IC	Integrated Circuit
L2CAP	Logical Link Control and Adaption Protocol
LS	Least Significant
MS	Most Significant
MSC	Message Sequence Chart
mst/MST	Master
MTU	Maximum Transmission Unit
PDU	Protocol Data Unit
PSM	Protocol Service Multiplexor
QoS	Quality of Service
RFCOMM	Radio Frequency COMMunication
RX	Receive
SDP	Service Discovery Profile
slv/SLV	Slave
SM	Streaming mode
UCD	Unicast Connectionless Data
uint16/uint16_t	Unsigned integer 16-bit type
uint32/uint32_t	Unsigned integer 32-bit type