Innovative Technology
INTELLIGENCE IN VALIDATION

Automated Transactions
INTELLIGENCE IN VALIDATION

Bellis Technology
INTELLIGENCE IN VALIDATION

INNOVATIVE TECHNOLOGY SSP LIBRARY AND EXAMPLES

1. PACKAGE CONTENTS

There are 5 folders included in the package.

BasicValidator: Example project showing how to setup and communicate with a validator.

DownloadValidator: Example project showing how to download a file to a validator.

inc: The header files for the library

lib: This is the libitlssp library source. When compiled, a static library (.a) will be placed in the lib/bin directory, and a shared(.so) library will be placed in the lib/bin/shared directory.

ValidatorInfo: Example project showing the data available using standard ssp commands.

2. COMPILATION

Each project (and the main library) has a Makefile. If you type 'make' in the base directory, all 3 projects and the library will be compiled in their directories. The only requirements are gcc, make and libpthreads.

3 USB SERIAL DEVICES

3.1 DA2

DA2 devices can be used in linux using the ftdi_sio module. To do this, make sure the module is not loaded.

rmmod ftdi_sio
modprobe ftdi_sio vendor=0x0403 product=0xf248

This should then create a device called /dev/ttyUSB0

3.2 BUILT-IN USB

The built in USB can be used in linux using the usbserial module. To do this, make sure the module is not loaded.

rmmod usbserial
modprobe usbserial vendor=0x191c product=0x4104

This should then create a device called /dev/ttyUSB0

5. BASICVALIDATOR6

Usage: BasicValidator6

BasicValidator6 demonstrates sending commands in eSSP version 6. For further details on the differences between version 6 and earlier versions see the eSSP specification, available from support@innovative-technology.co.uk.

When run, the program will prompt you to enter the port and the SSP address. Examples are /dev/ttyUSB0 for a USB serial port (DA2, Built-In USB), /dev/ttyS0 for the first standard serial port.

Once the port and SSP address have been entered, the program will configure the validator to eSSP version 6 using the 'Host Protocol Version' command (0x06) and enable it , then send a poll every 500msec. Any poll events will be printed to the console window. The validator will be enabled and validate notes, showing Credit events as they occur. It will also try to setup SSP Encryption using the default key, and if successful all commands will be sent encrypted.
Several eSSP commands can be sent to the validator, demonstrating some of the basic features of eSSP version 6. Pressing 'h' or '?' will display a list of the available commands and the key press which will send them to the validator.

5. DOWNLOADVALIDATOR

Usage DownloadValidator <port> <sspaddress> <file>

Port is the port to be used. Examples are /dev/ttyUSB0 for a usb serial port (DA2, Built-In USB), /dev/ttyS0 for the first standard serial port.
SSPAddress is the address of the validator. This is usually 0.
File is the ITL firmware or dataset file to download.

When run the program will attempt to download the file into the unit, displaying progress on the console window.

6. VALIDATORINFO

Usage: ValidatorInfo <port> <sspaddress>

Port is the port to be used. Examples are /dev/ttyUSB0 for a usb serial port (DA2, Built-In USB), /dev/ttyS0 for the first standard serial port.
SSPAddress is the address of the validator. This is usually 0.

When run, the program will query the validator using the ssp_setup_request, ssp_unit_data, ssp_get_serial, ssp_channel_data and ssp_security_data commands. It will also try to setup SSP Encryption using the default key, and if successful all commands will be sent encrypted.

7. LIBRARY

When compiled, the library will create two files. The first is a static library file (libitlssp.a), in the lib/bin directory. The second is a shared library (libitlssp.so) in the lib/bin/shared directory. To use the library, you need to include the "SSPComs.h" header file in the inc directory, and tell your linker to link to one of the two librarys as needed.

7.1 SSPSENDCOMMAND

int SSPSendCommand(const SSP_PORT,SSP_COMMAND cmd);*

Used to an SSP Command to a unit. The first parameter needs to be a valid SSP_PORT returned from OpenSSPPort. The second is a pointer to a SSP_COMMAND structure. The structure needs to have EncryptionStatus (1 if encrypted, 0 if unencrypted), SSPAddress, Timeout(milliseconds to wait for a reply), RetryLevel (number of times command is retried), CommandData (array containing the data to send) and CommandDataLength (length of data to send) set.

If the command fails, 0 will be returned. If it is successful, 1 is returned and ResponseStatus, ResponseData and ResponseDataLength will be set in the structure.

7.2 OPENSSPPORT

*SSP_PORT OpenSSPPort(const char * port);*

Opens a port for use with SSP communications. Port is the serial device you wish to send on (eg /dev/ttyUSB0). Returns -1 on failure, any other value is the SSP_PORT variable used to communicate on that port.

7.3 CloseSSPPort

void CloseSSPPort(const SSP_PORT port);

Closes the specified port. port should be the SSP_PORT variable returned from a successful OpenSSPPort call.

7.4 DOWNLOADFILETOTARGET

*int DownloadFileToTarget(const char * file, const char * port, const unsigned char sspAddress, const unsigned long long key);*

Downloads a file to the validator. The first parameter is the filename to download, the second is the port name (eg /dev/ttyUSB0) and the third is the ssp address to download too. See DownloadDataToTarget for further information.

7.5 DOWNLOADDATATOTARGET

int DownloadDataToTarget(const unsigned char data, const unsigned long dlength, const char * cPort, const unsigned char sspAddress, const unsigned long long key);*

Downloads data to the target. First parameter is an array containing the data to download, second is the length of the data, third is the port to use (eg /dev/ttyUSB0) and fourth is the ssp address.

The fourth parameter is the encryption key to use. If this is 0 no encryption will be attempted, any other value will fail if encryption cannot be established with that key.

The function will return a value below DOWNLOAD_COMPLETE(0x100000) on success, indicating the number of blocks to download. The download will then continue in a seperate thread, and can be monitored with the GetDownloadStatus function. Only one download is permitted at a time - the function will return with a failure code if multiple downloads are attempted.

7.6 GETDOWNLOADSTATUS

unsigned long GetDownloadStatus(void);

Returns the status of the download. If the download is in progress this will be a block number (less than DOWNLOAD_COMPLETE(0x100000)), if the download has finished it will be a status code. A list of the status codes are in the ssp_defines.h header file. DOWNLOAD_COMPLETE(0x100000) means the download was sucessful, any other status code indicates an error.

7.7 NEGOTIATESSPENCRYPTION

*int NegotiateSSPEncryption(SSP_PORT port, const char ssp_address, SSP_FULL_KEY * key);*

Negotiates the variable key part of the SSP Encryption. First Parameter is the port to use, Second Parameter is the ssp_address and third parameter is the structure to store the key in.

Returns 1 on success, 0 on failure.

This function only sets the variable (.EncryptKey) part of the SSP_FULL_KEY structure. The user needs to set the Fixed part (.FixedKey) manually before SSP Encryption will function.

7.8 SSP FUNCTIONS

The library also implements functions for the base SSP Commands. These functions will all return values from the enum SSP_RESPONSE_ENUM, with SSP_RESPONSE_OK indicating success and other values indicating errors. These functions all require an argument of type SSP_COMMAND_SETUP, which is used to set the parameters of the communcation.

```
typedef struct {
        SSP_FULL_KEY Key;
        unsigned long Timeout;
        unsigned char SSPAddress;
        unsigned char RetryLevel;
        unsigned char EncryptionStatus;
        SSP_PORT port;
}SSP_COMMAND_SETUP;
```

The SSP_COMMAND_SETUP structure must have the following set before it is used
.EncryptionStatus set to NO_ENCRYPTION (if encryption is needed, call the ssp_setup_encryption function before sending the encrypted commands)
.SSPAddress needs to be set to the ssp address
.port set to a valid SSP_PORT handle
.RetryLevel set to the number of times to retry a command
.Timeout to the required timeout in milliseconds.

7.8.1 SSP_SETUP_ENCRYPTION

*SSP_RESPONSE_ENUM ssp_setup_encryption(SSP_COMMAND_SETUP * setup,const unsigned long long fixedkey);*

The fixedkey is the fixed key in the unit - default key in a unit is 0x123456701234567. If successful this will change the SSP_COMMAND_SETUP function so to enable encryption.

7.8.2 SSP_RESET

SSP_RESPONSE_ENUM ssp_reset(SSP_COMMAND_SETUP setup);

This will reset the unit to power on state if successful.

7.8.3 SSP_HOST_PROTOCOL

SSP_RESPONSE_ENUM ssp_host_protocol(SSP_COMMAND_SETUP setup,const unsigned char host_protocol);

Host Protocol is the level of protocol the host is attempting to use. If this version is supported, SSP_RESPONSE_OK will be returned.

7.8.4 SSP_POLL

*SSP_RESPONSE_ENUM ssp_poll(SSP_COMMAND_SETUP setup,SSP_POLL_DATA * poll_response);*

```
typedef struct {
   unsigned char event;
   unsigned long data;
} SSP_POLL_EVENT;

typedef struct {
   SSP_POLL_EVENT events[20];
   unsigned char event_count;
} SSP_POLL_DATA;
```

This will send a poll command and populate the structure that was passed in. If successful, the structure will contain the number of new events and data for each event.

7.8.5 SSP_GET_SERIAL

*SSP_RESPONSE_ENUM ssp_get_serial(SSP_COMMAND_SETUP setup,unsigned long * serial);*

If successful, serial will be set to the value of the unit serial number.

7.8.6 SSP_SYNC

SSP_RESPONSE_ENUM ssp_sync(SSP_COMMAND_SETUP setup);

Sends the syncronisation command. Should be sent whenever the port is opened or the unit is reset. This is a good way to check the unit is present and responding.

7.8.7 SSP_DISABLE

SSP_RESPONSE_ENUM ssp_disable(SSP_COMMAND_SETUP setup);

Disables the unit so no new notes will be accepted.

7.8.8 SSP_ENABLE

SSP_RESPONSE_ENUM ssp_enable(SSP_COMMAND_SETUP setup);

Enables the unit so any notes that are uninhibited (see ssp_set_inhibits) will be accepted.

7.8.9 SSP_SET_INHIBITS

SSP_RESPONSE_ENUM ssp_set_inhibits(SSP_COMMAND_SETUP setup,const unsigned char lowchannels, const unsigned char highchannels);

Sets the inhibits based on the bitmasks. lowchannels is for channels 1-8, high channels is for 9-17. A value of 1 indicates that channel will be accepted, 0 disables that channel.

7.8.10 SSP_DISPLAY_ON

SSP_RESPONSE_ENUM ssp_display_on(SSP_COMMAND_SETUP setup);

Turns the front bezel on.

7.8.11 SSP_DISPLAY_OFF

SSP_RESPONSE_ENUM ssp_display_off(SSP_COMMAND_SETUP setup);

Turns the front bezel off.

7.8.12 SSP_HOLD_NOTE

SSP_RESPONSE_ENUM ssp_hold_note(SSP_COMMAND_SETUP setup);

If a unit has a note in escrow, this command can be used to hold it in escrow. If the unit doesn't recieve a hold command for 5 seconds, the note will be rejected.

7.8.13 SSP_UNIT_DATA

*SSP_RESPONSE_ENUM ssp_unit_data(SSP_COMMAND_SETUP setup, SSP_UNIT_DATA * sud);*

```
typedef struct{
   unsigned char UnitType;
   char FirmwareVersion[5];
   char CountryCode[4];
   unsigned long ValueMultiplier;
   unsigned char ProtocolVersion;
}SSP_UNIT_DATA;
```

Gets basic information about the unit and populates the structure.

7.8.14 SSP_ENABLE_HIGHER_PROTOCOL_EVENTS

SSP_RESPONSE_ENUM ssp_enable_higher_protocol_events(SSP_COMMAND_SETUP setup);

Enables the higher protocol events in the validator. See SSP Specification for more details

7.8.15 SSP_CHANNEL_VALUE_DATA

*SSP_RESPONSE_ENUM ssp_channel_value_data(SSP_COMMAND_SETUP setup, SSP_CHANNEL_DATA * scd);*

```
typedef struct{
   unsigned char NumberOfChannels;
   unsigned char ChannelData[16];
}SSP_CHANNEL_DATA;
```

Gets the values per channel from the unit. See SSP Specification for more details

7.8.16 SSP_CHANNEL_SECURITY_DATA

*SSP_RESPONSE_ENUM ssp_channel_security_data(SSP_COMMAND_SETUP setup, SSP_CHANNEL_DATA * scd);*

Gets channel security information from the unit. See SSP Specification for more details

7.8.17 SSP_LAST_REJECT

*SSP_RESPONSE_ENUM ssp_last_reject(SSP_COMMAND_SETUP setup, unsigned char * last_reject_reason);*

last_reject_reason will be set to the reason the last note was rejected. See SSP Specification for more details.

7.8.18 SSP_SETUP_REQUEST

```
SSP_RESPONSE_ENUM ssp_setup_request(SSP_COMMAND_SETUP setup,
SSP_SETUP_REQUEST_DATA * setup_request_data);

typedef struct {
   unsigned char UnitType;
   char FirmwareVersion[5];
   char CountryCode[4];
   unsigned long ValueMultiplier;
   SSP_CHANNEL_DATA ChannelValues;
   SSP_CHANNEL_DATA ChannelSecurity;
   unsigned long RealValueMultiplier;
   unsigned char ProtocolVersion;
}SSP_SETUP_REQUEST_DATA;
```

Gets all the available setup information from the connected unit.