



RELEASE NOTES

ZigBee Home Automation/Light Link SDK

JN-SW-4168

Build 1620

NXP Semiconductors

For the contact details of your local NXP office or distributor, refer to:

www.nxp.com

CONTENTS

CONTENTS	2
RELEASE SUMMARY (v1620)	5
1. SDK Software Components	5
2. Supported Products	6
3. Software Installation	6
4. Release Details	7
4.1 ZCL and Profile	7
4.1.1 New ZCL/Profile Features and Enhancements	7
4.1.2 ZCL/Profile Bug Fixes	7
4.1.3 ZCL/Profile Known Issues	8
4.2 ZigBee Green Power	8
4.3 ZigBee PRO Stack	9
4.3.1 New Stack Features and Enhancements	9
4.3.2 Stack Bug Fixes	9
4.3.3 Stack Known Issues	10
4.4 JenOS	10
4.4.1 Bug Fixes	10
4.4.2 JenOS Known Issues	10
4.5 JN516x Integrated Peripherals API	10
4.5.1 Bug Fixes	10
4.6 Production Test Libraries	10
4.6.1 Bug Fixes	10
4.7 Application Porting Notes	11
4.7.1 OS Configuration diagram update	11
4.7.2 Stack configurations	12
5. Release Details	15
5.1 ZCL and Profile	15
5.1.1 New ZCL/Profile Features and Enhancements	15
5.1.2 ZCL/Profile Bug Fixes	15
5.1.3 ZCL/Profile Known Issues	16
5.2 ZigBee Green Power	16
5.3 ZigBee PRO Stack	18
5.3.1 New Stack Features and Enhancements	18
5.3.2 Stack Bug Fixes	18
5.3.3 Stack Known Issues	20
5.4 JenOS	20
5.4.1 Bug Fixes	20
5.4.2 JenOS Known Issues	20
5.5 JN516x Integrated Peripherals API	21
5.5.1 Bug Fixes	21
5.6 Production Test Libraries	21
5.6.1 Bug Fixes	21
5.7 Application Porting Notes	21
5.7.1 Makefile Updates	21
5.8 Bootloader Version	22
RELEASE HISTORY (v1364)	23

6. Release Details	23
6.1 ZCL and Profile	23
6.1.1 New ZCL/Profile Features and Enhancements	23
6.1.2 ZCL/Profile Bug Fixes	23
6.2 ZigBee Green Power	24
6.3 ZigBee PRO Stack	24
6.3.1 New Stack Features and Enhancements	24
6.3.2 Stack Bug Fixes	25
6.3.3 Stack Known Issues	26
6.4 JN516x Integrated Peripherals API	26
6.5 Production Test Libraries	26
6.6 Application Porting Notes	27
6.6.1 Within 'BeyondStudio for NXP'	27
6.6.2 Within vAppMain	27
6.6.3 Setting Minimum Heap and Stack Sizes in Application	28
6.6.4 Modifying Application Makefile	28
6.6.5 PDM Additional Notes	31
6.6.6 Registering an Error Handler (JN516x EEPROM Only)	31
6.6.7 OS Error Checking	33
6.6.8 ZigBee PRO Extended Error Status	34
6.6.9 Beacon Filtering	34
6.6.10 Removal of PHY Interrupt from OS Diagram	35
6.6.11 Removal of Management Bind Server from Application	35

RELEASE HISTORY (v1279) 36

7. Release Details	36
7.1 ZCL and Profile	36
7.1.1 ZCL Changes	36
7.1.2 Green Power Changes	36
7.2 ZigBee PRO Stack	36
7.2.1 New Features and Enhancements	36
7.2.2 Bug Fixes	36
7.2.3 Known Issues	37
7.3 JN516x Integrated Peripherals API	37
7.4 Production Test Libraries	37
7.5 Application Porting Notes	38
7.5.1 Within 'BeyondStudio for NXP'	38
7.5.2 Within vAppMain	38
7.5.3 Setting Minimum Heap and Stack Sizes in Application	39
7.5.4 Modifying Application Makefile	39
7.5.5 PDM Additional Notes	42
7.5.6 Registering an Error Handler (JN516x EEPROM Only)	42
7.5.7 OS Error Checking	44
7.5.8 ZigBee PRO Extended Error Status	45
7.5.9 Beacon Filtering	45
7.5.10 Removal of PHY Interrupt from OS Diagram	46
7.5.11 Removal of Management Bind Server from Application	46
7.6 Bootloader Version	46

RELEASE HISTORY (v1270) 47

8. Release Details	47
8.1 ZCL and Profile	47
8.1.1 ZCL Changes	47

8.1.2 Green Power Changes	50
8.2 ZigBee PRO Stack	51
8.2.1 New Features and Enhancements	51
8.2.2 Known Issues	52
8.3 Production Test Libraries	52
8.4 Application Porting Notes	52
8.4.1 Within 'BeyondStudio for NXP'	53
8.4.2 Within vAppMain	53
8.4.3 Setting Minimum Heap and Stack Sizes in Application	54
8.4.4 Modifying Application Makefile	54
8.4.5 PDM Additional Notes	57
8.4.6 Registering an Error Handler (JN516x EEPROM Only)	57
8.4.7 OS Error Checking	59
8.4.8 ZigBee PRO Extended Error Status	60
8.4.9 Beacon Filtering	60
8.4.10 Removal of PHY Interrupt from OS Diagram	61
8.4.11 Removal of Management Bind Server from Application	61
8.5 Bootloader Version	61

RELEASE SUMMARY (v1620)

1. SDK Software Components

The JN516x ZigBee HA/ZLL Software Developer's Kit (JN-SW-4168) comprises the ZigBee PRO libraries and ZigBee Cluster Library (ZCL) together with the Home Automation (HA) and ZigBee Light Link (ZLL) application profiles, providing support for ZigBee PRO application development for HA and ZLL on an NXP JN516x wireless microcontroller (JN5169/JN5168 for HA, JN5169/JN5168/JN5164 for ZLL). This SDK must be installed on top of the 'BeyondStudio for NXP' toolchain (JN-SW-4141), which is available via the [Wireless Connectivity](#) area of the NXP web site.

This SDK release provides:

- ZCL and HA/ZLL profile source code and APIs
- ZigBee PRO stack and APIs for JN5169, JN5168 and JN5164
- JenOS (Jennic Operating System) modules
- Chip support libraries for JN516x
- 802.15.4 Stack API
- JN516x Integrated Peripherals API
- Makefile options for JN516x (JENNIC_CHIP_FAMILY=JN516x, JENNIC_CHIP=JN5169, JENNIC_CHIP=JN5168, JENNIC_CHIP=JN5164)
- PDM (Persistent Data Manager) for JN516x EEPROM
- Three IEEE802.15.4 MAC libraries:
 - Full MAC with 2006 security
 - MicroMAC – basic initialisation of MAC/PHY
 - MiniMAC – full functionality of the MAC with limited 2006 security customised for ZigBee
- ZPS and JenOS Configuration Editor plug-ins for Eclipse
- LCD driver updated for DR1174 Carrier Board (changes to **LcdDriver.h** and **libBoardLib_JN516x.a**)
- Production Test library for each chip: **libJPT_JN5169.a**, **libJPT_JN5168.a** and **libJPT_JN5164.a**
- Sniffer binaries for JN5168 and JN5169

2. Supported Products

The SDK supports the following NXP products:

Product Type	Part Number	Version	Supported Chips	Supported Protocols
JN516x	-	-	JN5169 JN5168 JN5164 JN5161	802.15.4, ZigBee PRO 802.15.4, ZigBee PRO 802.15.4, ZigBee PRO 802.15.4 ZCL r04 HA1.2 + Errata ZLL1.0 + Errata
SDK Toolchain	JN-SW-4141	v1308	JN51xx	-

3. Software Installation

If you already have the earlier version of SDK on your machine, before installing this JN516x SDK you should first back up your Applications development directory and any user-modified files within the SDK directory.

This SDK (JN-SW-4168) must be installed on top of the 'BeyondStudio for NXP' toolchain (JN-SW-4141), available via the [Wireless Connectivity](#) area of the NXP web site.

Therefore, before installing JN-SW-4168, you must install the toolchain from:

JN-SW-4141 Beyond Studio for NXP v1308.exe

You can then install the JN516x ZigBee HA/ZLL SDK from:

JN-SW-4168 ZigBee-HA-LL v1620.exe

For full installation instructions, refer to the *BeyondStudio for NXP Installation and User Guide (JN-UG-3098)*, available from the [Wireless Connectivity](#) area of the NXP web site. This manual also describes how to install the ZPS and JenOS Configuration Editor plug-ins for Eclipse (BeyondStudio for NXP), which you must install after the SDK.

Note that the BeyondStudio for NXP toolchain includes a built-in Flash programmer that can be used to program JN516x internal Flash memory from BeyondStudio.

Alternatively, the JN51xx Production Flash Programmer (JN-SW-4107) command-line tool can be used to program JN516x internal or external Flash memory. This tool is available from the [Wireless Connectivity](#) area of the NXP web site and is described in the *JN51xx Production Flash Programmer User Guide (JN-UG-3099)*.

4. Release Details

4.1 ZCL and Profile

4.1.1 New ZCL/Profile Features and Enhancements

This release includes the following new ZCL/Profile/Application features:

- [Ref. 7634] New Fan Control Cluster
- [Ref. 7635 and 7653] Fan Control Cluster server and Diagnostic cluster server have been added as optional clusters to the thermostat device

4.1.2 ZCL/Profile Bug Fixes

In this release, the following bugs have been fixed in the ZCL/Profile:

Reference	Description
lpsw8326	Issue: The OTA state machine causes an exception if the end request is sent with an upgrade time of 0xffffffff and subsequently the device is reset. Solution: Fixed.
lpsw7732	Issue: OTA ImageStamp attribute does not have a defined location from where it can be read. Solution: OTA ImageStamp is now populated by the JET. This location is exported out to the ZCL
lpsw7741	Issue: If the OTA_CLD_ATTR_REQUEST_DELAY is defined the OTA cluster stops requesting image Solution: When OTA_CLD_ATTR_REQUEST_DELAY is defined the OTA cluster doesn't restart the internal millisecond timer. This means further blocks are requested after the initial expiry of the timer. This timer is now restarted to make sure the OTA process continues.
lpsw7758	Issue: Scenes header file is present in the ZCL source with incorrect case. Solution: The header file should be present as "Scenes.h" and not "scenes.h".
lpsw7760	Issue: CLD_IDENTIFY_SUPPORT_ZLL_ENHANCED_COMMANDS definition is used in the OnOff Cluster instead of CLD_ONOFF_SUPPORT_ZLL_ENHANCED_COMMANDS Solution: Fixed.
lpsw7787	Issue: In a corner case Involving both Recall Scene and Level Control, it is possible that the global scene may be set to "OFF" when the expected behaviour would be for it to be set to "ON". Solution: Fixed
lpsw7849	Issue: eZCL_ReportAllAttributes does not send out reports for attributes which have been enabled by the application using the SetReportableFlag. Solution: Fixed
lpsw7866	Issue: Attribute report for multiple attributes in different clusters doesn't work. Solution: Fixed.
lpsw7868	Issue: IAS Zone Sensitivity Attributes code doesn't have user configurable , number of zones and current zone level settings.

	Solution: These are now provided through CLD_IASZONE_NUMBER_OF_ZONE_SENSITIVITY_LEVELS and CLD_IASZONE_CURRENT_ZONE_SENSITIVITY_LEVEL.
lpsw7878	Issue: Poll Control Cluster Internal Variable Initialization to track the check in intervals is incorrect. Solution: Fixed by initializing it properly.
lpsw7901	Issue: Instantaneous demand attribute is not reportable. Solution: Fixed, it has now been made reportable.
lpsw7967	Issue: OTA_MIN_TIMER_MS_RESOLUTION is not set to 0. Solution: Fixed
lpsw7876	Issue: There is not API in the Poll Control cluster to update the elapsed time when an end device comes out of sleep. Solution: Fixed, a new API is added.
lpsw7977	Issue: There is no compile time option to make the ZCL use a non-blocking bound transmission call to the stack. Solution: Fixed.
lpsw8061	Issue: The OTA cluster does not validate the Image time stamp prior to activating the image. Solution: Fixed.
lpsw8068	Issue: The OTA cluster does not Validate embedded OTA IDs with advertised OTA IDs Solution: Fixed.
lpsw8212	Issue: Zone Enrol Request handler does not initialise default response data Solution: Fixed.
lpsw8215	Issue: Scene Valid attribute is not reset if scene attributes are changed Solution: Fixed.
lpsw8216	Issue: The store scene command processing does not set the bActive flag to True for the stored scene, nor does it search the scene table to see if a scene is already active and set that scenes flag to False Solution: Fixed.

4.1.3 ZCL/Profile Known Issues

There are no known ZCL/profile issues in this release.

4.2 ZigBee Green Power

None.

4.3 ZigBee PRO Stack

ZigBee PRO libraries are included for the JN5169, JN5168 and JN5164 devices.

4.3.1 New Stack Features and Enhancements

This release includes the following new stack features:

- [Ref. 6504] Data frames are buffered and autonomously resent after successful completion of route discovery.
- [Ref. 6787] A new debug extended error code is added to indicate lack of BTT resources.
- [Ref. 8031] The stack allows the application to trigger to sleep between waiting for APS ack responses.

4.3.2 Stack Bug Fixes

In this release, the following bugs have been fixed in the ZigBee PRO stack:

Reference	Description
lpsw8359	Issue: Stack prefers table routing ahead of a newer source route and the final destination packet when sent with source route is corrupted. Solution: Fixed.
lpsw7664	Issue: Remove device API causes the parent also to be removed when the child is removed. Solution: Fixed.
lpsw7818	Issue: End device loses its old parent's address map when it joins a new parent. Solution: Fixed. The End device maintains the old parent's address map if a binding for the old parent is present.
lpsw7820	Issue: If the first child of the router device is an end device and it gets removed or aged out then the management LQI server stops sending valid neighbour table size. Management LQI request returns wrong neighbour table size Solution: Fixed.
lpsw7824	Issue: Management leave request sends leave with remove children always set Solution: Fixed.
lpsw7919	Issue: Receiver-On-When Idle End device responds to Beacon Request. Solution: Fixed.
lpsw7953	Issue: Routers are removed from the local neighbour tables on loss of 3 link status. There is no configuration to extend the aging mechanism. Solution: Fixed. It is now possible to allow the router age limit to be up-to 255 link status.
lpsw7956	Issue: The stack is slow in responding to high data volume over the air. Solution: Fixed. A new queue to free resources quicker is now added.
lpsw7954	Issue: Return path to a received packet requires an additional route discovery Solution: Fixed. The stack uses the same path to send the response as it received the request.

lpsw8168	Issue: End device route not removed after re-join. Solution: Fixed.
lpsw8228	Issue: The ZDO server is blocked until an APS ack is received for the response sent. This makes the server slow in responding. Solution: Fixed.

4.3.3 Stack Known Issues

There are no known stack issues in this release.

4.4 JenOS

4.4.1 Bug Fixes

None.

4.4.2 JenOS Known Issues

In this release, JenOS has the following known issues:

Reference	Description
lpsw7648	Issue: On a 250+ nodes network, an occasional exception is seen. This normally corrupts the stack buffers. This has only been observed when running OTA upgrade. It is believed to be timing related. Workaround: Software reset on exception allows the node to continue working normally after recovering persisted network data.

4.5 JN516x Integrated Peripherals API

4.5.1 Bug Fixes

None.

4.6 Production Test Libraries

4.6.1 Bug Fixes

None.

4.7 Application Porting Notes

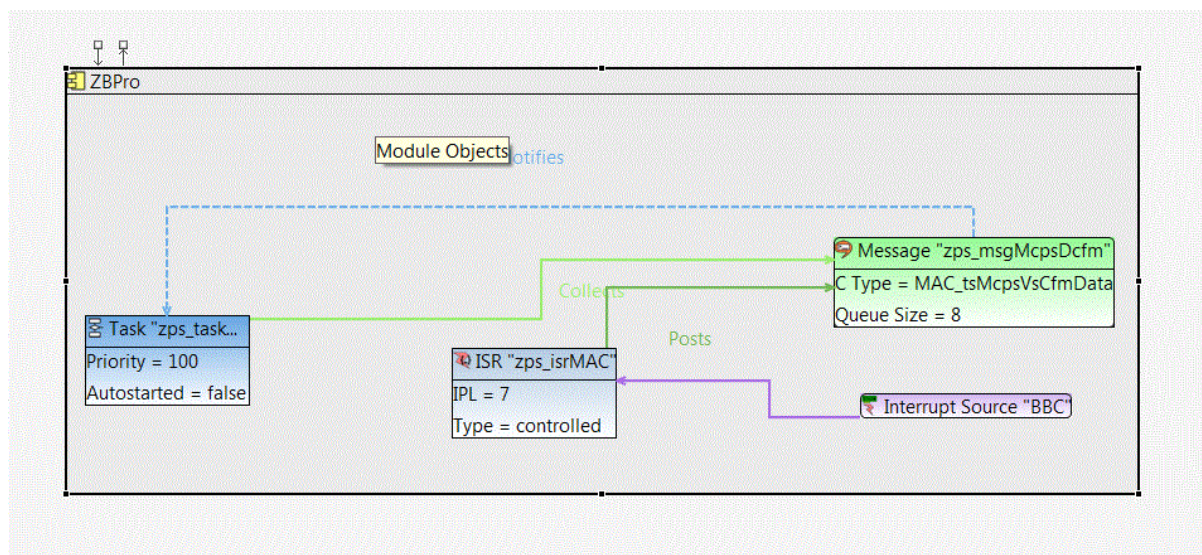
Before porting an existing application to this JN-SW-4168 SDK, it is recommended that you go through the *ZigBee PRO Stack User Guide (JN-UG-3101)* to ensure adherence to the recommended settings in the ZPS configuration.

The libraries supplied in the JN-SW-4168 SDK have been built against the new JN-SW-4141 toolchain. The migration guidelines for the new toolchain should be followed before using these porting guidelines. Migration to the new toolchain is described in the Application Note *BeyondStudio Migration Guidelines (JN-AN-1202)*.

4.7.1 OS Configuration diagram update

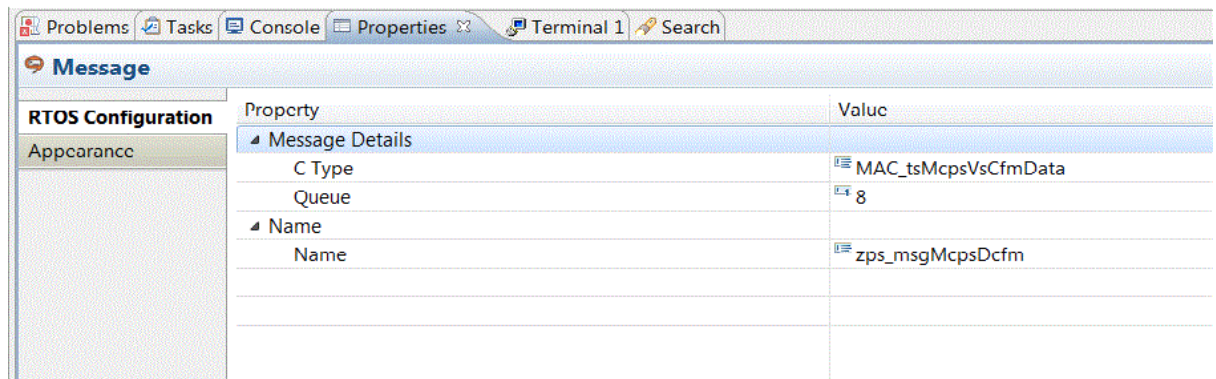
The Zigbee PRO stack requires an additional queue to increase its responsiveness to the high amount of traffic over the air. The queue is used to manage the asynchronous deferred confirmation of transmitted data requests submitted to the MAC/PHY interface. This new queue needs to be configured in the OS configuration diagram present in the application source code.

Figure 1



A new message queue should be added in the ZBPro stack scope box as shown in Figure 1. The properties of the queue should be setup as shown in Figure 2.

Figure 2:



The relationships of the message queue “zps_msgMcpsDcfm” , to the ISR “zps_isrMAC” and the task “zps_task” is shown in the Figure 1.

The ISR “zps_isrMAC” posts the deferred confirm message to the queue “zps_msgMcpsDcfm”, The queue “zps_msgMcpsDcfm” then “Notifies” the zps_task, The zps_task “Collects” the messages from the message queue “zps_msgMcpsDcfm”.

It is recommended that the queue size should be at least 8. It is further recommended that the queue which handles the “Mcps” data indication should be reduced since it no longer handles deferred confirms. For example, if the queue for the “Mcps” data request was of the size 24 it can now be reduced to a size of 16.

4.7.2 Stack configurations

In a large network, it is possible to miss a substantial number link status and this may result in devices ending up removed from the neighbour table. This results in routes being purged and a general churn of route discovery. This can be minimised by increasing the number missed link status before a router is removed from the neighbour table. Due to CCA and generally large amount of traffic over the air in a large network the chances of the far end missing transmissions is higher.

To counter these the stack now allows higher thresholds to be set for the network parameters which influence stability of routes. The following values are suggested for a general LNT setup:

```
ZPS_tsNwkNib * thisNib = ZPS_psNwkNibGetHandle(thisNet);
thisNib->u8RouterAgeLimit = 30;
thisNib->u8VsTxFailThreshold = 10;
```

To support an increase of number of missed link status to greater than the 3-bit value it previously was, the age argument is now an 8-bit value.

The age argument can be found in the neighbour table. The previous location of the age argument was as part of the bitfield parameter of the neighbour table structure.

The previous definition of the structure was:

```
typedef struct
{
    zps_tsNwkSlistNode sNode; /**< Single linked list node */
    uint16 u16Lookup;         /**< Extended address */
    uint16 u16NwkAddr;        /**< Network address */
    uint8  u8TxFailed;        /**< Transmit failed count */
    uint8  u8LinkQuality;     /**< Link Quality indication */
    uint8  u8ZedTimeoutindex; /**< index into the timeout const table */
    /*
     * Bitfields are used for syntactic neatness and space saving. May need to
     assess whether
     * these are suitable for embedded environment
     */
    union
    {
        struct
        {
            unsigned u1Used:1;          /* Overlays: Alternate PAN coordinator */
            unsigned u1DeviceType:1;
            unsigned u1PowerSource:1;
            unsigned u1RxOnWhenIdle:1;
            unsigned u2Relationship:2;  /* Overlays: Reserved */
            unsigned u1SecurityMode:1;
            unsigned u1Authenticated:1; /* Overlays: Allocate address */
            unsigned u1LinkStatusDone:1; /**< Link status has been processed for
this device */
            unsigned u3OutgoingCost:3;  /**< Outgoing cost for sym link = true
*/
            unsigned u3Age:3;
            unsigned u1ExpectAnnc:1;    /**** Set for newly joined children,
cleared on hearing their annce */
        } bfBitfields;
        uint8 au8Field[2];
    } uAncAttrs;
} ZPS_tsNwkActvNtEntry;
```

This is now changed to:

```
typedef struct
{
    zps_tsNwkSlistNode sNode; /**< Single linked list node */
    uint16 u16Lookup;         /**< Extended address */
    uint16 u16NwkAddr;        /**< Network address */
    uint8  u8TxFailed;        /**< Transmit failed count */
    uint8  u8LinkQuality;     /**< Link Quality indication */
    uint8  u8Age;             /**< Router age (in link status periods) */
    uint8  u8ZedTimeoutindex; /**< index into the timeout const table */
    /*
     * Bitfields are used for syntactic neatness and space saving. May need to
     assess whether
     * these are suitable for embedded environment
     */
    union
    {

```

```

struct
{
    unsigned u1Used:1;           /* Overlays: Alternate PAN coordinator */
    unsigned u1DeviceType:1;
    unsigned u1PowerSource:1;
    unsigned u1RxOnWhenIdle:1;
    unsigned u2Relationship:2;   /* Overlays: Reserved */
    unsigned u1SecurityMode:1;
    unsigned u1Authenticated:1; /* Overlays: Allocate address */
    unsigned u1LinkStatusDone:1; /**< Link status has been processed for
this device */
    unsigned u3OutgoingCost:3;   /**< Outgoing cost for sym link = true
*/
    unsigned u3Reserve:3;
    unsigned u1ExpectAnnc:1;     /**< Set for newly joined children,
cleared on hearing their annce */
    } bfBitFields;
    uint8 au8Field[2];
} uAncAttrs;
} ZPS_tsNwkActvNtEntry;

```

All references to `sNtEntry.uAncAttrs.bfBitFields.u3Age` should now be changed to `sNtEntry.u8Age`.

RELEASE HISTORY (v1461)

5. Release Details

5.1 ZCL and Profile

5.1.1 New ZCL/Profile Features and Enhancements

This release includes the following new ZCL/Profile/Application features:

- [Ref. 6957] Simple Metering cluster added to the Dimmable Light device type as an optional cluster for Icontrol certification.
- [Ref. 7072] Configurable OTA_BLOCK_REQUEST_DELAY_MAX_VALUE
- [Ref. 7360] Co-processor support for OTA
- [Ref. 7409] Faster compile and build times for ZCL/Profile/Application
- [Ref. 7414] OTA cluster (client/server) added into Simple Sensor device

5.1.2 ZCL/Profile Bug Fixes

In this release, the following bugs have been fixed in the ZCL/Profile:

Reference	Description
lpsw6948	Issue: Error in initializing AppProfileVersion attribute in the ZCL. Solution: Updated Basic.c for proper initialization of AppProfileVersion attribute.
lpsw6949	Issue: Basic cluster attribute ApplicationProfileVersion has an ID of 0x0009 which is incorrect Solution: In Basic.h, <ul style="list-style-type: none">• E_CLD_BAS_ATTR_ID_APPLICATION_PROFILE_VERSION will have attribute id (enum) of 0x0008, according to the HA121(13-0553-40) errata doc• E_CLD_BAS_ATTR_ID_APPLICATION_PROFILE_TYPE gets value of 0x0009 (this attribute's reference is not available in the documents HAv1.2.1 (05-3520-29), ZCL (075123r03ZB) or HA errata doc (13-0553-40)). It is present in the implementation for backward compatibility.
lpsw7064	Issue: OTA ImageType attribute is not changing its value according to the specification. Solution: OTA ImageType attribute is updated by the ZCL according to the specification. Implementation-specific Note: If ImageUpgradeStatus is "Normal" then the ImageType attribute will be 0xFFFF, else it will be the type of the file in download process.
lpsw7311	Issue: ConfigureReportingResponse for an unreportable/unsupportable attribute is C1 and not UNREPORTABLE_ATTRIBUTE/UNSUPPORTED_ATTRIBUTE. Solution: Check has been added in the vZCL_HandleConfigureReportingCommand() function to give UNREPORTABLE_ATTRIBUTE/UNSUPPORTED_ATTRIBUTE status in ConfigureReportingResponse for an unreportable/unsupportable attribute.
lpsw7370	Issue: When a ReadAttributeResponse command is received, the length and MaxLength of the string type attributes are incorrect when reported to the application via the callback event E_ZCL_CBET_READ_INDIVIDUAL_ATTRIBUTE_RESPONSE. Solution: Fixed.

	Implementation-specific Note: While exporting a string in the E_ZCL_CBET_READ_INDIVIDUAL_ATTRIBUTE_RESPONSE event, use tsZCL_CharacterString instead of tsZCL_String in uAttribData.
lpsw7374	Issue: DiscoveryAttributeResponse is dropped if the packet is larger than can be held in outgoing buffer. Solution: Sending only those attributes which fit in the outgoing buffer and set the 'discovery complete' flag to false.
lpsw7403	Issue: Attribute reporting does not work if attribute is of the string type. Solution: Fixed.
lpsw7413	Issue: The stack did not allow you to send multiple reports at the same time. Solution: The ZCL is using a bound 'no ack' transmission function that now allows you to send multiple reports. Updated ZCL for non-blocking bound transmissions. For non-blocking bound transmission, use E_ZCL_AM_BOUNDED_NON_BLOCKING or E_ZCL_AM_BOUNDED_NON_BLOCKING_NO_ACK.
lpsw7418	Issue: Control Bridge sends random status byte to the host due to attribute status not being initialized properly in the ZCL. Solution: Fixed by initializing it properly.
lpsw7495	Issue: In the Colour Control cluster, Hue is not updated when 'Move to Colour Temp' command is used. Solution: Fixed
lpsw7533	Issue: Even though an attribute value is unrepeatable/unsupported/incorrect in the ConfigureReporting command, when the ZCL generates the E_ZCL_CBET_REPORT_INDIVIDUAL_ATTRIBUTES_CONFIGURE callback event to application, the eZCL status value is success. This is causing confusion to the application. Solution: E_ZCL_CBET_REPORT_INDIVIDUAL_ATTRIBUTES_CONFIGURE callback event has the following status codes for the application: <ul style="list-style-type: none"> • If attribute search failed, callback event with eZCL_Status of E_ZCL_ERR_ATTRIBUTE_NOT_FOUND generated • If attribute type unsupported, callback event with eZCL_Status of E_ZCL_ERR_ATTRIBUTE_TYPE_UNSUPPORTED generated • If attribute type is not reportable, callback event with eZCL_Status of E_ZCL_ERR_ATTRIBUTE_NOT_REPORTABLE generated If reporting interval is not in range, callback event with eZCL_Status of E_ZCL_ERR_INVALID_VALUE generated

5.1.3 ZCL/Profile Known Issues

There are no known ZCL/profile issues in this release.

5.2 ZigBee Green Power

In this release, the following bugs have been fixed in the ZigBee Green Power:

Reference	Description
lpsw7639	Issue: Address conflict is not resolved when a node on the network has the same short address as the Green Power Device. Solution: Updated GreenPower.c to return the evaluated status of the conflict (bAliasMatched).

5.3 ZigBee PRO Stack

ZigBee PRO libraries are included for the JN5169, JN5168 and JN5164 devices.

5.3.1 New Stack Features and Enhancements

This release includes the following new stack features:

- [Ref. 7019] New callback for 'update device' indication when a device is leaving
- [Ref. 7261] Check that no circular routes are present when sending data over the air
- [Ref. 7305] API function to prevent parent routing devices from notifying the Trust Centre when Orphan notifications are used as a way of rejoining the network. This closes a security hole in which the Trust Centre would transport a key to the newly joined devices
- [Ref. 7310] User-defined LQI to Link Cost mappings
- [Ref. 7426] MAC allows two different values of minBE. One is used when the data request is received and the other for everything else. This is useful on a routing device, so that it can respond faster for a sleepy child.

5.3.2 Stack Bug Fixes

In this release, the following bugs have been fixed in the ZigBee PRO stack:

Reference	Description
lpsw5870	Issue: MiniMAC for JN5169 does not support all Tx power levels. Solution: The MAC now supports the whole range of power levels that the JN5169 can support.
lpsw6907	Issue: Management Leave server incorrectly sent a response of NOT SUPPORTED. Solution: Fixed.
lpsw6938	Issue: When issuing a request to set the User Descriptor on a remote node using the ZPS_eAplZdpUserDescSetRequest() function, the length field of the request is always set to 16 rather than the actual length of the descriptor set when the request is made. Solution: Fixed.
lpsw7005	Issue: Parent sends poll response with no frame pending but proceeds to send data anyway. Solution: Fixed.
lpsw7012	Issue: Extended error status 0x87 due to AMT portion of the IEEE address table is full. Solution: Fixed.
lpsw7016	Issue: Management leave request is not being sent up to the application. Solution: Fixed.
lpsw7017	Issue: The stack parses the rejoin/remove child bits incorrectly (which has been rectified by the ZigBee Alliance). Solution: Parsing and setting the Leave bytes is now done correctly.
lpsw7018	Issue: MAC address table mis-management causes Address Map table to fill prematurely. Solution: Fixed.

lpsw7037	<p>Issue: Rejoins happen more frequently with End Devices due to LQI filtering when LQI is poor.</p> <p>Solution: Removed End Device link cost filtering.</p>
lpsw7145	<p>Issue: Parameter CTRIM of radio settings for JN5169 has been changed to provide more flexibility in crystal choice. Need to re-build JN5169 SDKs with this new lib.</p> <p>Solution: Enhanced JN5169 radio settings to extend the range of supported 32MHz external crystals.</p>
lpsw7185	<p>Issue: MiniMAC not setting frame pending bit in outgoing frames.</p> <p>Solution: When sending a frame in response to a data request, the MAC now sets the header field in the frame to indicate if there are additional frames that are still pending for the requestor.</p>
lpsw7231	<p>Issue: Stack is not checking that the destination address and originator address for network status is not the same.</p> <p>Solution: Fixed.</p>
lpsw7300	<p>Issue: Before populating the route record table, stack code was not checking whether the table has the capacity to store all the hops.</p> <p>Solution: Fixed.</p>
lpsw7303	<p>Issue: Stack was not persisting the Neighbour table and IEEE Address table after a previously aged device becomes part of the network again.</p> <p>Solution: Fixed.</p>
lpsw7337	<p>Issue: Group address table is not persisted correctly if there are more than one endpoint.</p> <p>Solution: Fixed.</p>
lpsw7344	<p>Issue: When declaring an RxOnIdle End Device and after it has joined the network, whenever a beacon request is sent by a device, the RxOnIdle End Device will respond with a junk beacon packet.</p> <p>Solution: No beacon response sent from RxOnIdle End Device.</p>
lpsw7409	<p>Issue: Application Notes take a long time to compile.</p> <p>Solution: Speed of compile time improved (by 6 minutes for the Control Bridge).</p>
lpsw7412	<p>Issue: In the Bound transmission API function, there is a state machine that does not allow you to send multiple bound transmissions at once. This is because it uses the same code as Bound with Ack which needs a state machine.</p> <p>Solution: Added new function to send non-blocking bound transmission to allow multiple bound requests to be sent at the same time.</p>
lpsw7428	<p>Issue: A sleepy End Device can prematurely turn its receiver off during a data poll if it receives a broadcast packet while waiting for buffered data from its parent.</p> <p>Solution: Broadcasts are ignored on 'Rx On When Idle=False' devices when waiting for buffered data and the receiver is turned off only after the buffered data turnaround period has elapsed.</p>
lpsw7518	<p>Issue: Simple descriptor returning 0x81 (invalid device) rather than 0x80 (invalid request) return code.</p> <p>Solution: Fixed.</p>
lpsw7520	<p>Issue: A data indication with a security status of 'security fail' occurs when application's security failure checks are done. To be consistent with network security failure, no APS acks should be sent back for security failure.</p> <p>Solution: APS acks now suppressed when APS security fails (similar to NWK security fail).</p>

lpsw7542	Issue: The stack does not clear the source routing table properly when the next hop in the route is aged out. Solution: Fixed.
lpsw7545	Issue: On a rejoin request with invalid address, the stack does not issue the rejoin response with a valid address unless the allocate address bit is set. Solution: Fixed.
lpsw7547	Issue: Stack does not issue a new key on request key if after commissioning it has been reset. Solution: Fixed.
lpsw7611	Issue: The SDK does not include the library file libAppApi_JN5169.a for the IEEE802.15.4 Stack API (needed for direct IEEE802.15.4 application coding for the JN5169 chip). Solution: File now included.

5.3.3 Stack Known Issues

There are no known stack issues in this release.

5.4 JenOS

5.4.1 Bug Fixes

In this release, the following bugs have been fixed in JenOS:

Reference	Description
lpsw7584	Issue: When saving 320 bytes with multiple bytes changed, the PDM evaluated the changes properly but did not write them correctly into the EEPROM. The internal bitmap used to track individual segment changes was using the wrong byte mask to isolate the correct read bit, yielding spurious results and causing the wrong segments to be updated (if they were not in a continuous run in the bitmap). Solution: Fixed

5.4.2 JenOS Known Issues

In this release, JenOS has the following known issues:

Reference	Description
lpsw7648	Issue: On a 250+ nodes network, an occasional exception is seen. This normally corrupts the stack buffers. This has only been observed when running OTA upgrade. It is believed to be timing related. Workaround: Software reset on exception allows the node to continue working normally after recovering persisted network data.

5.5 JN516x Integrated Peripherals API

5.5.1 Bug Fixes

In this release, the following bugs have been fixed in the JN516x Integrated Peripherals API:

Reference	Description
lpsw7429	Issue: The calibration of the wake timer, when using the function <code>u32AHI_WakeTimerCalibrateEnhanced()</code> , could terminate prematurely if an interrupt fired during the calibration operation. This would cause the reported calibration value to be incorrect. Solution: Fixed

5.6 Production Test Libraries

5.6.1 Bug Fixes

In this release, the following bugs have been fixed in the Production Test Libraries:

Reference	Description
lpsw7145	Issue: Parameter CTRIM of radio settings for JN5169 has been changed to provide more flexibility in crystal choice. Need to re-build JN5169 SDKs with this new lib. Solution: Enhanced JN5169 radio settings to extend the range of supported 32MHz external crystals.

5.7 Application Porting Notes

Before porting an existing application to this JN-SW-4168 SDK, it is recommended that you go through the *ZigBee PRO Stack User Guide (JN-UG-3101)* to ensure adherence to the recommended settings in the ZPS configuration.

The libraries supplied in the JN-SW-4168 SDK have been built against the new JN-SW-4141 toolchain. The migration guidelines for the new toolchain should be followed before using these porting guidelines. Migration to the new toolchain is described in the Application Note *BeyondStudio Migration Guidelines (JN-AN-1202)*.

5.7.1 Makefile Updates

The CPU stack size and minimum heap size are no longer referenced from the linker command scripts. The presence of **app_stack_size.ld** files in the build path of the application will now result in an error. Any **app_stack_size.ld** files in the application build folders must be removed.

The CPU stack size and minimum heap size are referenced from two variables. If CPU stack and heap sizes are required that are larger than the defaults of 5K bytes and 2K bytes respectively, these should now be defined in the application makefiles.

The variables to be defined in the makefile are:

```
STACK_SIZE ?= 6000  
MINIMUM_HEAP_SIZE ?= 2000
```

5.8 Bootloader Version

The version of the bootloader in the JN5168 device must be either of the following:

- 0x00080003
- 0x00080006

The version of the bootloader in the JN5169 device must be the following:

- 0x000B0000

RELEASE HISTORY (v1364)

6. Release Details

6.1 ZCL and Profile

6.1.1 New ZCL/Profile Features and Enhancements

This release includes the following new ZCL features:

- [Ref. 6054] Support for a Mains Power Outlet device
- [Ref. 6745] New clusters: Analogue Input, Analogue Output, Multistate Input, Multistate Output and Binary Output

6.1.2 ZCL/Profile Bug Fixes

In this release, the following bugs have been fixed in the ZCL/Profile:

Reference	Description
6100	<p>To speed up the reporting mechanism, the user needs to provide E_ZCL_CBET_TIMER_MS (millisecond) ticks along with E_ZCL_CBET_TIMER (one second) ticks. This can speed up 'report on change' in terms of milliseconds. The following code needs to be called from the application:</p> <pre>sCallbackEvent.eEventType = E_ZCL_CBET_TIMER_MS; vZCL_EventHandler(&sCallbackEvent);</pre> <p>Along with the above changes, E_ZCL_CBET_TIMER ticks still need to be provided, as they are used by UTC time and inherently by the ZCL report manager to keep track of time.</p>
6305	<p>An issue has been fixed relating to the update of the Block Request Delay attribute when an OTA client receives a WAIT FOR DATA command from the server.</p>
6469	<p>A new check has been added in which the user will not be allowed to create a group table if the number of groups specified in CLD_GROUPS_MAX_NUMBER_OF_GROUPS (in the ZCL) is greater than the value configured in the ZPS config diagram or in the stack.</p>
6674	<p>An issue has been fixed whereby the range check for the Saturation and Colour Temperature attributes was missing, which caused these attributes to go out of their valid ranges when a Move to Colour command was received by a Colour Control cluster server.</p>
6849	<p>The ReportAllAttributes() was exhausting resources with multiple endpoints and reportable attributes. A new ZCL function eZCL_ReportAttribute() has been introduced to send out the report for an individual reportable attribute. The function is detailed below.</p> <p>Function Prototype:</p> <pre>PUBLIC teZCL_Status eZCL_ReportAttribute(tsZCL_Address *psDestinationAddress, uint16 u16ClusterID, uint16 u16AttributeID, uint8 u8SrcEndPoint, uint8 u8DestEndPoint,</pre>

	<p style="text-align: center;">PDUM_thAPdulInstance hAPdulInst)</p> <p>Returns: E_ZCL_SUCCESS E_ZCL_ERR_ATTRIBUTE_NOT_FOUND E_ZCL_ERR_ATTRIBUTE_NOT_REPORTABLE E_ZCL_ERR_ZBUFFER_FAIL E_ZCL_ERR_EP_RANGE E_ZCL_ERR_CLUSTER_NOT_FOUND</p>
6970	An issue has been fixed whereby a Get Group Membership command did not return the groups having group ID greater than or equal to 0x8000.
6971	The Remove All Groups command previously removed all scenes associated with groups but included the global scene. However, the global scene is not associated with any groups, so should not be removed. This issue has been fixed and the global scene is now retained.
6973	An issue has been fixed whereby the Scenes extension field set was corrupted if the Add or Enhanced Add Scene command contained a scene name of length greater than the maximum scene length supported by the Scenes cluster server.
6974	An issue has been fixed whereby the ZCL did not allow to the addition of a manufacturer-specific cluster having a cluster ID greater than or equal to 0xFC01.
6975	The ZCL specification indicates that additional bytes found appended to a ZCL frame must be ignored, as these may be added as part of an updated frame format. In order to support this, the E_ZCL_ACCEPT_MORE flag (which was used to check whether more data than expected had been received and, if so, send back a default response with malformed command) has been removed from the ZCL.
6976	Previously, the Add Scene and Enhanced Add Scene commands resulted in a default response of malformed command if there were more or fewer bytes than expected in the extension field. For forward compatibility, the reception of the Add Scene and Enhanced Add Scene command now allows for the possible future addition of other attributes to the trailing ends of the scene extension list given in the cluster specifications (by ignoring them). Similarly, it allows for one or more attributes to be omitted from the trailing ends of the scene extension list.

6.2 ZigBee Green Power

No changes in addition to those made in v1279, described in Section 7.1.2.

6.3 ZigBee PRO Stack

ZigBee PRO libraries are included for the JN5169, JN5168 and JN5164 devices.

6.3.1 New Stack Features and Enhancements

This release includes no new stack features in addition to those in v1279, described in Section 7.2.1.

6.3.2 Stack Bug Fixes

In this release, the following bugs have been fixed in the ZigBee PRO stack:

Reference	Description
4600	ZDO servers reported the storage capacity of a node when responding to ZDP requests like management bind, management LQI and management routing requests. This resulted in reading empty entries and requiring interrogation of the full capacity. These servers now report only the active entries. This makes the discovery process faster and more efficient.
6171	The management NWK update from the ZDO server did not report the actual energy detection value generated by the energy scan for channel 26, but reported it as 0. This issue has been resolved and 0 is no longer reported for this channel.
6499	Local broadcast of permit join and management update requests could previously result in the server not accepting any further such requests. This prevented the device from moving channels or opening a permit join window. This has now been fixed.
6601	The ZPS_vTCSetCallback() function scope has been made PUBLIC and exposed to users in the zps_apl_af.h public header file.
6616	Inactive and initialised nodes previously responded to beacon requests by sending a beacon response in which the reported PAN ID was 0. This could result in PAN ID conflicts. This issue has been fixed and these nodes no longer respond to beacon requests.
6629	The MAC table was previously holding on to stale entries of devices that had left the network, reducing the number of devices that could join the network. This issue has been fixed and devices that leave the network are removed from the MAC table.
6648	The route discovery complete event did not previously report the short address of the device for which a route discovery had been initiated. This issue has been fixed and the event now reports the short address. This enhances the event usability and improves the application interface.
6677	Certain files were missing for the JN5164 devices. Support for HA and ZLL applications has now been extended to JN5164.
6718	Previously during a route discovery, if a route reply took a routed path, this would cause the route reply frame to be rejected and the route discovery to fail. Route reply frames are now regenerated at every hop needed to reach the source of the route discovery.
6732	A routing device could previously initiate a partial link status when a device announce was received from a node which was not its immediate neighbour. This was done to make routing quicker but could have an undesired effect of causing additional route discoveries for itself from other nodes. This issue has been fixed such that no link status is sent in this case.
6741	Additional route record frames for child End Devices from the parent are now suppressed.
6760	Previously, when a device had been aged out, the address map table was not updated correctly, which could result on an address conflict being declared incorrectly. This has now been fixed.
6785	The override profile callback was not being called for bound transmissions Therefore, the feature to override the profile ID of a data frame has been extended to bound transmissions.
6829	As for issue 4600 (above).

6.3.3 Stack Known Issues

Reference	Description
6086	<p>Circular Routes:</p> <p>It is possible that a unicast packet will not reach its destination because the packet is lost - for example, it becomes caught in a circular route. The APS acknowledgement mechanism can be used on the source node to monitor sent packets - an event will be generated when an acknowledgement is received from the target node. To allow an acknowledgment to be received, the source node must remain awake for a time equal to the APS retry timeout period.</p> <p>On a battery-powered node, the use of APS acknowledgements and retries may not be desirable from a power-saving point of view.</p> <p>Resolution:</p> <p>If the response is not observed within a pre-defined time then the application should take one of the following actions, depending on whether the source node is an End Device or Router:</p> <p>If an End Device, the application should notify the parent node about the routing problem by sending it a unicast network status command, ZPS_vNwkSendNwkStatusCommand(), with the status as "No Route Available (0x00)"</p> <p>If a Router, the application should initiate an explicit route discovery to the destination node by calling the function ZPS_eApiZdoRouteRequest()</p> <p>The above functions are described in the <i>ZigBee PRO Stack User Guide (JN-UG-3101)</i>.</p>
6169	<p>Discovery Failure Due to Circular Route:</p> <p>When a discovery command such as Match Descriptor is broadcast, a matching node will unicast the response back to the originator. If the response gets caught in a circular route, the discovery will not be successful.</p>
6133	<p>In a dense network it has been observed that a route reply does not get generated due to a CCA failure resulting in a Tx failure of the route reply. This prevents a route discovery from completing.</p>
6134	<p>During route discovery on a LNT, there have been instances where the route reply packet gets retried out. This results in a failure to complete the route discovery. The MAC error status returned is 'no ACK'.</p>

6.4 JN516x Integrated Peripherals API

In this release, there are no changes in addition to those made in v1279, described in Section 7.3.

6.5 Production Test Libraries

In this release, there are no changes in addition to those made in v1279, described in Section 7.4.

6.6 Application Porting Notes

Before porting an existing application to this JN-SW-4168 SDK, it is recommended that you go through the *ZigBee PRO Stack User Guide (JN-UG-3101)* to ensure adherence to the recommended settings in the ZPS configuration.

The libraries supplied in the JN-SW-4168 SDK have been built against the new JN-SW-4141 toolchain. The migration guidelines for the new toolchain should be followed before using these porting guidelines. Migration to the new toolchain is described in the Application Note *BeyondStudio Migration Guidelines (JN-AN-1202)*.

First note that:

- The JN516x device does not support overlays, so any code relating to overlays must be removed.
- The files **App_timer_driver.c** and **App_timer_driver.h** must be removed from the application **source/include** folders.

6.6.1 Within 'BeyondStudio for NXP'

Run BeyondStudio for NXP and follow the instructions below:

1. Select the project root directory.
2. Select **Properties** (in the **File** menu or pop-up menu obtained by right-clicking).
3. In the **Properties** window, select **C++ Build**.
4. For each node (selected from the **Configuration** drop-down list), on the **Build Settings** tab ensure that the **Build command** field contains a line of the form:

```
make JENNIC_CHIP_FAMILY=JN516x JENNIC_CHIP=JN5168 OVERLAY_BUILD=0  
PDM_BUILD_TYPE=_EEPROM
```

The above example assumes that the JN5168 chip is used.

6.6.2 Within vAppMain

The following changes should be made to the **vAppMain** code.

At the start of **vAppMain**, add the following:

```
// Wait until FALSE i.e. on XTAL - otherwise UART data will be at  
wrong speed  
while (bAHI_GetClkSource() == TRUE);  
// Now we are running on the XTAL, optimise the Flash memory wait  
states  
vAHI_OptimiseWaitStates();
```

The **Get/Release Mutex** functions for the Flash mutex should be non-counting:

```
PUBLIC void vGetMutex(void)  
{  
    OS_eEnterCriticalSection(mutexFlash);  
}  
  
PUBLIC void vReleaseMutex(void)  
{  
    OS_eExitCriticalSection(mutexFlash);  
}
```

If there is a call to:

```
u32AppApiInit(NULL, NULL, NULL, NULL, NULL, NULL);
```

it should be removed or replaced with:

```
#ifndef JENNIC_MAC_MiniMacShim
    u32AppApiInit(NULL, NULL, NULL, NULL, NULL, NULL);
#endif
```

It is now possible to rejoin a network without performing a discovery. This is achieved by calling the function **ZPS_eAplZdoRejoinNetwork()** and passing a flag of FALSE. This requires a parent to be present in the Neighbour table.

If discovery is still required then the above function should now be called with a flag of TRUE - refer to the *ZigBee PRO Stack User Guide (JN-UG-3101)* for details.

The stack no longer holds a map of short and long addresses in the NIB. It holds a map of short addresses and an index into a global long address. To find the actual long address, the following function can be called:

ZPS_u64NwkNibGetMappedExtendedAddress()

Refer to the *ZigBee PRO Stack User Guide (JN-UG-3101)* for details of the above function.

The stack no longer holds the MAC address in individual network tables. This has been replaced with a look-up table. The extended address can be found using the above function.

The indexes in the Neighbour table should never be written to by the application.

6.6.3 Setting Minimum Heap and Stack Sizes in Application

The minimum heap size can be set to a required value by defining the following in the application linker script **APP_stack_size.ld** located in the application build directory:

```
_minimum_heap_size = <required value>;
```

And the stack size can be set by:

```
_stack_size = <required size>;
```

6.6.4 Modifying Application Makefile

In the application makefile, you need to specify the type of IEEE802.15.4 MAC to use.

For all ZigBee applications, the MiniMAC will be used. It is referenced by setting the following in the makefile:

```
JENNIC_MAC ?= MiniMacShim
JENNIC_STACK ?= ZLLHA
```

The ZigBee Green Power and ZigBee Light Link optional functionality can now be controlled through the ZPS Configuration Editor.

These functionalities are present in the installer as optional libraries. To incorporate them, the following lines should be added to the Makefile:

```
JENNIC_SDK ?= JN-SW-4168
```

```
        SDK_BASE_DIR    = $(abspath
../../../../../sdk/$(JENNIC_SDK) )
APP_BASE                = $(abspath ../../..)
```

Include the correct profile source in the application by defining the following flags, as required.

For an HA lighting application:

```
APP_CLUSTER_HA_LIGHTING_SRC ?= 1
```

For a ZLL lighting applications:

```
APP_CLUSTER_ZLL_SRC ?= 1
```

To include appropriate cluster code, enable one or more of the following macros, as required

```
# OTA Cluster
#APP_CLUSTERS_OTA_SRC ?=1

# Green Power clusters
#APP_CLUSTERS_GREENPOWER_SRC ?=1

# Lighting Clusters
#APP_CLUSTER_LIGHTING_SRC ?=1

# Measurement and sensing clusters
#APP_CLUSTERS_MEASUREMENT_AND_SENSING ?=1

# Energy at Home clusters
#APP_CLUSTERS_ENERGY_AT_HOME_SRC ?=1

# HVAC clusters
#APP_CLUSTERS_HVAC_SRC ?=1

# Smart Energy clusters
#APP_CLUSTERS_SMART_ENERGY_SRC ?=1

# IAS clusters
#APP_CLUSTERS_IAS_SRC ?=1
```

Replace the `OSCONFIG` , `PDUMCONFIG` and `ZPSCONFIG` lines in the makefile with the following:

```
$(OSCONFIG) -f $< -o $(APP_SRC_DIR) -v $(JENNIC_CHIP)
$(PDUMCONFIG) -z $(TARGET) -f $< -o $(APP_SRC_DIR)
$(ZPSCONFIG) -n $(TARGET) -t $(JENNIC_CHIP) -l
$(ZPS_NWK_LIB) -a $(ZPS_APL_LIB) -c
$(TOOL_COMMON_BASE_DIR)/$(TOOLCHAIN_PATH) -f $< -o
$(APP_SRC_DIR)

OPTIONAL_STACK_FEATURES = $(shell $(ZPSCONFIG) -n $(TARGET) -f
$(APP_SRC_DIR)/$(APP_ZPSCFG) -y )
```

All references to Cygwin paths/shell should be removed.

All references to `TEMPCHP` should be replaced with `JENNIC_CHIP`.

Utilities functions (e.g. NumToString) need to be built from source.

Add the utilities source directory (**C:\Jennic\Components\Utilities\Source**) to vpath:

- UTIL_SRC_DIR = \$(COMPONENTS_BASE_DIR)/Utilities/Source
- Add to vpath : \$(UTIL_SRC_DIR)

Add the utilities source files:

- APPSRC += NumToString.c
- APPSRC += appZpsBeaconHandler.c
- APPSRC += appZdpExtraction.c
- APPSRC += pdum_apdu.S

Add the following to the include path:

```
INCFLAGS += -I$(COMPONENTS_BASE_DIR)/Random/Include
INCFLAGS += -I$(COMPONENTS_BASE_DIR)/MAC/Include
INCFLAGS += -I$(COMPONENTS_BASE_DIR)/NXPLogo/Include
INCFLAGS += -I$(COMPONENTS_BASE_DIR)/Utilities
```

Replace the current LDLIBS line with the following lines:

```
APPLDLIBS := $(foreach lib,$(APPLIBS),$(if $(wildcard $(addprefix
$(COMPONENTS_BASE_DIR)/Library/lib,$(addsuffix
_$(JENNIC_CHIP).a,$(lib)))),$(addsuffix _$(
JENNIC_CHIP),$ (lib)),$(addsuffix _$(JENNIC_CHIP_FAMILY),$ (lib))))
LDLIBS := $(APPLDLIBS) $(LDLIBS)
```

Replace the build elf file creation option with:

```
$(TARGET)_$(JENNIC_CHIP).elf: $(APPOBJS) $(addsuffix.a,$(addprefix
$(COMPONENTS_BASE_DIR)/Library/lib,$(APPLDLIBS)))
    $(info Linking $@ ...)
    $(CC) -Wl,--gc-sections -Wl,-u_AppColdStart -Wl,-
u_AppWarmStart $(LD_FLAGS) -TAppBuildZLLHA_$(JENNIC_CHIP).ld -o $@
-Wl,--start-group $(APPOBJS) $(addprefix -l,$(LDLIBS)) -Wl,--end-
group -Wl,-Map,$(TARGET)_$(JENNIC_CHIP)$(BIN_SUFFIX).map
    $(SIZE) $@
```

Replace the bin file creation option with:

```
$(TARGET)_$(JENNIC_CHIP)$(BIN_SUFFIX).bin:
$(TARGET)_$(JENNIC_CHIP)$(BIN_SUFFIX).elf
    $(info Generating binary ...)
    $(OBJCOPY) -j .version -j .bir -j .flashheader -j
.vsr_table -j .vsr_handlers -j .rodata -j .text -j .data -j .bss
-j .heap -j .stack -S -O binary $< $@
```

In the OS_config diagram, ensure that the chip type is JN516x.

6.6.5 PDM Additional Notes

This release contains an updated PDM (Persistent Data Manager) library with a new API function.

In **vInitialiseApp** change:

```
PDM_vInit(7, 1, 64 * 1024, NULL, mutexMEDIA, NULL, &g_sKey);
```

to

```
PDM_eInitialise(63, NULL);
```

If **PDM_eInitialise(0, NULL)** already appears in your code, replace it with **PDM_eInitialise(63, NULL)**.

The new function does not use descriptors. All application variables of type **PDM_tsRecordDescriptor** must be removed.

Replace all calls to **PDM_eLoadRecord()** with:

```
PUBLIC PDM_teStatus PDM_eReadDataFromRecord(  
    uint16                ul6IdValue,  
    void                  *pvDataBuffer,  
    uint16                ul6DataBufferLength,  
    uint16                *pul6DataBytesRead);
```

Replace all calls to **PDM_vSaveRecord()** with:

```
PUBLIC PDM_teStatus PDM_eSaveRecordData(  
    uint16                ul6IdValue,  
    void                  *pvDataBuffer,  
    uint16                ul6DataLength);
```

Replace all calls to **PDM_vDeleteRecord()** with:

```
PUBLIC void PDM_vDeleteDataRecord(  
    uint16                ul6IdValue);
```

The new PDM library uses a different record format to previous versions. It is essential that old-format records are erased from the EEPROM before using the new library. This must be done from the JN516x Flash Programmer (within BeyondStudio for NXP) by selecting **Erase EEPROM->Complete PDM**.

The new PDM library does not contain the **PDM_vSave()** function. If it is required to force a save of the ZigBee PRO stack records, a call can be made to **ZPS_vSaveAllZpsRecords()**.

6.6.6 Registering an Error Handler (JN516x EEPROM Only)

The internal PDM library allows an error handler to be called to alert the application to error conditions. This error handler is registered using the function **PDM_vRegisterSystemCallback()**.

The application must trap **E_PDM_SYSTEM_EVENT_PDM_NOT_ENOUGH_SPACE** and **E_PDM_SYSTEM_EVENT_DESCRIPTOR_SAVE_FAILED** callback errors during testing.

The ZigBee PRO stack uses multiple records. Once an out-of-space error has occurred, the records will be in an inconsistent state. In this case, the software should be modified to use smaller record sizes or an external SPI Flash device.

The PDM record sizes for the ZigBee PRO stack are dependent on table sizes set in the ZPS Configuration Editor.

The following example code is provided for the error handler callback function:

```
PDM_vRegisterSystemCallback(vPdmEventHandlerCallback);
PRIVATE void vPdmEventHandlerCallback(uint32 u32EventNumber,
PDM_eSystemEventCode eSystemEventCode)
{
    switch (eSystemEventCode) {
        /*
         * The next three events will require the application to
         take some action
         */
        case E_PDM_SYSTEM_EVENT_WEAR_COUNT_TRIGGER_VALUE_REACHED:
            DBG_vPrintf	TRACE_APP, "PDM: Segment %d reached
trigger wear level\n", u32EventNumber);
            break;
        case E_PDM_SYSTEM_EVENT_DESCRIPTOR_SAVE_FAILED:
            DBG_vPrintf	TRACE_APP, "PDM: Record Id %d failed to
save\n", u32EventNumber);
            DBG_vPrintf	TRACE_APP, "PDM: Capacity %d\n",
u8PDM_CalculateFileSystemCapacity() );
            DBG_vPrintf	TRACE_APP, "PDM: Occupancy %d\n",
u8PDM_GetFileSystemOccupancy() );
            break;
        case E_PDM_SYSTEM_EVENT_PDM_NOT_ENOUGH_SPACE:
            DBG_vPrintf	TRACE_APP, "PDM: Record %d not enough
space\n", u32EventNumber);
            DBG_vPrintf	TRACE_APP, "PDM: Capacity %d\n",
u8PDM_CalculateFileSystemCapacity() );
            DBG_vPrintf	TRACE_APP, "PDM: Occupancy %d\n",
u8PDM_GetFileSystemOccupancy() );
            break;
        /*
         * The following events are really for information only
         */
        case E_PDM_SYSTEM_EVENT_EEPROM_SEGMENT_HEADER_REPAIRED:
            DBG_vPrintf	TRACE_APP, "PDM: Segment %d header
repaired\n", u32EventNumber);
            break;
        case
E_PDM_SYSTEM_EVENT_SYSTEM_INTERNAL_BUFFER_WEAR_COUNT_SWAP:
            DBG_vPrintf	TRACE_APP, "PDM: Segment %d buffer wear
count swap\n", u32EventNumber);
            break;
        case
E_PDM_SYSTEM_EVENT_SYSTEM_DUPLICATE_FILE_SEGMENT_DETECTED:
            DBG_vPrintf	TRACE_APP, "PDM: Segement %d duplicate
selected\n", u32EventNumber);
            break;
        default:
            DBG_vPrintf	TRACE_APP, "PDM: Unexpected call back Code
%d Number %d\n", eSystemEventCode, u32EventNumber);
            break;
    }
}
```


6.6.7 OS Error Checking

Errors can be detected by testing the return codes from the calls to OS functions. However, this requires application code to test the return code from every OS call. Registering an error callback function provides a robust alternative to checking the return codes. This function is invoked whenever an OS function returns an error. The function is registered as a parameter of **OS_vStart()**. The error callback option must also be enabled in the JenOS Configuration Editor. Many OS errors leave the scheduler in an undefined state. For example, nesting a mutex by calling **OS_eEnterCriticalSection()** twice before calling **OS_eExitCriticalSection()** causes a **OS_E_BAD_NESTING** error. Once an error of this nature has occurred, the OS scheduler is in an undefined state.

The OS scheduler will enter an undefined state if there are inconsistencies between the OS configuration diagram (in the JenOS Configuration Editor) and the application code. A strict error check option can be enabled in the JenOS Configuration Editor to check for inconsistencies between the OS configuration diagram and the software. The strict mode has a slight overhead in code space and execution time but it is good practice to enable strict checking where possible. For example, calling **OS_eEnterCriticalSection()** from a task which is not in the group for the mutex will generate **OS_E_CURRENT_TASK_NOT_A_MUTEX_MEMBER** with strict checking enabled. If strict checks were not enabled, the scheduler operation would be undefined and the system may become unstable.

During testing, an application's error callback function should stop the application with a stack dump and the error should be fixed. The OS passes two parameters to the error callback - the status code of the error and a pointer to the handle which caused the error. These parameters should be printed out to help determine the cause of the error. In production code, the device must be re-started from cold by calling **vAHL_SwReset()**. Data in the PDM module does not normally need to be erased, so the device can rejoin a ZigBee PRO network with existing security material. The error callback function will be called on some non-fatal errors. Depending on the application design, the following errors can be ignored by the error callback function:

- **OS_E_QUEUE_EMPTY**
- **OS_E_SWTIMER_STOPPED**
- **OS_E_SWTIMER_EXPIRED**
- **OS_E_SWTIMER_RUNNING**

6.6.8 ZigBee PRO Extended Error Status

The extended error status may be used to obtain further information about an error from the ZigBee PRO stack. The extended status information is available via a callback function:

```
ZPS_vExtendedStatusSetCallback(vExtendedStatusCb);
```

This callback function can be registered prior to initialising the stack.

An example callback function is:

```
PUBLIC void vExtendedStatusCb(ZPS_teExtendedStatus
eExtendedStatus)
{
    DBG_vPrintf(TRUE, "Extended error 0x%x\n",
eExtendedStatus);
    if (eExtendedStatus < ZPS_XS_E_RESOURCE)
    {
        DBG_vDumpStack();
        while(1);
    }
}
```

The extended error codes are defined in **zps_nwk_pub.h**. They have been categorised into fatal (ZPS_XS_E_FATAL) and resource (ZPS_XS_E_RESOURCE) errors:

- Fatal errors typically occur when a bad parameter is passed in a function call. Since the function will never complete, it is advisable to halt execution and fix the error when debugging.
- Resource errors typically occur when there are not enough NPDUs available to send a frame.

6.6.9 Beacon Filtering

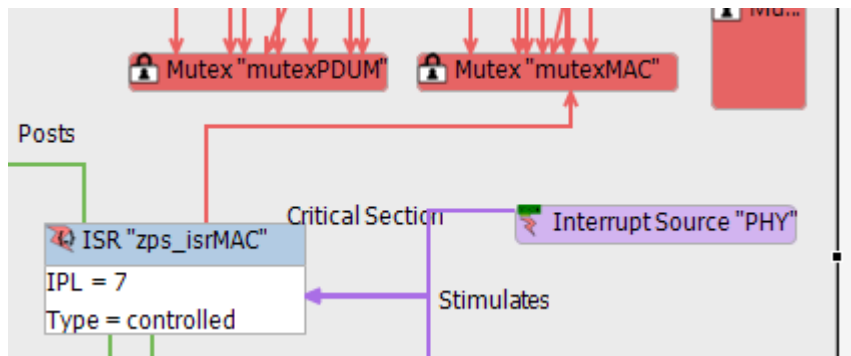
It is possible to filter the available networks on a scan so that only those networks which match the required criteria are reported to the application. This can be done by using the beacon filtering functionality provided in the file **Components/Utilities/appZpsBeaconHandler.c**.

- To switch the filter ON, **ZPS_bAppAddBeaconFilter()** should be invoked
- To switch the filter OFF, **ZPS_bAppRemoveBeaconFilter()** should be invoked

For more details, refer to the *ZigBee PRO Stack User Guide (JN-UG-3101)*.

6.6.10 Removal of PHY Interrupt from OS Diagram

In earlier HA applications, the OS diagram contained a PHY interrupt source that stimulated the MAC ISR, as shown in the following example.





In the new SDK, the application needs to remove this interrupt source, since it is no longer required in the OS diagram.

6.6.11 Removal of Management Bind Server from Application

The Management Bind Server can now be configured through the ZPS Configuration Editor. The server no longer exists in the application, since it is now part of the stack.

In earlier releases, the HA application contained the source and callback function registration for the Management Bind Server. This is no longer required in the application and must be removed.

The following files that are part of the application need to be removed from the source and also from the makefiles:

-  **app_management_bind.c**
-  **app_management_bind.h**

All the function calls that are related to these file must be removed.

RELEASE HISTORY (v1279)

7. Release Details

7.1 ZCL and Profile

7.1.1 ZCL Changes

No changes in addition to those made in v1270, described in Section 8.1.1.

7.1.2 Green Power Changes

No changes in addition to those made in v1270, described in Section 8.1.2.

7.2 ZigBee PRO Stack

ZigBee PRO libraries are included for the JN5169, JN5168 and JN5164 devices.

7.2.1 New Features and Enhancements

This release includes no new stack features in addition to those in v1270, described in Section 8.2.1.

This release includes one stack enhancement relating to the JN5169 radio settings – the default transmission power for the JN5169 device has been changed to 10 dBm.

7.2.2 Bug Fixes

In this release, the following stack bugs have been fixed:

Reference	Description
6166	Issue: In the function ZPS_vNMPurgeEntry(u64MacAddress) , the parameter type was missed in the function definition. The compiler therefore assumed it to be an int type. This had the undesired effect of shifting and setting the lower 32 bits to 0 due to type promotion. Solution: The correct parameter type has been added to the function definition.
6167	Issue: The HA Colour Dimmer Switch did not sleep if a rejoin was unsuccessful because the PAN was at full capacity. The sleep activity counter was not decremented in the stack when the 'poll deferred confirm' arrived in the wrong stack state machine handler. Solution: A change has been made to decrement the counter at the source of the event.
6171	Issue: The management network update server was not populating the channel energy detect value for channel 26. The real issue was that it was not populating the channel 11 energy detect value. Therefore, values reported for each channel were off by one channel (channel 12 value reported for channel 11, channel 13 value reported for channel 12, etc). Solution: Energy detect value is now populated for channel 11.
-	The transmit power level for the JN5168-001-M06 high-power module has been corrected (at the IEEE802.15.4 level of the stack).

7.2.3 Known Issues

Reference	Description
6086	<p>Circular Routes:</p> <p>It is possible that a unicast packet will not reach its destination because the packet is lost - for example, it becomes caught in a circular route. The APS acknowledgement mechanism can be used on the source node to monitor sent packets - an event will be generated when an acknowledgement is received from the target node. To allow an acknowledgment to be received, the source node must remain awake for a time equal to the APS retry timeout period.</p> <p>On a battery-powered node, the use of APS acknowledgements and retries may not be desirable from a power-saving point of view.</p> <p>Resolution:</p> <p>If the response is not observed within a pre-defined time then the application should take one of the following actions, depending on whether the source node is an End Device or Router:</p> <p>If an End Device, the application should notify the parent node about the routing problem by sending it a unicast network status command, ZPS_vNwkSendNwkStatusCommand(), with the status as "No Route Available (0x00)"</p> <p>If a Router, the application should initiate an explicit route discovery to the destination node by calling the function ZPS_eAplZdoRouteRequest()</p> <p>The above functions are described in the <i>ZigBee PRO Stack User Guide (JN-UG-3101)</i>.</p>
6169	<p>Discovery Failure Due to Circular Route:</p> <p>When a discovery command such as Match Descriptor is broadcast, a matching node will unicast the response back to the originator. If the response gets caught in a circular route, the discovery will not be successful.</p>
6133	<p>In a dense network it has been observed that a route reply does not get generated due to a CCA failure resulting in a Tx failure of the route reply. This prevents a route discovery from completing.</p>
6134	<p>During route discovery on a LNT, there have been instances where the route reply packet gets retried out. This results in a failure to complete the route discovery. The MAC error status returned is 'no ACK'.</p>
6267	<p>Coordinator node does not allow any new nodes to join the network, even if the first slot of the child table is available after the previous child has been removed.</p>

7.3 JN516x Integrated Peripherals API

The function **vAHI_RadioSetReducedInputPower()** has been added, which reduces the maximum radio signal power that the JN5169 device can receive before saturating. For more details, refer to the *JN516x Integrated Peripherals User Guide (JN-UG-3087)* v1.3 or higher.

The Wi-Fi counter measures for the JN5169 device that are described in the *JN516x Integrated Peripherals User Guide* are not yet available on the device.

7.4 Production Test Libraries

In this release, there are no changes in addition to those made in v1270, described in Section 8.3.

7.5 Application Porting Notes

Before porting an existing application to this JN-SW-4168 SDK, it is recommended that you go through the *ZigBee PRO Stack User Guide (JN-UG-3101)* to ensure adherence to the recommended settings in the ZPS configuration.

The libraries supplied in the JN-SW-4168 SDK have been built against the new JN-SW-4141 toolchain. The migration guidelines for the new toolchain should be followed before using these porting guidelines. Migration to the new toolchain is described in the Application Note *BeyondStudio Migration Guidelines (JN-AN-1202)*.

First note that:

- The JN516x device does not support overlays, so any code relating to overlays must be removed.
- The files **App_timer_driver.c** and **App_timer_driver.h** must be removed from the application **source/include** folders.

7.5.1 Within 'BeyondStudio for NXP'

Run BeyondStudio for NXP and follow the instructions below:

1. Select the project root directory.
2. Select **Properties** (in the **File** menu or pop-up menu obtained by right-clicking).
3. In the **Properties** window, select **C++ Build**.
4. For each node (selected from the **Configuration** drop-down list), on the **Build Settings** tab ensure that the **Build command** field contains a line of the form:

```
make JENNIC_CHIP_FAMILY=JN516x JENNIC_CHIP=JN5168 OVERLAY_BUILD=0  
PDM_BUILD_TYPE=_EEPROM
```

The above example assumes that the JN5168 chip is used.

7.5.2 Within vAppMain

The following changes should be made to the **vAppMain** code.

At the start of **vAppMain**, add the following:

```
// Wait until FALSE i.e. on XTAL - otherwise UART data will be at  
wrong speed  
while (bAHI_GetClkSource() == TRUE);  
// Now we are running on the XTAL, optimise the Flash memory wait  
states  
vAHI_OptimiseWaitStates();
```

The **Get/Release Mutex** functions for the Flash mutex should be non-counting:

```
PUBLIC void vGetMutex(void)  
{  
    OS_eEnterCriticalSection(mutexFlash);  
}  
  
PUBLIC void vReleaseMutex(void)  
{  
    OS_eExitCriticalSection(mutexFlash);  
}
```

If there is a call to:

```
u32AppApiInit(NULL, NULL, NULL, NULL, NULL, NULL);
```

it should be removed or replaced with:

```
#ifndef JENNIC_MAC_MiniMacShim
    u32AppApiInit(NULL, NULL, NULL, NULL, NULL, NULL);
#endif
```

It is now possible to rejoin a network without performing a discovery. This is achieved by calling the function **ZPS_eAplZdoRejoinNetwork()** and passing a flag of FALSE. This requires a parent to be present in the Neighbour table.

If discovery is still required then the above function should now be called with a flag of TRUE - refer to the *ZigBee PRO Stack User Guide (JN-UG-3101)* for details.

The stack no longer holds a map of short and long addresses in the NIB. It holds a map of short addresses and an index into a global long address. To find the actual long address, the following function can be called:

ZPS_u64NwkNibGetMappedExtendedAddress()

Refer to the *ZigBee PRO Stack User Guide (JN-UG-3101)* for details of the above function.

The stack no longer holds the MAC address in individual network tables. This has been replaced with a look-up table. The extended address can be found using the above function.

The indexes in the Neighbour table should never be written to by the application.

7.5.3 Setting Minimum Heap and Stack Sizes in Application

The minimum heap size can be set to a required value by defining the following in the application linker script **APP_stack_size.ld** located in the application build directory:

```
_minimum_heap_size = <required value>;
```

And the stack size can be set by:

```
_stack_size = <required size>;
```

7.5.4 Modifying Application Makefile

In the application makefile, you need to specify the type of IEEE802.15.4 MAC to use.

For all ZigBee applications, the MiniMAC will be used. It is referenced by setting the following in the makefile:

```
JENNIC_MAC ?= MiniMacShim
JENNIC_STACK ?= ZLLHA
```

The ZigBee Green Power and ZigBee Light Link optional functionality can now be controlled through the ZPS Configuration Editor.

These functionalities are present in the installer as optional libraries. To incorporate them, the following lines should be added to the Makefile:

```
JENNIC_SDK ?= JN-SW-4168
```

```

        SDK_BASE_DIR    = $(abspath
../../../../../sdk/$(JENNIC_SDK))
APP_BASE               = $(abspath ../../..)

```

Include the correct profile source in the application by defining the following flags, as required.

For an HA lighting application:

```
APP_CLUSTER_HA_LIGHTING_SRC ?= 1
```

For a ZLL lighting applications:

```
APP_CLUSTER_ZLL_SRC ?= 1
```

To include appropriate cluster code, enable one or more of the following macros, as required

```

# OTA Cluster
#APP_CLUSTERS_OTA_SRC ?=1

# Green Power clusters
#APP_CLUSTERS_GREENPOWER_SRC ?=1

# Lighting Clusters
#APP_CLUSTER_LIGHTING_SRC ?=1

# Measurement and sensing clusters
#APP_CLUSTERS_MEASUREMENT_AND_SENSING ?=1

# Energy at Home clusters
#APP_CLUSTERS_ENERGY_AT_HOME_SRC ?=1

# HVAC clusters
#APP_CLUSTERS_HVAC_SRC ?=1

# Smart Energy clusters
#APP_CLUSTERS_SMART_ENERGY_SRC ?=1

# IAS clusters
#APP_CLUSTERS_IAS_SRC ?=1

```

Replace the `OSCONFIG` , `PDUMCONFIG` and `ZPSCONFIG` lines in the makefile with the following:

```

$(OSCONFIG) -f $< -o $(APP_SRC_DIR) -v $(JENNIC_CHIP)
$(PDUMCONFIG) -z $(TARGET) -f $< -o $(APP_SRC_DIR)
$(ZPSCONFIG) -n $(TARGET) -t $(JENNIC_CHIP) -l
$(ZPS_NWK_LIB) -a $(ZPS_APL_LIB) -c
$(TOOL_COMMON_BASE_DIR)/$(TOOLCHAIN_PATH) -f $< -o
$(APP_SRC_DIR)

OPTIONAL_STACK_FEATURES = $(shell $(ZPSCONFIG) -n $(TARGET) -f
$(APP_SRC_DIR)/$(APP_ZPSCFG) -y )

```

All references to Cygwin paths/shell should be removed.

All references to `TEMPCHP` should be replaced with `JENNIC_CHIP`.

Utilities functions (e.g. NumToString) need to be built from source.

Add the utilities source directory (**C:\Jennic\Components\Utilities\Source**) to vpath:

- UTIL_SRC_DIR = \$(COMPONENTS_BASE_DIR)/Utilities/Source
- Add to vpath : \$(UTIL_SRC_DIR)

Add the utilities source files:

- APPSRC += NumToString.c
- APPSRC += appZpsBeaconHandler.c
- APPSRC += appZdpExtraction.c
- APPSRC += pdum_apdu.S

Add the following to the include path:

```
INCFLAGS += -I$(COMPONENTS_BASE_DIR)/Random/Include
INCFLAGS += -I$(COMPONENTS_BASE_DIR)/MAC/Include
INCFLAGS += -I$(COMPONENTS_BASE_DIR)/NXPLogo/Include
INCFLAGS += -I$(COMPONENTS_BASE_DIR)/Utilities
```

Replace the current LDLIBS line with the following lines:

```
APPLDLIBS := $(foreach lib,$(APPLIBS),$(if $(wildcard $(addprefix
$(COMPONENTS_BASE_DIR)/Library/lib,$(addsuffix
_$(JENNIC_CHIP).a,$(lib)))),$(addsuffix _$(
JENNIC_CHIP),$ (lib)),$(addsuffix _$(JENNIC_CHIP_FAMILY),$ (lib))))
LDLIBS := $(APPLDLIBS) $(LDLIBS)
```

Replace the build elf file creation option with:

```
$(TARGET)_$(JENNIC_CHIP).elf: $(APPOBJS) $(addsuffix.a,$(addprefix
$(COMPONENTS_BASE_DIR)/Library/lib,$(APPLDLIBS)))
    $(info Linking $@ ...)
    $(CC) -Wl,--gc-sections -Wl,-u_AppColdStart -Wl,-
u_AppWarmStart $(LD_FLAGS) -TAppBuildZLLHA_$(JENNIC_CHIP).ld -o $@
-Wl,--start-group $(APPOBJS) $(addprefix -l,$(LDLIBS)) -Wl,--end-
group -Wl,-Map,$(TARGET)_$(JENNIC_CHIP)$(BIN_SUFFIX).map
    $(SIZE) $@
```

Replace the bin file creation option with:

```
$(TARGET)_$(JENNIC_CHIP)$(BIN_SUFFIX).bin:
$(TARGET)_$(JENNIC_CHIP)$(BIN_SUFFIX).elf
    $(info Generating binary ...)
    $(OBJCOPY) -j .version -j .bir -j .flashheader -j
.vsr_table -j .vsr_handlers -j .rodata -j .text -j .data -j .bss
-j .heap -j .stack -S -O binary $< $@
```

In the OS_config diagram, ensure that the chip type is JN516x.

7.5.5 PDM Additional Notes

This release contains an updated PDM (Persistent Data Manager) library with a new API function.

In **vInitialiseApp** change:

```
PDM_vInit(7, 1, 64 * 1024, NULL, mutexMEDIA, NULL, &g_sKey);
```

to

```
PDM_eInitialise(63, NULL);
```

If **PDM_eInitialise(0, NULL)** already appears in your code, replace it with **PDM_eInitialise(63, NULL)**.

The new function does not use descriptors. All application variables of type **PDM_tsRecordDescriptor** must be removed.

Replace all calls to **PDM_eLoadRecord()** with:

```
PUBLIC PDM_teStatus PDM_eReadDataFromRecord(
    uint16          ul6IdValue,
    void            *pvDataBuffer,
    uint16          ul6DataBufferLength,
    uint16          *pul6DataBytesRead);
```

Replace all calls to **PDM_vSaveRecord()** with:

```
PUBLIC PDM_teStatus PDM_eSaveRecordData(
    uint16          ul6IdValue,
    void            *pvDataBuffer,
    uint16          ul6DataLength);
```

Replace all calls to **PDM_vDeleteRecord()** with:

```
PUBLIC void PDM_vDeleteDataRecord(
    uint16          ul6IdValue);
```

The new PDM library uses a different record format to previous versions. It is essential that old-format records are erased from the EEPROM before using the new library. This must be done from the JN516x Flash Programmer (within BeyondStudio for NXP) by selecting **Erase EEPROM->Complete PDM**.

The new PDM library does not contain the **PDM_vSave()** function. If it is required to force a save of the ZigBee PRO stack records, a call can be made to **ZPS_vSaveAllZpsRecords()**.

7.5.6 Registering an Error Handler (JN516x EEPROM Only)

The internal PDM library allows an error handler to be called to alert the application to error conditions. This error handler is registered using the function **PDM_vRegisterSystemCallback()**.

The application must trap **E_PDM_SYSTEM_EVENT_PDM_NOT_ENOUGH_SPACE** and **E_PDM_SYSTEM_EVENT_DESCRIPTOR_SAVE_FAILED** callback errors during testing.

The ZigBee PRO stack uses multiple records. Once an out-of-space error has occurred, the records will be in an inconsistent state. In this case, the software should be modified to use smaller record sizes or an external SPI Flash device.

The PDM record sizes for the ZigBee PRO stack are dependent on table sizes set in the ZPS Configuration Editor.

The following example code is provided for the error handler callback function:

```
PDM_vRegisterSystemCallback(vPdmEventHandlerCallback);
PRIVATE void vPdmEventHandlerCallback(uint32 u32EventNumber,
PDM_eSystemEventCode eSystemEventCode)
{
    switch (eSystemEventCode) {
        /*
         * The next three events will require the application to
         take some action
         */
        case E_PDM_SYSTEM_EVENT_WEAR_COUNT_TRIGGER_VALUE_REACHED:
            DBG_vPrintf(TRACE_APP, "PDM: Segment %d reached
trigger wear level\n", u32EventNumber);
            break;
        case E_PDM_SYSTEM_EVENT_DESCRIPTOR_SAVE_FAILED:
            DBG_vPrintf(TRACE_APP, "PDM: Record Id %d failed to
save\n", u32EventNumber);
            DBG_vPrintf(TRACE_APP, "PDM: Capacity %d\n",
u8PDM_CalculateFileSystemCapacity() );
            DBG_vPrintf(TRACE_APP, "PDM: Occupancy %d\n",
u8PDM_GetFileSystemOccupancy() );
            break;
        case E_PDM_SYSTEM_EVENT_PDM_NOT_ENOUGH_SPACE:
            DBG_vPrintf(TRACE_APP, "PDM: Record %d not enough
space\n", u32EventNumber);
            DBG_vPrintf(TRACE_APP, "PDM: Capacity %d\n",
u8PDM_CalculateFileSystemCapacity() );
            DBG_vPrintf(TRACE_APP, "PDM: Occupancy %d\n",
u8PDM_GetFileSystemOccupancy() );
            break;
        /*
         * The following events are really for information only
         */
        case E_PDM_SYSTEM_EVENT_EEPROM_SEGMENT_HEADER_REPAIRED:
            DBG_vPrintf(TRACE_APP, "PDM: Segment %d header
repaired\n", u32EventNumber);
            break;
        case
E_PDM_SYSTEM_EVENT_SYSTEM_INTERNAL_BUFFER_WEAR_COUNT_SWAP:
            DBG_vPrintf(TRACE_APP, "PDM: Segment %d buffer wear
count swap\n", u32EventNumber);
            break;
        case
E_PDM_SYSTEM_EVENT_SYSTEM_DUPLICATE_FILE_SEGMENT_DETECTED:
            DBG_vPrintf(TRACE_APP, "PDM: Segment %d duplicate
selected\n", u32EventNumber);
            break;
        default:
            DBG_vPrintf(TRACE_APP, "PDM: Unexpected call back Code
%d Number %d\n", eSystemEventCode, u32EventNumber);
            break;
    }
}
```

7.5.7 OS Error Checking

Errors can be detected by testing the return codes from the calls to OS functions. However, this requires application code to test the return code from every OS call. Registering an error callback function provides a robust alternative to checking the return codes. This function is invoked whenever an OS function returns an error. The function is registered as a parameter of **OS_vStart()**. The error callback option must also be enabled in the JenOS Configuration Editor. Many OS errors leave the scheduler in an undefined state. For example, nesting a mutex by calling **OS_eEnterCriticalSection()** twice before calling **OS_eExitCriticalSection()** causes a **OS_E_BAD_NESTING** error. Once an error of this nature has occurred, the OS scheduler is in an undefined state.

The OS scheduler will enter an undefined state if there are inconsistencies between the OS configuration diagram (in the JenOS Configuration Editor) and the application code. A strict error check option can be enabled in the JenOS Configuration Editor to check for inconsistencies between the OS configuration diagram and the software. The strict mode has a slight overhead in code space and execution time but it is good practice to enable strict checking where possible. For example, calling **OS_eEnterCriticalSection()** from a task which is not in the group for the mutex will generate **OS_E_CURRENT_TASK_NOT_A_MUTEX_MEMBER** with strict checking enabled. If strict checks were not enabled, the scheduler operation would be undefined and the system may become unstable.

During testing, an application's error callback function should stop the application with a stack dump and the error should be fixed. The OS passes two parameters to the error callback - the status code of the error and a pointer to the handle which caused the error. These parameters should be printed out to help determine the cause of the error. In production code, the device must be re-started from cold by calling **vAHI_SwReset()**. Data in the PDM module does not normally need to be erased, so the device can rejoin a ZigBee PRO network with existing security material. The error callback function will be called on some non-fatal errors. Depending on the application design, the following errors can be ignored by the error callback function:

- **OS_E_QUEUE_EMPTY**
- **OS_E_SWTIMER_STOPPED**
- **OS_E_SWTIMER_EXPIRED**
- **OS_E_SWTIMER_RUNNING**

7.5.8 ZigBee PRO Extended Error Status

The extended error status may be used to obtain further information about an error from the ZigBee PRO stack. The extended status information is available via a callback function:

```
ZPS_vExtendedStatusSetCallback(vExtendedStatusCb);
```

This callback function can be registered prior to initialising the stack.

An example callback function is:

```
PUBLIC void vExtendedStatusCb(ZPS_teExtendedStatus
eExtendedStatus)
{
    DBG_vPrintf(TRUE, "Extended error 0x%x\n",
eExtendedStatus);
    if (eExtendedStatus < ZPS_XS_E_RESOURCE)
    {
        DBG_vDumpStack();
        while(1);
    }
}
```

The extended error codes are defined in **zps_nwk_pub.h**. They have been categorised into fatal (ZPS_XS_E_FATAL) and resource (ZPS_XS_E_RESOURCE) errors:

- Fatal errors typically occur when a bad parameter is passed in a function call. Since the function will never complete, it is advisable to halt execution and fix the error when debugging.
- Resource errors typically occur when there are not enough NPDUs available to send a frame.

7.5.9 Beacon Filtering

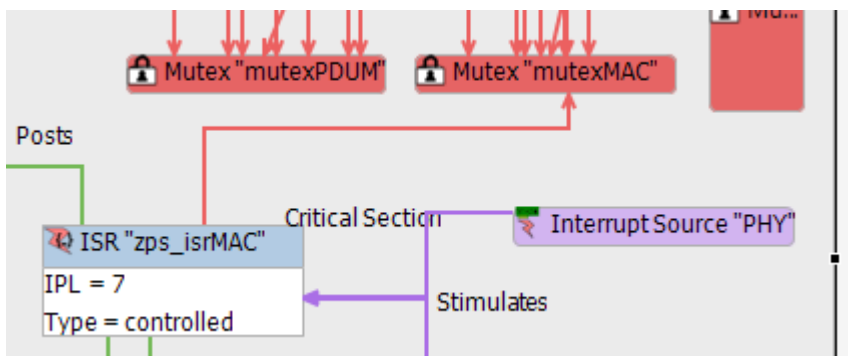
It is possible to filter the available networks on a scan so that only those networks which match the required criteria are reported to the application. This can be done by using the beacon filtering functionality provided in the file **Components/Utilities/appZpsBeaconHandler.c**.

- To switch the filter ON, **ZPS_bAppAddBeaconFilter()** should be invoked
- To switch the filter OFF, **ZPS_bAppRemoveBeaconFilter()** should be invoked

For more details, refer to the *ZigBee PRO Stack User Guide (JN-UG-3101)*.

7.5.10 Removal of PHY Interrupt from OS Diagram

In earlier HA applications, the OS diagram contained a PHY interrupt source that stimulated the MAC ISR, as shown in the following example.





In the new SDK, the application needs to remove this interrupt source, since it is no longer required in the OS diagram.

7.5.11 Removal of Management Bind Server from Application

The Management Bind Server can now be configured through the ZPS Configuration Editor. The server no longer exists in the application, since it is now part of the stack.

In earlier releases, the HA application contained the source and callback function registration for the Management Bind Server. This is no longer required in the application and must be removed.

The following files that are part of the application need to be removed from the source and also from the makefiles:

 `app_management_bind.c`
 `app_management_bind.h`

All the function calls that are related to these file must be removed.

7.6 Bootloader Version

The version of the bootloader in the JN5168 device must be either of the following:

- 0x00080003
- 0x00080006

The version of the bootloader in the JN5169 device must be the following:

- 0x000B0000

RELEASE HISTORY (v1270)

8. Release Details

8.1 ZCL and Profile

8.1.1 ZCL Changes

The ZCL in this SDK has been optimised in terms of memory usage for Flash memory and RAM by making the following changes to the previous ZCL supplied in the former HA SDK (JN-SW-4067) and ZLL SDK (JN-SW-4062).

ZCL Change 1: Shadow Structure on Client

The shadow attributes in the shared ZCL client data structure definition have been deprecated. They are not validated in attribute read/write requests and are not filtered on the responses. This enables code optimisation and flexibility to originate and receive non-profile-specific messages without allocating any storage space for them beforehand.

In the old HA and ZLL SDKs, there was a copy of the attributes structure (as a shared structure), and its control bits, in the form of an array at the client instance. This shadow structure was used to validate a general cluster command from the client. The shadow structure in a client instance was also updated with a new attribute value when an attribute read response was received or a report was received. Similarly, for a write request to go out, the attribute was first updated locally and then the write request issued.

Example:

```
#if (defined CLD_ONOFF) && (defined ONOFF_CLIENT)
/* Create an instance of an On/Off cluster as a client */
eCLD_OnOffCreateOnOff(&psDeviceInfo->
>sClusterInstance.sOnOffClient,
                        FALSE,
                        &sCLD_OnOff,
                        &psDeviceInfo->sOnOffClientCluster,
                        &au8OnOffClientAttributeControlBits[0],
                        NULL);
#endif
```

In this version of the SDK, general ZCL commands (such as read or write requests) will not be validated and will be sent to the destination address.

Similarly, a read response or report for an attribute will be directly passed to the application, and the application needs to consume the data.

In this version of the SDK, while creating a client cluster instance for a device, the attribute data structure allocation can be passed as NULL.

Example:

In the new SDK, as explained above, the shared structure and the control bits are not required for the client, and are hence passed as NULL.

```
#if (defined CLD_ONOFF) && (defined ONOFF_CLIENT)
    /* Create an instance of an On/Off cluster as a client */
    eCLD_OnOffCreateOnOff(&psDeviceInfo-
>sClusterInstance.sOnOffClient,
                        FALSE,
                        &sCLD_OnOff,
                        NULL,
                        NULL,
                        NULL);
#endif
```

The *ZCL User Guide (JN-UG-3103)* provides more details of the Create function parameters for each cluster.

ZCL Change 2: 'Write Attributes' Functions

The following functions have changed:

- **eZCL_SendWriteAttributesRequest()**
- **eZCL_SendWriteAttributesUndividedRequest()**
- **eZCL_SendWriteAttributesNoResponseRequest()**

In the old SDK, the above write functions at the client instance used the local shared structure to update the attribute values before sending them over the air.

Example:

```
eZCL_SendWriteAttributesRequest (
    uint8                u8SourceEndPointId,
    uint8                u8DestinationEndPointId,
    uint16               u16ClusterId,
    bool_t               bDirectionIsServerToClient,
    tsZCL_Address        *psDestinationAddress,
    uint8                *pu8TransactionSequenceNumber,
    uint8                u8NumberOfAttributesInRequest,
    bool_t               bIsManufacturerSpecific,
    uint16               u16ManufacturerCode,
    uint16               *pul6AttributeRequestList)
```

In the new SDK, as explained above, there is no shared structure at the client side. Hence, the above function has been changed to allow the final input parameter to take a structure of type `tsZCL_WriteAttributeRecord` (instead of a `uint16`).

```
eZCL_SendWriteAttributesNoResponseRequest (
    uint8                u8SourceEndPointId,
    uint8                u8DestinationEndPointId,
    uint16               u16ClusterId,
    bool_t               bDirectionIsServerToClient,
    tsZCL_Address        *psDestinationAddress,
    uint8                *pu8TransactionSequenceNumber,
    uint8                u8NumberOfAttributesInRequest,
    bool_t               bIsManufacturerSpecific,
    uint16               u16ManufacturerCode,
    tsZCL_WriteAttributeRecord *pul6AttributeRequestList)
```


ZCL Change 3: 'Read All Attributes' Function

The **eZCL_ReadAllAttributes()** function has been deprecated. The recommendation is now to discover attributes and read them in turn

In the old versions of the HA and ZLL SDKs, the above function could be called when the client needed to read all the attributes of the server. This function obtained a list of attributes from the local shadow structure (as indicated in the above section) and sent out a read attribute request for the cluster specified in an argument.

In the new SDK, since there is no shadow attribute structure at the client side, this method is no longer applicable and has been removed.

The new recommendation is to read the attribute by issuing a read attribute command, if the client already knows which attribute it needs to read. If not, it should perform a 'discover attributes' and then start the read if the attribute of interest is supported.

ZCL Change 4: 'Report All Attributes' Function

The behaviour of the **eZCL_ReportAllAttributes()** function has changed in the new SDK.

In the old HA and ZLL SDKs, this function sent an unsolicited report from the server application for all the attributes present in a cluster, irrespective of whether an attribute is reportable. It also sent reports for the manufacturer-specific attributes along with the other attributes.

In the new SDK, this function only sends out reports for the attributes that are configured as reportable. Also, it does not report the manufacturer-specific attributes for the reason that if the server has a manufacturer-specific attribute which the client does not recognise (different Manufacturer ID), it may generate a default response, causing confusion. Hence, this function no longer reports any manufacture-specific attributes.

The new recommendation is to read a manufacturer-specific attribute from the client through a standard 'read attribute' function call.

ZCL Change 5: Function Parameter Checks

Function parameter sanity checks are now optional and can be enabled by defining the macro **STRICT_PARAM_CHECK** in the **zcl_options.h** file for the application.

In earlier code, there were a lot of checks in the ZCL code-base for many function calls to validate their parameters, but these checks may be only necessary during development (it is advisable to have them during the development phase).

In the new SDK, there is an optional macro **STRICT_PARAM_CHECK** which, when defined, enables the parameter checks.

Example:

```
#ifdef STRICT_PARAM_CHECK
/* Parameter check */
if((psClusterInstance==NULL) ||
    (psClusterDefinition==NULL) ||
    (psCustomDataStructure==NULL))
{
    return E_ZCL_ERR_PARAMETER_NULL;
}
#endif
```

The reason for doing this is that many of these checks are in functions that are hierarchically called under a top-level root function, such as a data indication for the ZCL. Hence, the checks can be performed at the top level.

ZCL Change 6: Cooperative Scheduling

If cooperative scheduling is to be used then it must be enabled by including the macro COOPERATIVE in the **zcl_options.h** file for the application.

This release makes an optional compile-time flag COOPERATIVE available to disable the mutex events (E_ZCL_CBET_UNLOCK_MUTEX and E_ZCL_CBET_LOCK_MUTEX) in the case when all the tasks are cooperative by the nature of the application. This reduces callback events and the CPU load, and speeds up other event processing.

8.1.2 Green Power Changes

The ZigBee Green Power (GP) code has been optimised in terms of the number of times persistence is required and the size of the persistence data. The changes are listed and described below.

GP Change 1: Command Mapping

A new element `u8NoOfCmdInfo` has been added to the structure `tsGP_TranslationTableEntry`. This element should be set to the number of commands mapped in `psGpToZclCmdInfo`. This allows multiple commands to be mapped to a GP device using a single translation table entry.

```
struct tsGP_TranslationTableEntry
{
    zbmap8                b8Options;
    tuGP_ZgpdDeviceAddr   uZgpdDeviceAddr;
    uint8                 u8NoOfCmdInfo;
    tsGP_GpToZclCommandInfo *psGpToZclCmdInfo;
};
```

GP Change 2: 'Persisted Data' Structure

The structure `tsGP_PersistedData` has been changed. The previous structure members `sAttributes`, `asZgpsSinkTable` and `asZgppProxyTable` have become pointers `psAttributes`, `pasZgpsSinkTable` and `pasZgppProxyTable` respectively. Also, an enumerated type `teGP_ZgpsPersistChange` has been added, which describes which of the members have changed so that the application need not persist all the data when the callback event occurs.

```
typedef struct
{
    teGP_ZgpsPersistChange    eGPChangeCause;
    tsCLD_GreenPower          *psAttributes;
#ifdef GP_COMBO_MIN_DEVICE
    tsGP_ZgpsSinkTable        *pasZgpsSinkTable;
#endif
#ifdef GP_PROXY_DEVICE
    tsGP_ZgppProxyTable       *pasZgppProxyTable;
#endif
}tsGP_PersistedData;
```

GP Change 3: 'Restore Persisted Data' Function

The function `vGP_RestorePersistedData()` has been changed to restore only the requested parameters to their default values. Previously, this function was used to restore all the Green Power attributes to their default values. Now the option is provided to specify whether to restore sink table, proxy table or other attributes.

```
PUBLIC void vGP_RestorePersistedData(  
    tsGP_PersistedData          *psPersistedData,  
    teGP_ResetToDefaultConfig  eSetToDefault);
```

To restore all attributes to their default values, `eSetToDefault` can be set as follows:

```
eSetToDefault = E_GP_DEFAULT_ATTRIBUTE_VALUE |  
E_GP_DEFAULT_SINK_TABLE_VALUE | E_GP_DEFAULT_PROXY_TABLE_VALUE
```

8.2 ZigBee PRO Stack

ZigBee PRO libraries and APIs are included for the JN5168 and JN5164 devices.

8.2.1 New Features and Enhancements

This release includes the following new features and changes:

Reference	Description
NFR1	New child table configuration to distinguish between child and neighbour relationship.
NFR2	Extended error reporting to provide better debugging of errors reported from the ZigBee PRO stack.
NFR3	Allow rejoin to a known parent without the need to perform network scans.
NFR4	New API function to originate profile-agnostic ZigBee data requests.
NFR5	Provides network scalability to support networks larger than 250 nodes and reduce memory footprint (RAM, Flash, EEPROM).
NFR6	Provides a mechanism of beacon filtering to support faster network joins.
NFR7	Plug-ins updated for Eclipse version Kepler.
NFR8	New plug-in options to support network and APS frame-counter save thresholds.
NFR9	Provides a method for filtering beacons on discovery in order to join only relevant networks.

8.2.2 Known Issues

6086	<p>Circular Routes:</p> <p>It is possible that a unicast packet will not reach its destination because the packet is lost - for example, it becomes caught in a circular route. The APS acknowledgement mechanism can be used on the source node to monitor sent packets - an event will be generated when an acknowledgement is received from the target node. To allow an acknowledgment to be received, the source node must remain awake for a time equal to the APS retry timeout period.</p> <p>On a battery-powered node, the use of APS acknowledgements and retries may not be desirable from a power-saving point of view.</p> <p>Resolution:</p> <p>If the response is not observed within a pre-defined time then the application should take one of the following actions, depending on whether the source node is an End Device or Router:</p> <p>If an End Device, the application should notify the parent node about the routing problem by sending it a unicast network status command, ZPS_vNwkSendNwkStatusCommand(), with the status as "No Route Available (0x00)"</p> <p>If a Router, the application should initiate an explicit route discovery to the destination node by calling the function ZPS_eAplZdoRouteRequest()</p> <p>The above functions are described in the <i>ZigBee PRO Stack User Guide (JN-UG-3101)</i>.</p>
6169	<p>Discovery Failure Due to Circular Route:</p> <p>When a discovery command such as Match Descriptor is broadcast, a matching node will unicast the response back to the originator. If the response gets caught in a circular route, the discovery will not be successful.</p>
6133	<p>In a dense network it has been observed that a route reply does not get generated due to a CCA failure resulting in a Tx failure of the route reply. This prevents a route discovery from completing.</p>
6134	<p>During route discovery on a LNT, there have been instances where the route reply packet gets retried out. This results in a failure to complete the route discovery. The MAC error status returned is 'no ACK'.</p>

8.3 Production Test Libraries

The Production Test libraries for the JN516x device have been updated to 1v48.

The following updated versions of the PER and CMET Application Notes should be used with the JN516x device: JN-AN-1175, JN-AN-1172.

8.4 Application Porting Notes

Before porting an existing application to this JN-SW-4168 SDK, it is recommended that you go through the *ZigBee PRO Stack User Guide (JN-UG-3101)* to ensure adherence to the recommended settings in the ZPS configuration.

The libraries supplied in the JN-SW-4168 SDK have been built against the new JN-SW-4141 toolchain. The migration guidelines for the new toolchain should be followed before using these porting guidelines. Migration to the new toolchain is described in the Application Note *BeyondStudio Migration Guidelines (JN-AN-1202)*.

First note that:

- The JN516x device does not support overlays, so any code relating to overlays must be removed.
- The files **App_timer_driver.c** and **App_timer_driver.h** must be removed from the application **source/include** folders.

8.4.1 Within 'BeyondStudio for NXP'

Run BeyondStudio for NXP and follow the instructions below:

1. Select the project root directory.
2. Select **Properties** (in the **File** menu or pop-up menu obtained by right-clicking).
3. In the **Properties** window, select **C++ Build**.
4. For each node (selected from the **Configuration** drop-down list), on the **Build Settings** tab ensure that the **Build command** field contains a line of the form:

```
make JENNIC_CHIP_FAMILY=JN516x JENNIC_CHIP=JN5168 OVERLAY_BUILD=0  
PDM_BUILD_TYPE=_EEPROM
```

The above example assumes that the JN5168 chip is used.

8.4.2 Within vAppMain

The following changes should be made to the **vAppMain** code.

At the start of **vAppMain**, add the following:

```
// Wait until FALSE i.e. on XTAL - otherwise UART data will be at  
wrong speed  
while (bAHI_GetClkSource() == TRUE);  
// Now we are running on the XTAL, optimise the Flash memory wait  
states  
vAHI_OptimiseWaitStates();
```

The Get/Release Mutex functions for the Flash mutex should be non-counting:

```
PUBLIC void vGetMutex(void)  
{  
    OS_eEnterCriticalSection(mutexFlash);  
}  
  
PUBLIC void vReleaseMutex(void)  
{  
    OS_eExitCriticalSection(mutexFlash);  
}
```

If there is a call to:

```
u32AppApiInit(NULL, NULL, NULL, NULL, NULL, NULL);
```

it should be removed or replaced with:

```
#ifndef JENNIC_MAC_MiniMacShim  
    u32AppApiInit(NULL, NULL, NULL, NULL, NULL, NULL);  
#endif
```

It is now possible to rejoin a network without performing a discovery. This is achieved by calling the function **ZPS_eAplZdoRejoinNetwork()** and passing a flag of FALSE. This requires a parent to be present in the Neighbour table.

If discovery is still required then the above function should now be called with a flag of TRUE - refer to the *ZigBee PRO Stack User Guide (JN-UG-3101)* for details.

The stack no longer holds a map of short and long addresses in the NIB. It holds a map of short addresses and an index into a global long address. To find the actual long address, the following function can be called:

ZPS_u64NwkNibGetMappedExtendedAddress()

Refer to the *ZigBee PRO Stack User Guide (JN-UG-3101)* for details of the above function.

The stack no longer holds the MAC address in individual network tables. This has been replaced with a look-up table. The extended address can be found using the above function.

The indexes in the Neighbour table should never be written to by the application.

8.4.3 Setting Minimum Heap and Stack Sizes in Application

The minimum heap size can be set to a required value by defining the following in the application linker script **APP_stack_size.ld** located in the application build directory:

```
_minimum_heap_size = <required value>;
```

And the stack size can be set by:

```
_stack_size = <required size>;
```

8.4.4 Modifying Application Makefile

In the application makefile, you need to specify the type of IEEE802.15.4 MAC to use.

For all ZigBee applications, the MiniMAC will be used. It is referenced by setting the following in the makefile:

```
JENNIC_MAC ?= MiniMacShim  
JENNIC_STACK ?= ZLLHA
```

The ZigBee Green Power and ZigBee Light Link optional functionality can now be controlled through the ZPS Configuration Editor.

These functionalities are present in the installer as optional libraries. To incorporate them, the following lines should be added to the Makefile:

```
JENNIC_SDK ?= JN-SW-4168  
SDK_BASE_DIR = $(abspath  
../.../.../sdk/$(JENNIC_SDK) )  
APP_BASE = $(abspath ../...)
```

Include the correct profile source in the application by defining the following flags, as required.

For an HA lighting application:

```
APP_CLUSTER_HA_LIGHTING_SRC ?= 1
```

For a ZLL lighting applications:

```
APP_CLUSTER_ZLL_SRC ?= 1
```

To include appropriate cluster code, enable one or more of the following macros, as required

```
# OTA Cluster
#APP_CLUSTERS_OTA_SRC ?=1

# Green Power clusters
#APP_CLUSTERS_GREENPOWER_SRC ?=1

# Lighting Clusters
#APP_CLUSTER_LIGHTING_SRC ?=1

# Measurement and sensing clusters
#APP_CLUSTERS_MEASUREMENT_AND_SENSING ?=1

# Energy at Home clusters
#APP_CLUSTERS_ENERGY_AT_HOME_SRC ?=1

# HVAC clusters
#APP_CLUSTERS_HVAC_SRC ?=1

# Smart Energy clusters
#APP_CLUSTERS_SMART_ENERGY_SRC ?=1

# IAS clusters
#APP_CLUSTERS_IAS_SRC ?=1
```

Replace the `OSCONFIG` , `PDUMCONFIG` and `ZPSCONFIG` lines in the makefile with the following:

```
$(OSCONFIG) -f $< -o $(APP_SRC_DIR) -v $(JENNIC_CHIP)
$(PDUMCONFIG) -z $(TARGET) -f $< -o $(APP_SRC_DIR)
$(ZPSCONFIG) -n $(TARGET) -t $(JENNIC_CHIP) -l
$(ZPS_NWK_LIB) -a $(ZPS_APL_LIB) -c
$(TOOL_COMMON_BASE_DIR)/$(TOOLCHAIN_PATH) -f $< -o
$(APP_SRC_DIR)

OPTIONAL_STACK_FEATURES = $(shell $(ZPSCONFIG) -n $(TARGET) -f
$(APP_SRC_DIR)/$(APP_ZPSCFG) -y )
```

All references to Cygwin paths/shell should be removed.

All references to `TEMPCHP` should be replaced with `JENNIC_CHIP`.

Utilities functions (e.g. `NumToString`) need to be built from source.

Add the utilities source directory (**C:\Jennic\Components\Utilities\Source**) to vpath:

- UTIL_SRC_DIR = \$(COMPONENTS_BASE_DIR)/Utilities/Source
- Add to vpath : \$(UTIL_SRC_DIR)

Add the utilities source files:

- APPSRC += NumToString.c
- APPSRC += appZpsBeaconHandler.c
- APPSRC += appZdpExtraction.c
- APPSRC += pdum_apdu.S

Add the following to the include path:

```
INCFLAGS += -I$(COMPONENTS_BASE_DIR)/Random/Include
INCFLAGS += -I$(COMPONENTS_BASE_DIR)/MAC/Include
INCFLAGS += -I$(COMPONENTS_BASE_DIR)/NXPLogo/Include
INCFLAGS += -I$(COMPONENTS_BASE_DIR)/Utilities
```

Replace the current LDLIBS line with the following lines:

```
APPLDLIBS := $(foreach lib,$(APPLIBS),$(if $(wildcard $(addprefix
$(COMPONENTS_BASE_DIR)/Library/lib,$(addsuffix
_$(JENNIC_CHIP).a,$(lib)))),$(addsuffix _$(
JENNIC_CHIP),$(lib)),$(addsuffix _$(JENNIC_CHIP_FAMILY),$(lib))))
LDLIBS := $(APPLDLIBS) $(LDLIBS)
```

Replace the build elf file creation option with:

```
$(TARGET)_$(JENNIC_CHIP).elf: $(APPOBJS) $(addsuffix.a,$(addprefix
$(COMPONENTS_BASE_DIR)/Library/lib,$(APPLDLIBS)))
    $(info Linking $@ ...)
    $(CC) -Wl,--gc-sections -Wl,-u_AppColdStart -Wl,-
u_AppWarmStart $(LD_FLAGS) -TAppBuildZLLHA_$(JENNIC_CHIP).ld -o $@
-Wl,--start-group $(APPOBJS) $(addprefix -l,$(LDLIBS)) -Wl,--end-
group -Wl,-Map,$(TARGET)_$(JENNIC_CHIP)$(BIN_SUFFIX).map
    $(SIZE) $@
```

Replace the bin file creation option with:

```
$(TARGET)_$(JENNIC_CHIP)$(BIN_SUFFIX).bin:
$(TARGET)_$(JENNIC_CHIP)$(BIN_SUFFIX).elf
    $(info Generating binary ...)
    $(OBJCOPY) -j .version -j .bir -j .flashheader -j
.vsr_table -j .vsr_handlers -j .rodata -j .text -j .data -j .bss
-j .heap -j .stack -S -O binary $< $@
```

In the OS_config diagram, ensure that the chip type is JN516x.

8.4.5 PDM Additional Notes

This release contains an updated PDM (Persistent Data Manager) library with a new API function.

In **vInitialiseApp** change:

```
PDM_vInit(7, 1, 64 * 1024, NULL, mutexMEDIA, NULL, &g_sKey);
```

to

```
PDM_eInitialise(0, NULL);
```

The new function does not use descriptors. All application variables of type `PDM_tsRecordDescriptor` must be removed.

Replace all calls to **PDM_eLoadRecord()** with:

```
PUBLIC PDM_teStatus PDM_eReadDataFromRecord(  
    uint16                                ul6IdValue,  
    void                                  *pvDataBuffer,  
    uint16                                ul6DataBufferLength,  
    uint16                                *pul6DataBytesRead);
```

Replace all calls to **PDM_vSaveRecord()** with:

```
PUBLIC PDM_teStatus PDM_eSaveRecordData(  
    uint16                                ul6IdValue,  
    void                                  *pvDataBuffer,  
    uint16                                ul6DataLength);
```

Replace all calls to **PDM_vDeleteRecord()** with:

```
PUBLIC void PDM_vDeleteDataRecord(  
    uint16                                ul6IdValue);
```

The new PDM library uses a different record format to previous versions. It is essential that old-format records are erased from the EEPROM before using the new library. This must be done from the JN516x Flash Programmer (within BeyondStudio for NXP) by selecting **Erase EEPROM->Complete PDM**.

The new PDM library does not contain the **PDM_vSave()** function. If it is required to force a save of the ZigBee PRO stack records, a call can be made to **ZPS_vSaveAllZpsRecords()**.

8.4.6 Registering an Error Handler (JN516x EEPROM Only)

The internal PDM library allows an error handler to be called to alert the application to error conditions. This error handler is registered using the function **PDM_vRegisterSystemCallback()**.

The application must trap `E_PDM_SYSTEM_EVENT_PDM_NOT_ENOUGH_SPACE` and `E_PDM_SYSTEM_EVENT_DESCRIPTOR_SAVE_FAILED` callback errors during testing.

The ZigBee PRO stack uses multiple records. Once an out-of-space error has occurred, the records will be in an inconsistent state. In this case, the software should be modified to use smaller record sizes or an external SPI Flash device.

The PDM record sizes for the ZigBee PRO stack are dependent on table sizes set in the ZPS Configuration Editor.

The following example code is provided for the error handler callback function:

```
PDM_vRegisterSystemCallback(vPdmEventHandlerCallback);
PRIVATE void vPdmEventHandlerCallback(uint32 u32EventNumber,
PDM_eSystemEventCode eSystemEventCode)
{
    switch (eSystemEventCode) {
        /*
         * The next three events will require the application to
         take some action
         */
        case E_PDM_SYSTEM_EVENT_WEAR_COUNT_TRIGGER_VALUE_REACHED:
            DBG_vPrintf	TRACE_APP, "PDM: Segment %d reached
            trigger wear level\n", u32EventNumber);
            break;
        case E_PDM_SYSTEM_EVENT_DESCRIPTOR_SAVE_FAILED:
            DBG_vPrintf	TRACE_APP, "PDM: Record Id %d failed to
            save\n", u32EventNumber);
            DBG_vPrintf	TRACE_APP, "PDM: Capacity %d\n",
            u8PDM_CalculateFileSystemCapacity() );
            DBG_vPrintf	TRACE_APP, "PDM: Occupancy %d\n",
            u8PDM_GetFileSystemOccupancy() );
            break;
        case E_PDM_SYSTEM_EVENT_PDM_NOT_ENOUGH_SPACE:
            DBG_vPrintf	TRACE_APP, "PDM: Record %d not enough
            space\n", u32EventNumber);
            DBG_vPrintf	TRACE_APP, "PDM: Capacity %d\n",
            u8PDM_CalculateFileSystemCapacity() );
            DBG_vPrintf	TRACE_APP, "PDM: Occupancy %d\n",
            u8PDM_GetFileSystemOccupancy() );
            break;
        /*
         * The following events are really for information only
         */
        case E_PDM_SYSTEM_EVENT_EEPROM_SEGMENT_HEADER_REPAIRED:
            DBG_vPrintf	TRACE_APP, "PDM: Segment %d header
            repaired\n", u32EventNumber);
            break;
        case
        E_PDM_SYSTEM_EVENT_SYSTEM_INTERNAL_BUFFER_WEAR_COUNT_SWAP:
            DBG_vPrintf	TRACE_APP, "PDM: Segment %d buffer wear
            count swap\n", u32EventNumber);
            break;
        case
        E_PDM_SYSTEM_EVENT_SYSTEM_DUPLICATE_FILE_SEGMENT_DETECTED:
            DBG_vPrintf	TRACE_APP, "PDM: Segment %d duplicate
            selected\n", u32EventNumber);
            break;
        default:
            DBG_vPrintf	TRACE_APP, "PDM: Unexpected call back Code
            %d Number %d\n", eSystemEventCode, u32EventNumber);
            break;
    }
}
```

8.4.7 OS Error Checking

Errors can be detected by testing the return codes from the calls to OS functions. However, this requires application code to test the return code from every OS call. Registering an error callback function provides a robust alternative to checking the return codes. This function is invoked whenever an OS function returns an error. The function is registered as a parameter of **OS_vStart()**. The error callback option must also be enabled in the JenOS Configuration Editor. Many OS errors leave the scheduler in an undefined state. For example, nesting a mutex by calling **OS_eEnterCriticalSection()** twice before calling **OS_eExitCriticalSection()** causes a **OS_E_BAD_NESTING** error. Once an error of this nature has occurred, the OS scheduler is in an undefined state.

The OS scheduler will enter an undefined state if there are inconsistencies between the OS configuration diagram (in the JenOS Configuration Editor) and the application code. A strict error check option can be enabled in the JenOS Configuration Editor to check for inconsistencies between the OS configuration diagram and the software. The strict mode has a slight overhead in code space and execution time but it is good practice to enable strict checking where possible. For example, calling **OS_eEnterCriticalSection()** from a task which is not in the group for the mutex will generate **OS_E_CURRENT_TASK_NOT_A_MUTEX_MEMBER** with strict checking enabled. If strict checks were not enabled, the scheduler operation would be undefined and the system may become unstable.

During testing, an application's error callback function should stop the application with a stack dump and the error should be fixed. The OS passes two parameters to the error callback - the status code of the error and a pointer to the handle which caused the error. These parameters should be printed out to help determine the cause of the error. In production code, the device must be re-started from cold by calling **vAHL_SwReset()**. Data in the PDM module does not normally need to be erased, so the device can rejoin a ZigBee PRO network with existing security material. The error callback function will be called on some non-fatal errors. Depending on the application design, the following errors can be ignored by the error callback function:

- **OS_E_QUEUE_EMPTY**
- **OS_E_SWTIMER_STOPPED**
- **OS_E_SWTIMER_EXPIRED**
- **OS_E_SWTIMER_RUNNING**

8.4.8 ZigBee PRO Extended Error Status

The extended error status may be used to obtain further information about an error from the ZigBee PRO stack. The extended status information is available via a callback function:

```
ZPS_vExtendedStatusSetCallback(vExtendedStatusCb);
```

This callback function can be registered prior to initialising the stack.

An example callback function is:

```
PUBLIC void vExtendedStatusCb(ZPS_teExtendedStatus
eExtendedStatus)
{
    DBG_vPrintf(TRUE, "Extended error 0x%x\n",
eExtendedStatus);
    if (eExtendedStatus < ZPS_XS_E_RESOURCE)
    {
        DBG_vDumpStack();
        while(1);
    }
}
```

The extended error codes are defined in **zps_nwk_pub.h**. They have been categorised into fatal (ZPS_XS_E_FATAL) and resource (ZPS_XS_E_RESOURCE) errors:

- Fatal errors typically occur when a bad parameter is passed in a function call. Since the function will never complete, it is advisable to halt execution and fix the error when debugging.
- Resource errors typically occur when there are not enough NPDUs available to send a frame.

8.4.9 Beacon Filtering

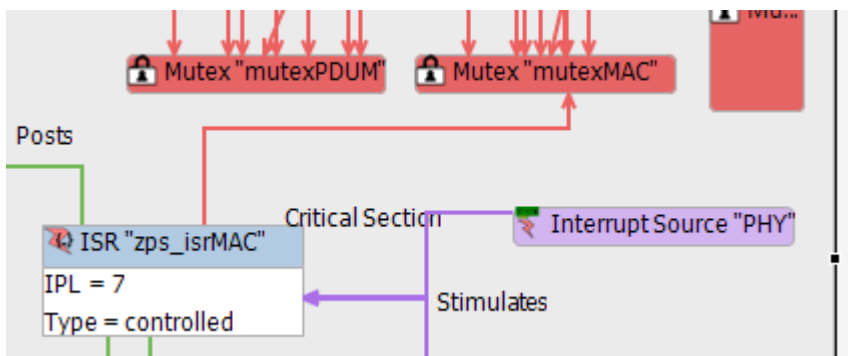
It is possible to filter the available networks on a scan so that only those networks which match the required criteria are reported to the application. This can be done by using the beacon filtering functionality provided in the file **Components/Utilities/appZpsBeaconHandler.c**.

- To switch the filter ON, **ZPS_bAppAddBeaconFilter()** should be invoked
- To switch the filter OFF, **ZPS_bAppRemoveBeaconFilter()** should be invoked

For more details, refer to the *ZigBee PRO Stack User Guide (JN-UG-3101)*.

8.4.10 Removal of PHY Interrupt from OS Diagram

In earlier HA applications, the OS diagram contained a PHY interrupt source that stimulated the MAC ISR, as shown in the following example.





In the new SDK, the application needs to remove this interrupt source, since it is no longer required in the OS diagram.

8.4.11 Removal of Management Bind Server from Application

The Management Bind Server can now be configured through the ZPS Configuration Editor. The server no longer exists in the application, since it is now part of the stack.

In earlier releases, the HA application contained the source and callback function registration for the Management Bind Server. This is no longer required in the application and must be removed.

The following files that are part of the application need to be removed from the source and also from the makefiles:

 **app_management_bind.c**
 **app_management_bind.h**

All the function calls that are related to these file must be removed.

8.5 Bootloader Version

The version of the bootloader in the JN5168 device must be either of the following:

- 0x00080003
- 0x00080006

The version of the bootloader in the JN5169 device must be the following:

- 0x000B0000