AnyNet Secure SIM Device Integration

Developer Guide

Document: 8424 v1.2

Copyright

Last updated: 13 April 2021

Contents

About this guide

This guide is designed to help you connect one or more things to Amazon Web Services (AWS) for data collection, storage and analysis purposes, using a modem of your choice and AnyNet Secure SIM card per device. This guide describes how to extract files from the provisioned AnyNet Secure SIM to your modem so that your device can securely connect to AWS.

We assume that your thing is designed to transmit data over cellular networks. You must have knowledge of AT command and cellular modem usage for data communications.

If you want to connect with other cloud providers, or to a private cloud, speak to your Account Manager.

Extra reading

For information about the other products we support that integrate with the cloud, see https://docs.eseye.com/Content/General/CloudIntegration.htm.

For information about the Eseye AnyNet+ SIM solution, see https://www.eseye.com/technology/anynet-sim/.

Standards and conventions

This guide uses consistent visual cues and standard text styles to help you locate and interpret information easily.

Style	Description
 Note	Extra information or a recommendation related to the current topic.
 Tip	Good-to-know information that helps users complete a procedure or understand a topic.
 **Warning**	Information that alerts the user about significant or critical actions or outcomes.

Title names	Window or section names, denoting a title, appear in italics.
Field or **button name**	Element names in a user interface, for example fields and buttons, appear in bold.
Ctrl+X; Ctrl+click	A key combination with a plus sign separating two key names or a key name and a mouse action, indicates that you hold down the first key while pressing the second key or performing the mouse action.
Cross reference *Title* and page	Cross references appear in italics, for example: For more information, see *Copyright* on page ii. Select the cross reference to view it.
Hyperlinks	Underlined cross references are hyperlinks to electronic forms of the document. Select the hyperlink to open the cross reference.

AT Commands

`Commands`	Command formats are displayed in monospaced typeface.
`<Parameter>`	Angle brackets enclose the AT Command parameter, for example `<topic>`. The brackets do not appear in the command line.
`"ParameterString"`	Quotation marks enclose parameter strings.
`CommandValue`	Italics in a command depicts values or examples that need replacing with your specific parameters.
`[CommandOptionalEntry]`	Square brackets display optional entries.
`ATCommandResponse`	Returned responses to AT Commands are displayed in monospaced bold typeface.
`<ASCIICHARACTERS>`	Returned ASCII characters are in uppercase.

About AnyNet Secure SIM device integration

The AnyNet Secure SIM enables you to simply, easily and securely connect your *thing* to Amazon Web Services (AWS) from anywhere in the world over cellular networks. This enables you to remotely extract data from your thing for a variety of industrial and commercial applications, such as metering, monitoring, transportation, security, and so on.

Eseye provides a single AnyNet Secure SIM with multi-IMSI capability, which enables worldwide wireless connectivity and ensures that your thing has near constant connectivity to a cellular network. Connecting to AWS provides a flexible and scalable cloud service solution for your internet of things enterprise.

Preparing your thing for the AnyNet Secure SIM

Your thing modem must support:

- SIM Toolkit, with SIM Toolkit enabled.

  For more information, see *Enabling SIM Toolkit* on the facing page.

- Restricted SIM access using the +CRSM command, as defined in 3GPP TS 27.007.

Before you begin

- Sign up for an Amazon Web Services (AWS) account, or log into an existing account: https://aws.amazon.com

  For instructions, see: https://aws.amazon.com/premiumsupport/knowledge-center/create-and-activate-aws-account/

- Create a thing in AWS IoT Core, using an AnyNet Secure SIM number as an identifier. For more information, see *Creating a thing with AnyNetThingType* on page 6.

- Ensure that the AnyNet Secure SIM with the matching number is correctly installed in your IoT device.

- Power on the modem.

- Ensure the modem can acquire a network signal. To do this, ensure you have a GPS antenna connected to the device, and the network signal strength is good.

Connecting securely to AWS

AWS provisions the AnyNet Secure SIM over a cellular network. During provisioning, the AnyNet Secure service transfers the following identity and security information to your AnyNet Secure SIM:

- The unique AWS thing name
- The Amazon Resource Name (ARN) that defines which AWS endpoint supports the thing
- A set of X.509 certificates
- An encrypted private key – AWS and the modem use key pairs for signing data

You must extract this information into the modem to enable the secure data connection from your device to the AWS IoT Core platform.

For information about monitoring provisioning progress, see *Provisioning the AnyNet Secure SIM* on page 7.

For information about which files to extract from the AnyNet Secure SIM, see *About AnyNet Secure SIM files* on page 9.

For information about how to extract the files, see *AT+CRSM – reading files from the SIM* on page 10.

Enabling SIM Toolkit

For most current modules, SIM Toolkit is already enabled.

The table below shows the commands for enabling the SIM toolkit on some modems.

Vendor	Command
Telit	`AT#STIA=1`
Quectel	`AT+QSTK=1`
Sierra Wireless	`AT+STSF=1`
Huawei	`AT^STSF=1`

Creating a thing in AWS

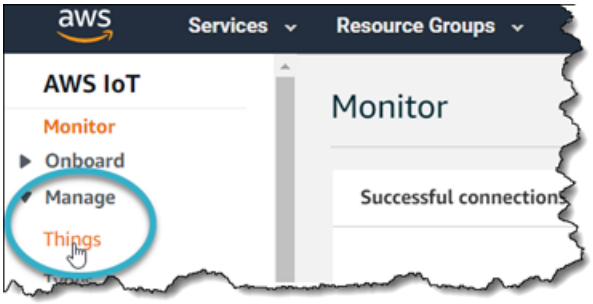
Do not create things until you have verified the advisory and alert email addresses that you supplied during AnyNet IRIS configuration. Receiving a verification link may take up to 30 minutes. If you have not received an email, contact Eseye Support: support@eseye.com.

Before you begin

- Ensure that the AnyNet Secure Cellular Connectivity configuration process is complete on your AWS account.
- Eseye connects your thing to AWS over a cellular network. To purchase the requisite SIM cards, search for Eseye SIM on Amazon.com.
- You need a AnyNet Secure SIM number, which is the unique serial number printed on the back of the SIM card.

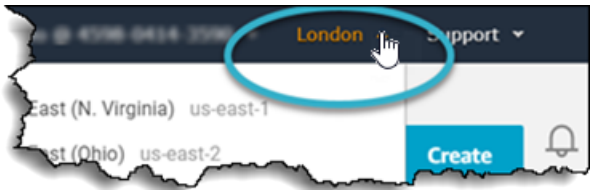
To create a thing in AWS IoT Core:

1. Sign in to the AWS Management Console: aws.amazon.com/console
2. In the *AWS Services* section, find the **IoT Core** service.
3. If this is your first time to sign into the AWS Management Console, select **Get Started**.
4. In the left-hand *AWS IoT* menu, select **Manage** > **Things**.



5. In the top right corner, select the correct AWS **Region** you want to use.

   When you create the thing, it will only exist in this Region. Select a Region where you deployed AnyNet IRIS. For more information, see *Installing AnyNet IRIS* on page 1.



6. If this is your first time creating things, select **Register a thing**, otherwise select **Create**.

   The Creating AWS IoT things page appears.

7. Select **Create a single thing**.

   The Add your device to the thing registry page appears.

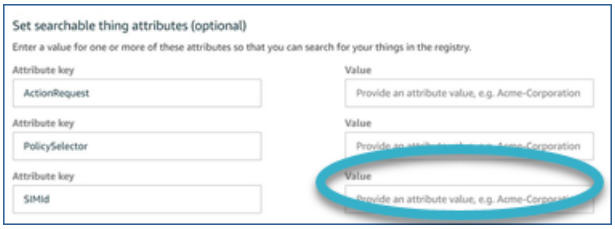
8. Type a **Name** for your thing.

   This name must match the Thing Name used by the device to connect to AWS. Thing Name is delivered to and stored in the SIM card as part of the security information. For information about delivered files, see *Available files and sizes* on page 9.

9. In the **Thing Type** drop-down box, select **AnyNetThingType**.

   For information about the AnyNetThingType, see https://docs.eseye.com/Content/General/AnyNetThingType.htm.

10. If required, add the thing to a group.
11. In the *Set searchable thing attributes* section, leave the **ActionRequest Value** field blank.
12. Leave the **PolicySelector Value** field blank.
13. In the **SimId Value** field, type the SIM number (described above).



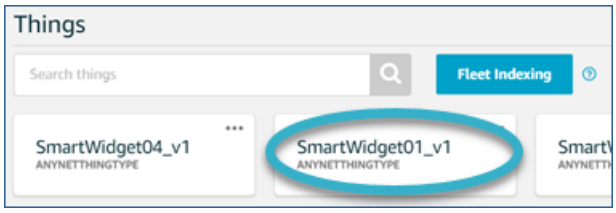
14. Select **Next**.

    The Add a certificate for your thing page appears. Eseye manages the certificate process so you don't have to.

15. Select **Create thing without certificate**.

> When you use AWS IoT in your AWS account for the first time, an AWS error may appear because Device Gateway endpoint provisioning is not complete on your account. If this occurs, allow AWS services 5-10 minutes to complete, and try again.

A Successfully created thing message appears, and the new thing appears on the Things page, for example:



Creating a thing with AnyNetThingType

When you create an AWS IoT thing that uses an AnyNet Secure SIM, it must have either of the following types:

- **AnyNetThingType** – for Eseye-designed implementation
- **CompatibleAnyNetThingType** – for bespoke implementation, to indicate that the Eseye service must manage the thing

AnyNetThingType contains the following attributes:

Attribute	Description	Mandatory?
ActionRequest	Allows actions to be performed when updating the thing	×
PolicySelector	The name of the policy that you want to attach to the thing. The policy must exist in the same AWS Region where you are creating the thing. Leave blank to use the Eseye-supplied default policy. For more information, see *IAM permissions* on page 1.	×
SimID	The SIM number of the Eseye SIM card associated with the device.	✓

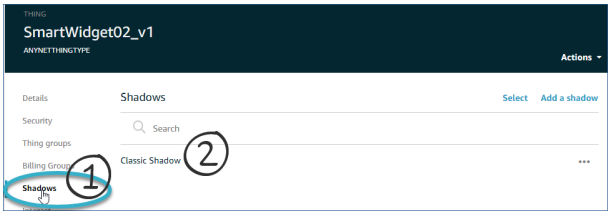
If you create custom things, CompatibleAnyNetThingType is mandatory and must contain the SimID attribute, along with your bespoke attributes.

For information about the properties of a thing, see: http://docs.aws.amazon.com/iot/latest/apireference/API_ThingAttribute.html.

Provisioning the AnyNet Secure SIM

Eseye automatically provisions the AnyNet Secure SIM. During this process, the security and identity information is downloaded and programmed into the SIM card. You can observe the provisioning progress in the device shadow. You can access the shadow using Lambda functions, programatically, or through the *AWS IoT Console*.

- Using the *AWS IoT Console*, select the thing you created using the matching SIM number, then select **Shadows > Classic Shadow** to view the *Shadow Document*.



  Use the **Shadow state** pane to view the certificate delivery progress. The certificate is delivered after the status changes from Pending to Provisioned. You can also view smart terminal message consumption and location information.

If the device regularly restarts the modem, the security and identity information delivery may slow down. We highly recommend that the device allows the modem to register on the network and remain registered for long enough to download the required file updates. Most devices will receive the file updates within 10 minutes, although the process may take up to an hour. If your AnyNet Secure SIM has not connected in 24 hours, contact Support: support@eseye.com

Provisioning normally takes 5-10 minutes to complete, although the process may take up to an hour. If your AnyNet Secure SIM has not connected in 24 hours, contact Support: support@eseye.com.

For information about setting up a terminal emulator, see .

You must wait until you receive the `ETM: IDLE` URC before you can use the device to send and receive commands.

Limiting the AWS IoT Core policy

AWS IoT Core policies allow you to control access to the AWS IoT Core data plane. The data plane consists of operations that allow you to connect to the AWS IoT Core message broker, send and receive MQTT messages, and get or update the device shadow.

By default, the AnyNet Secure provisioning service creates things with an open policy. This occurs because the provisioning has no knowledge of your application, or the publish and subscribe topics and processing you are using with your AWS account.

It is best practice to limit the policy to allow access to only the required resource and to limit that access to only authenticated devices.

We recommend that you edit or replace the installed default policy. Only `Allow` required actions or `Deny` actions that the thing never performs. Use a resource control for each action to restrict resource access.

For example, if the thing only publishes and never subscribes, remove the subscribe action from the `Allow` policy statement. Alternatively, specifically `Deny` the subscribe action. Use a resource control such as `Resource`, which restricts the connection to a thing using a thing name registered in the AWS IoT registry and authenticated against the ARN. For example:

```
["arn:aws:iot:Region:123456789012:client/${iot:Connection.Thing.AWSThingName}"]
```

For more detailed examples of how to adjust policies to manage resource access, see: AWS IoT Core policies.

About AnyNet Secure SIM files

Read AnyNet Secure SIM files using `AT+CRSM`. For more information, see *AT+CRSM – reading files from the SIM* on page 10.

File format

All files follow the same basic format:

`<length><checksum><package>`

where:

Variable	Parameter	Description
`length`	2-byte unsigned 16-bit integer	`package` byte length, part of the file header
`checksum`	4-byte unsigned 32-bit integer, stored by most significant byte first	`package` CRC-32 checksum, part of the file header
`package`	Binary data	The data field. Encryption keys and certificates may require some post-processing. For information about file location and post-processing requirements, see *Available files and sizes* below.

Available files and sizes

Eseye delivers the following files to your AnyNet Secure SIM. *File Max Size* indicates the maximum amount of data available. The file header describes the length of the actual data.

Usable Bytes refers to the number of bytes available after the header is accounted for. For more information, see *Reading the file header* on page 14.

File Name	File ID (decimal)	File ID (hexadecimal)	File Max Size (bytes)	Usable bytes
Thing Name	28640	6FE0	256	250
Root CA	28641	6FE1	2048	2042
Public signed certificate	28642	6FE2	2048	2042
Private key	28643	6FE3	2048 (not encrypted) 4096 (encrypted)	2042 4090
MQTT Endpoint	28644	6FE4	256	250

The default path ID for the available files is: `"3F007FEE"`. For more information, see *AT+CRSM – reading files from the SIM* on page 10.

For information about the basic data read process for each of the listed files, see *Basic data read process* on page 12.

AT+CRSM – reading files from the SIM

Read files from the AnyNet Secure SIM. For precise formats and response details, refer to the modem provider documentation.

We recommend that you use `AT+CRSM` over the generic SIM access `AT+CSIM`. `AT+CRSM` provides easier access to the SIM file system, as the modem facilitates the SIM interface locking and file selection routines.

The following command describes only those fields that are used and required for Eseye file extraction. Refer to 3GPP TS 27.007 for a complete definition.

Type	Syntax	Returned Result
Write	`AT+CRSM=<command>[,<fileid>[,<P1offset>,<P2offset>,<P3readlength>[,<data>[,"<pathid>"]]]]` where • `command` is the command passed on to the SIM. Use `176` – `READ BINARY` to extract security and identity information. • `fileid` is the file identifier on the SIM, of type integer. For information about which files are delivered, see *Available files and sizes* on page 9.	**`+CRSM: <sw1>,<sw2>[,"<RETURNEDDATA>"]`** or **`+CME ERROR: <err>`** where: • **`sw1`**,**`sw2`** contain information from the AnyNet Secure SIM about the command execution. These may include a success indication or specific file read errors. • **`"RETURNEDDATA"`** contains the requested data for a successful read, in ASCII hexadecimal string format. For information about converting this string into binary, see *Converting the returned data into binary* on page 13. • **`err`** is the error result code response when the command cannot pass to the SIM. To understand the value, refer to the `CME ERROR` response in the modem provider documentation.

Type	Syntax	Returned Result

	• `Pn...` parameters are used for the `READ BINARY` command.
	
	`AT+CRSM` cannot return more than 255 bytes in a single read. These fields enable the file to be read in chunks.
	
	You can use the following arithmetic operators: (+, -, /, *)
	
	  • `P1offset` is the integer value contained in the top 8 bits of the 16 bit file offset value, from which to start the file read.
	
	  • `P2offset` is the integer value contained in the bottom 8 bits of the 16 bit file offset value, from which to start the file read. `P2` can include a modulus (MOD) operator.
	
	  • `P3readlength` is the read data length as an integer. Limit the read to 255 bytes per block.
	
	• `data` is unused for `READ BINARY`. Leave empty.
	
	• `pathid` is the elementary file path on the SIM/UICC in hexadecimal format. Use `"3F007FEE"` to read files from the AnyNet Secure SIM.	

For an example of the read process, see *Basic data read process* on page 12.

Basic data read process

Repeat the basic read data process for each file that Eseye delivers to the AnyNet Secure SIM. For more information, see *Available files and sizes* on page 9.

Use the following process actions:

Read file header using the block read function

```
If ((HeaderLength != 0) and (HeaderLength != 0xFFFF) and (HeaderChecksum != 0)
and (HeaderChecksum != 0xFFFFFFFF)) then
{
     // FILE ready to read
     Read file data in chunks() // using calls to block read function()
     Verify checksum ()
     File Format Conversion () // convert DER to PEM as required
}
else
{
     // FILE not found – random back-off and try again
}
```

Read block function

We recommend that you implement a single function to request a block read of file data from the AnyNet Secure SIM.

For a description of the parameters, see *AT+CRSM – reading files from the SIM* on page 10.

Use `AT+CRSM` to achieve a block read, with the following format:

```
AT+CRSM=<command>,<fileid>,(<Offset>/256),(<Offset> MOD 256),
<readlength>,<data>,<pathid>
```

where `data` is unused and therefore empty.

This command returns:

`+CRSM: 144,0,"<RETURNEDDATA>"`

where **`"<RETURNEDDATA>"`** is an ASCII hex string that you must return to binary form.

Example

To read the *Thing Name* file header into a buffer called `returned_data`, call the read block function as follows:

`output_data` (byte pointer) = `returned_data`

- *Thing Name* `fileid`: `28640`
- `Offset`: `0`
- `readlength`: `6` bytes

```
AT+CRSM=176,28640,0,0,6,,"3F007FEE"
+CRSM: 144,0,"4D4150"
```

where "**4D4150**" is the **<RETURNEDDATA>** string for "**MAP**".

The first character in the data is the upper nibble of the first data byte. The second character is the lower nibble of the first data byte. You must convert each nibble into its hexadecimal data byte value.

Converting the returned data into binary

You must convert the "**RETURNEDDATA**" ASCII hexadecimal string into binary.

Example

The following example processes the returned string, reading two ASCII characters and converting them to binary. The resultant byte value is stored at the location defined by `output_data`:

```
index = 0;
while (index < <readlength>)
{
      sscanf( &data[index], "%02X", *output_data++);
      index = index + 2
}
```

Reading the file header

The following example uses the read block function to read the *Thing Name* header, returning 6 bytes from offset 0. 6 bytes is the header size.

Call the read block function with the following parameters:

`output_data` = `returned_data`

- *Thing Name* `fileid`: `28640`
- `offset`: `0`
- `readlength`: `6` bytes

If you use the suggested read block functionality, then the returned data is in binary format, not the ASCII hex returned by the modem.

After the response data is received, you can extract the file header length (2 bytes) and checksum (4 bytes). For more information, see *File format* on page 9.

Both values are big-endian. You may need to reverse the byte order.

```
datalength = *(unsigned short *)&returned_data[0] // read 16 bit length
checksum = *(unsigned int *)&returned_data[2] // read 32 bit checksum
```

If `datalength` is not equal to 0 and 0xFFFF, and the `checksum` is not equal to 0 and 0xFFFFFFFF, you can read the rest of the file.

Reading the file in chunks

After reading the file header, you can read the data content, starting at offset 6. Read up to 255 bytes at a time.

Check the modem command manual, as shorter length limits per read may apply.

Example

The following example reads the *Root CA* file, of length 1239 bytes.

```
file_data = malloc(1240); // allocate storage for the file read assembly
offset=6; // skip the header bytes
while (length)
{
     if (length > 255)
          bytes_to_read = 255;
     else
          bytes_to_read = length;
     // Wait for the read block response and process the data nibbles
     read_block_function(&file_data[offset – 6], 28640, offset, bytes_to_read)
     length = length – bytes_to_read
     offset = offset + bytes_to_read
     // repeat if more data to read
}
```

For a file with a data record of 927 bytes, this code results in the following call sequence:

```
read_lock_function(&file_data[0], 28641, 6, 255)
read_lock_function(&file_data[255], 28641, 261, 255)
read_lock_function(&file_data[510], 28641, 516, 255)
read_lock_function(&file_data[765], 28641, 771, 255)
read_lock_function(&file_data[1020], 28641, 1026, 219)
```

which in turn will issue the following AT commands:

```
AT+CRSM=176,28641,0,6,255,,”3F007FEE”
AT+CRSM=176,28641,1,5,255,,”3F007FEE”
AT+CRSM=176,28641,2,4,255,,”3F007FEE”
AT+CRSM=176,28641,3,3,255,,”3F007FEE”
AT+CRSM=176,28641,4,2,219,,”3F007FEE”
```

File verification

After all the data bytes are read from the AnyNet Secure SIM, you must verify them. Perform a CRC32 calculation for the data and compare it to the checksum in the header. If the sum matches, the file read is complete. Otherwise randomly backoff to wait until the file update completes, and then try again.

For more information, see *CRC32 code example* on page 17.

File format conversion

The following files are transferred in Binary DER-encoded / ASN.1 BER-encoded format:

- *Root CA* `fileid`: 28641
- *Public signed certificate* `fileid`: 28642
- *Private key* `fileid`: 28643

You may need to convert these into PEM or base64 format for use with the SSL/TLS client. Use base64 to encode the data and add header and footer lines.

Example

For the *Public signed certificate* file:

```
certificate = sprint( "-----BEGIN CERTIFICATE-----\n")
offset = strlen( certificate);
cert_offset = 0;
while (cert_offset < cert_length )
{
      Length = (cert_length - cert_offset);
      If (length > 64) length = 64;
      base64encode(encoded, certificate[cert_offset], length);
            cert_offset += length;
      Memcpy( certificate[offset], encoded, length);
            Offset += length;
      certificate[offset++] = '\n';
}
Memcpy( certificate[offset], "-----END CERTIFICATE-----\n" )
```

CRC32 code example

You can find this code example here: https://opensource.apple.com/source/xnu/xnu-1456.1.26/bsd/libkern/crc32.c. © Gary S. Brown.

First, the polynomial itself and its table of feedback terms. The polynomial is:

`X^32+X^26+X^23+X^22+X^16+X^12+X^11+X^10+X^8+X^7+X^5+X^4+X^2+X^1+X^0`

We take it "backwards" and put the highest-order term in the lowest-order bit. The `X^32` term is implied; the LSB is the `X^31` term, and so on. The `X^0` term (usually shown as `+1`) results in the MSB being `1`.

This example optimises the usual hardware shift register implementation by shifting bits into the lowest-order term, eight-bit chunks at a time. The example shifts towards the right because you must transmit the calculated CRC in order from highest-order term to lowest-order term.

UARTs transmit characters in order from LSB to MSB. By storing the CRC this way, the characters are handed to the UART in the order low-byte to high-byte. The UART sends each low-bit to high-bit. The result is transmission bit by bit from highest- to lowest-order term without requiring any bit shuffling. Reception works similarly.

The feedback terms table consists of 256, 32-bit entries.

You can generate the table at runtime if desired; code to do so is shown below. The feedback terms simply represent the results of eight `shift/xor` operations for all combinations of data and CRC register values.

The updcrc logic must right-shift the values by eight bits; the shift must be unsigned (bring in zeroes). On some hardware, you could probably optimise the shift in assembler by using byte-swap instructions polynomial $edb88320.

```
#include <sys/param.h>
#include <sys/systm.h>
static uint32_t crc32_tab[] = {
      0x00000000, 0x77073096, 0xee0e612c, 0x990951ba, 0x076dc419, 0x706af48f,
      0xe963a535, 0x9e6495a3, 0x0edb8832, 0x79dcb8a4, 0xe0d5e91e, 0x97d2d988,
      0x09b64c2b, 0x7eb17cbd, 0xe7b82d07, 0x90bf1d91, 0x1db71064, 0x6ab020f2,
      0xf3b97148, 0x84be41de, 0x1adad47d, 0x6ddde4eb, 0xf4d4b551, 0x83d385c7,
      0x136c9856, 0x646ba8c0, 0xfd62f97a, 0x8a65c9ec, 0x14015c4f, 0x63066cd9,
      0xfa0f3d63, 0x8d080df5, 0x3b6e20c8, 0x4c69105e, 0xd56041e4, 0xa2677172,
      0x3c03e4d1, 0x4b04d447, 0xd20d85fd, 0xa50ab56b, 0x35b5a8fa, 0x42b2986c,
      0xdbbbc9d6, 0xacbcf940, 0x32d86ce3, 0x45df5c75, 0xdcd60dcf, 0xabd13d59,
      0x26d930ac, 0x51de003a, 0xc8d75180, 0xbfd06116, 0x21b4f4b5, 0x56b3c423,
      0xcfba9599, 0xb8bda50f, 0x2802b89e, 0x5f058808, 0xc60cd9b2, 0xb10be924,
      0x2f6f7c87, 0x58684c11, 0xc1611dab, 0xb6662d3d, 0x76dc4190, 0x01db7106,
      0x98d220bc, 0xefd5102a, 0x71b18589, 0x06b6b51f, 0x9fbfe4a5, 0xe8b8d433,
      0x7807c9a2, 0x0f00f934, 0x9609a88e, 0xe10e9818, 0x7f6a0dbb, 0x086d3d2d,
      0x91646c97, 0xe6635c01, 0x6b6b51f4, 0x1c6c6162, 0x856530d8, 0xf262004e,
```

```
      0x6c0695ed, 0x1b01a57b, 0x8208f4c1, 0xf50fc457, 0x65b0d9c6, 0x12b7e950,
      0x8bbeb8ea, 0xfcb9887c, 0x62dd1ddf, 0x15da2d49, 0x8cd37cf3, 0xfbd44c65,
      0x4db26158, 0x3ab551ce, 0xa3bc0074, 0xd4bb30e2, 0x4adfa541, 0x3dd895d7,
      0xa4d1c46d, 0xd3d6f4fb, 0x4369e96a, 0x346ed9fc, 0xad678846, 0xda60b8d0,
      0x44042d73, 0x33031de5, 0xaa0a4c5f, 0xdd0d7cc9, 0x5005713c, 0x270241aa,
      0xbe0b1010, 0xc90c2086, 0x5768b525, 0x206f85b3, 0xb966d409, 0xce61e49f,
      0x5edef90e, 0x29d9c998, 0xb0d09822, 0xc7d7a8b4, 0x59b33d17, 0x2eb40d81,
      0xb7bd5c3b, 0xc0ba6cad, 0xedb88320, 0x9abfb3b6, 0x03b6e20c, 0x74b1d29a,
      0xead54739, 0x9dd277af, 0x04db2615, 0x73dc1683, 0xe3630b12, 0x94643b84,
      0x0d6d6a3e, 0x7a6a5aa8, 0xe40ecf0b, 0x9309ff9d, 0x0a00ae27, 0x7d079eb1,
      0xf00f9344, 0x8708a3d2, 0x1e01f268, 0x6906c2fe, 0xf762575d, 0x806567cb,
      0x196c3671, 0x6e6b06e7, 0xfed41b76, 0x89d32be0, 0x10da7a5a, 0x67dd4acc,
      0xf9b9df6f, 0x8ebeeff9, 0x17b7be43, 0x60b08ed5, 0xd6d6a3e8, 0xa1d1937e,
      0x38d8c2c4, 0x4fdff252, 0xd1bb67f1, 0xa6bc5767, 0x3fb506dd, 0x48b2364b,
      0xd80d2bda, 0xaf0a1b4c, 0x36034af6, 0x41047a60, 0xdf60efc3, 0xa867df55,
      0x316e8eef, 0x4669be79, 0xcb61b38c, 0xbc66831a, 0x256fd2a0, 0x5268e236,
      0xcc0c7795, 0xbb0b4703, 0x220216b9, 0x5505262f, 0xc5ba3bbe, 0xb2bd0b28,
      0x2bb45a92, 0x5cb36a04, 0xc2d7ffa7, 0xb5d0cf31, 0x2cd99e8b, 0x5bdeae1d,
      0x9b64c2b0, 0xec63f226, 0x756aa39c, 0x026d930a, 0x9c0906a9, 0xeb0e363f,
      0x72076785, 0x05005713, 0x95bf4a82, 0xe2b87a14, 0x7bb12bae, 0x0cb61b38,
      0x92d28e9b, 0xe5d5be0d, 0x7cdcefb7, 0x0bdbdf21, 0x86d3d2d4, 0xf1d4e242,
      0x68ddb3f8, 0x1fda836e, 0x81be16cd, 0xf6b9265b, 0x6fb077e1, 0x18b74777,
      0x88085ae6, 0xff0f6a70, 0x66063bca, 0x11010b5c, 0x8f659eff, 0xf862ae69,
      0x616bffd3, 0x166ccf45, 0xa00ae278, 0xd70dd2ee, 0x4e048354, 0x3903b3c2,
      0xa7672661, 0xd06016f7, 0x4969474d, 0x3e6e77db, 0xaed16a4a, 0xd9d65adc,
      0x40df0b66, 0x37d83bf0, 0xa9bcae53, 0xdebb9ec5, 0x47b2cf7f, 0x30b5ffe9,
      0xbdbdf21c, 0xcabac28a, 0x53b39330, 0x24b4a3a6, 0xbad03605, 0xcdd70693,
      0x54de5729, 0x23d967bf, 0xb3667a2e, 0xc4614ab8, 0x5d681b02, 0x2a6f2b94,
      0xb40bbe37, 0xc30c8ea1, 0x5a05df1b, 0x2d02ef8d
};
uint32_t
crc32(uint32_t crc, const void *buf, size_t size)
{
      const uint8_t *p;
      p = buf;
      crc = crc ^ ~0U;
      while (size--)
      crc = crc32_tab[(crc ^ *p++) & 0xFF] ^ (crc >> 8);
      return crc ^ ~0U;
}
```

Technical Support:

UK (Head office): +44 1483 802503

France: +33 9 87 67 53 37

Australia: +61 8 9551 5200

USA: +1 484-935-3130

Brazil: +55 11 4950-7015

Email: support@eseye.com

Sales:

UK (Head office): +44 1483 802501

France: +33 9 87 67 53 36

Norway: +47 454 62 017

South Africa: +27 87 551 8200

USA: +1 512-813-0599

Brazil: +55 11 5059-1574

Email: sales@eseye.com