

---

Juphoon Protocol Framework

# Signaling Compression (SigComp)

Published: Nov 2006

For more information on Juphoon Protocol Framework, see <http://www.juphoon.com>

---

## Juphoon SigComp Function Definition

Juphoon System Software Corporation.

<http://www.juphoon.com>

Tel: +86-574-87304379

Fax: +86-574-87304379

Text Part Number:

Copyright © 2007, Juphoon System Software Corporation.

All rights reserved.

## Contents

<b>1. INTRODUCTION</b> .....	<b>4</b>
1.1 SIGCOMP .....	4
1.2 AUDIENCE .....	4
1.3 SCOPE.....	4
1.4 ABBREVIATIONS .....	4
<b>2. DEFINITIONS</b> .....	<b>4</b>
2.1 BASIC DATA TYPES .....	4
2.2 TERMINOLOGY .....	5
2.2.1 <i>Application</i> .....	5
2.2.2 <i>Compressor</i> .....	5
2.2.3 <i>Compressor Dispatcher</i> .....	5
2.2.4 <i>UDVM Cycles</i> .....	5
2.2.5 <i>Decompressor Dispatcher</i> .....	5
2.2.6 <i>Endpoint</i> .....	5
2.2.7 <i>Compartment</i> .....	6
2.2.8 <i>Compartment Identifier</i> .....	6
2.2.9 <i>SigComp Message</i> .....	6
2.2.10 <i>Negative Acknowledgement (NACK) mechanism</i> .....	6
<b>3. SIPCOMP ARCHITECTURE</b> .....	<b>6</b>
3.1 COMPRESSOR DISPATCHER.....	7
3.2 DECOMPRESSOR DISPATCHER .....	8
3.3 ONE OR MORE COMPRESSORS .....	8
3.4 UDVM .....	8
3.5 STATE HANDLER .....	8
<b>4. INTERFACE STRUCTURES</b> .....	<b>8</b>
4.1 ST_ZOS_DBUF.....	8
4.2 ST_SIGC_MSG_NACK.....	9
4.3 PFN_SIGCGETCPMID.....	9
4.4 PFN_SIGCNTFYNACK .....	9
<b>5. USER INTERFACES</b> .....	<b>9</b>
5.1 SIGC_CPMCREATE .....	9
5.2 SIGC_CPMDELETE.....	10
5.3 SIGC_MSGCOMP .....	11
5.4 SIGC_MSGDECOMP .....	11
<b>6. CONFIG INTERFACES</b> .....	<b>12</b>
6.1.1 <i>SigC_CfgSetLogLevel</i> .....	12
6.1.2 <i>SigC_CfgInitByteCode</i> .....	13

---

**List of Tables**

TABLE 2-1: BASIC DATA TYPES .....5

**List of Figures**

FIGURE 3-1 HIGH-LEVEL ARCHITECTURAL OVERVIEW OF ONE SIGCOMP ENDPOINT .....7

## 1. Introduction

### 1.1 SigComp

Signaling Compression (SigComp) is a solution for compressing messages generated by application protocols such as Session Initiation Protocol (SIP) and the Real Time Streaming Protocol (RTSP). SigComp is offered to applications as a layer between the application and an underlying transport. SigComp supports a wide range of transports including TCP, UDP and SCTP.

Decompression functionality for SigComp is provided by a Universal Decompressor Virtual Machine (UDVM) optimized for the task of running decompression algorithms. The UDVM can be configured to understand the output of many well-known compressors such as DEFLATE.

### 1.2 Audience

This document is for developers who want to use the SigComp interfaces to optimize messages such as SIP messages in terms of size.

### 1.3 Scope

This document provides the definitions of SigComp interfaces for users compress messages.

### 1.4 Abbreviations

The following abbreviations are used in this document:

Abbreviation	Description
NACK	Negative Acknowledgement
SigComp	Signaling Compression
SIP	Session Initiation Protocol
RTSP	Real Time Streaming Protocol
UDVM	Universal Decompressor Virtual Machine
UDP	User Datagram Protocol

## 2. Definitions

### 2.1 Basic Data Types

There are some basic data types provided by ZOS platform.

Name	Type
ZDOUBLE	double
ZFLOAT	float

ZLONG	long
ZINT	int
ZSHORT	short
ZCHAR	char
ZULONG	unsigned long
ZUINT	unsigned int
ZSIZE_T	unsigned int
ZUSHORT	unsigned short
ZUCHAR	unsigned char
ZBOOL	int
ZVOID	void

Table 2-1: Basic Data Types

## 2.2 Terminology

### 2.2.1 Application

Application is an entity that invokes SigComp and performs the following tasks:

- 1 Supplying application messages to the compressor dispatcher
- 2 Receiving decompress messages from the decompressor dispatcher
- 3 Determining the compartment identifier for a decompress message

### 2.2.2 Compressor

Compressor is the entity that encodes application messages using a certain compression algorithm, and keeps track of the state that can be used for compression. The compressor is responsible for ensuring that the messages it generates can be decompressed by the remote UDVM.

### 2.2.3 Compressor Dispatcher

It is an entity that receives application messages, invokes a compressor and forwards the resulting SigComp compressed messages to a remote endpoint.

### 2.2.4 UDVM Cycles

It is a measure of the amount of "CPU power" required to execute a UDVM instruction (the simplest UDVM instructions require a single UDVM cycle). An upper limit is placed on the number of UDVM cycles that can be used to decompress each bit in a SigComp message.

### 2.2.5 Decompressor Dispatcher

Entity that receives SigComp messages, invokes a UDVM, and forwards the resulting decompressed messages to the application.

### 2.2.6 Endpoint

One instance of an application, a SigComp layer, and a transport layer for sending and/or receiving SigComp messages.

### **2.2.7 Compartment**

It is an application-specific grouping of messages that relate to a peer endpoint. Depending on the signaling protocol, this grouping may relate to application concepts such as "session", "dialog", "connection", or "association". The application allocates state memory on a per-compartment basis, and determines when a compartment should be created or closed.

### **2.2.8 Compartment Identifier**

It is an identifier (in a locally chosen format) that uniquely references a compartment.

### **2.2.9 SigComp Message**

A message which is sent from the compressor dispatcher to the decompressor dispatcher. In case of a message-based transport such as UDP, a SigComp message corresponds to exactly one datagram. For a stream-based transport such as TCP, the SigComp messages are separated by reserved delimiters.

### **2.2.10 Negative Acknowledgement (NACK) mechanism**

A solution which allows the recipient to communicate to the sender that a failure has occurred. This NACK contains a reason code that communicates the nature of the failure. For certain types of failures, the NACK will also contain additional details that might be useful in recovering from the failure.

## **3. SipComp Architecture**

In the SigComp architecture, compression and decompression is performed at two communicating endpoints. The layout of a single endpoint is illustrated in the figure below:

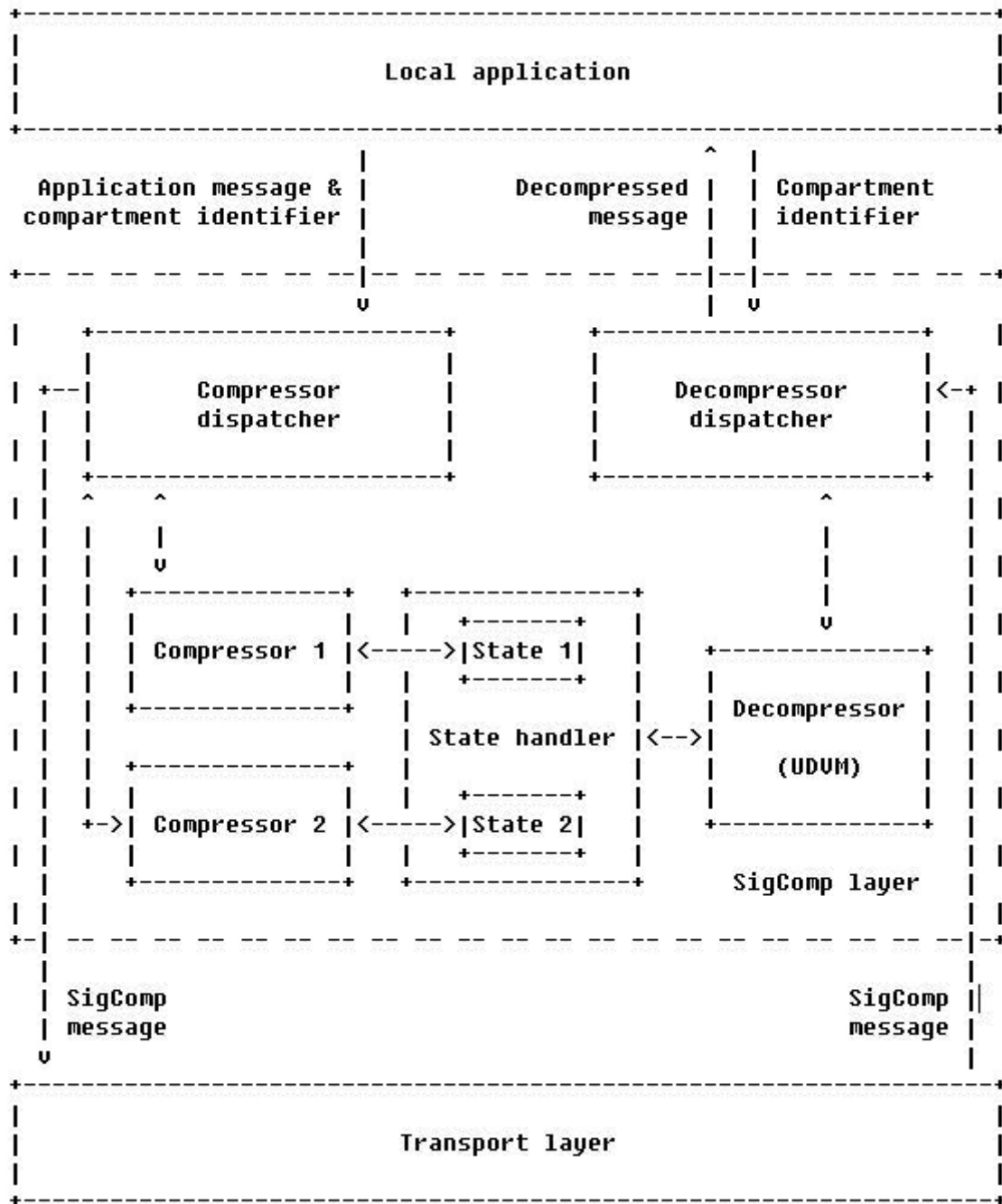


Figure 3-1 High-level architectural overview of one SigComp endpoint

Note that SigComp is offered to applications as a layer between the application and the underlying transport, and so Figure 3-1 is an endpoint when viewed from a transport layer perspective.

The SigComp layer is further decomposed into the following entities:

### 3.1 Compressor Dispatcher

The interface from the application. The application supplies the compressor dispatcher with an application message and a compartment identifier. The compressor dispatcher invokes a particular compressor, which returns a SigComp message to be forwarded to the remote endpoint.

## 3.2 Decompressor Dispatcher

The interface towards the application. The decompressor dispatcher receives a SigComp message and invokes an instance of the Universal Decompressor Virtual Machine (UDVM). It then forwards the resulting decompressed message to the application, which may return a compartment identifier if it wishes to allow state to be saved for the message.

## 3.3 One or more compressors

The entities that convert application messages into SigComp messages. Distinct compressors are invoked on a per-compartment basis, using the compartment identifiers supplied by the application. A compressor receives an application message from the compressor dispatcher, compresses the message, and returns a SigComp message to the compressor dispatcher. Each compressor chooses a certain algorithm to encode the data.

## 3.4 UDVM

The entity that decompresses SigComp messages. Note that since SigComp can run over an unsecured transport layer, a separate instance of the UDVM is invoked on a per-message basis. However, during the decompression process the UDVM may invoke the state handler to access existing state or create new state.

## 3.5 State Handler

The entity that can store and retrieve state. State is information that is stored between SigComp messages, avoiding the need to upload the data on a per-message basis. For security purposes it is only possible to create new state with the permission of the application.

# 4. Interface Structures

## 4.1 ST\_ZOS\_DBUF

```
typedef struct tagZOS_DBUF
{
    struct tagZOS_DBUF *pstNext;    /* next data buffer */
    ZPOOLID zPoolId;                /* memory pool id for dbuf alloc */
    ZULONG dwBufLen;                /* buffer used length */
    ZULONG dwDftBlkSize;            /* default data block size in buffer */
    ZUCHAR ucBufType;               /* buffer mode ZDBUF_TYPE_BYTE... */
    ZUCHAR ucRefCnt;                /* buffer reference count */
    ZUCHAR aucSpare[2];             /* for 32 bit alignment */
#ifdef ZOS_SUPT_DUMP
    ZDUMPID zDumpId;                /* stack dump */
#endif
    ST_ZOS_DBUF_DATA *pstHead;      /* the first data block in buffer */
    ST_ZOS_DBUF_DATA *pstTail;     /* the last data block in buffer */
} ST_ZOS_DBUF;
```

## 4.2 ST\_SIGC\_MSG\_NACK

```
typedef struct tagSIGC_MSG_NACK
{
    ZUCHAR ucReasonCode;           /* error reason EN_SIGC_NACK_REASON_CODE */
    ZUCHAR ucOp;                   /* opcode of failed instruction */
    ZUSHORT wPc;                   /* pc of failed instruction */
    ZUCHAR *pucMsgHash;           /* SHA-1 Hash of failed message */
    union
    {
        ST_ZOS_USTR stStateId;     /* state id for error 1, 21, 23 */
        ZUCHAR ucCpb;             /* cycles per bit for error 2 */
        ZUSHORT wDms;             /* DECOMPRESSION_MEMORY_SIZE for err 18 */
    } u;
} ST_SIGC_MSG_NACK;
```

## 4.3 PFN\_SIGCGETCPMID

This function will be invoked at the end of decompression. The sigcomp module use this interface to give the application the output of a sigcomp message, and then application should fill the id of corresponding compartment back to the sigcomp module, with which the sigcomp module will do the following action.

```
typedef ZINT (* PFN_SIGCGETCPMID) (ZULONG dwUserId, ST\_ZOS\_DBUF *pstOrgMsg,
                                   ZULONG *pdwCpmId);
```

## 4.4 PFN\_SIGCNTFYNACK

This function will be invoked when the sigcomp message is a NACK message.

```
typedef ZINT (* PFN_SIGCNTFYNACK) (ZULONG dwCpmId, ST\_SIGC\_MSG\_NACK *pstNack);
```

# 5. User Interfaces

## 5.1 Sigc\_CpmCreate

Creates a [compartment](#).

```
ZINT Sigc_CpmCreate(ZUCHAR ucCompType, ZULONG *pdwCpmId);
```

[Parameters]

Input parameters:

ZUCHAR ucCompType

Type of the compressor. The type can be EN\_SIGC\_COMP\_LZ77, EN\_SIGC\_COMP\_LZSS, EN\_SIGC\_COMP\_LZW, EN\_SIGC\_COMP\_DEFLATE, EN\_SIGC\_COMP\_LZJH, EN\_SIGC\_COMP\_MDEFLATE, EN\_SIGC\_COMP\_DEFLATED, or EN\_SIGC\_COMP\_MAX.

ZULONG \*pdwCpmId

This will point to the identifier of the compartment.

**Output parameters:**

ZULONG \*pdwCpmId

The identifier of the newly created compartment.

**[Return value]**

Returns ZOK on success, or ZFULL on failure.

**[Example]**

```
ZULONG dwCpmId;
ZUCHAR ucCompType;
ZINT iRet;
.....
/* Create a compartment. */
iRet = Sigc_CpmCreate(ucCompType, &dwCpmId);
```

## 5.2 Sigc\_CpmDelete

Deletes a [compartment](#).

```
ZINT Sigc_CpmDelete(ZULONG dwCpmId);
```

**[Parameters]**

**Input parameters:**

ZULONG dwCpmId

The compartment identifier.

**Output parameters:**

None.

**[Return value]**

None.

**[Example]**

```
ZULONG dwCpmId;
.....
/* Delete a compartment. */
Sigc_CpmDelete(dwCpmId);
```

### 5.3 Sigc\_MsgComp

Compresses a message generated by an application.

```
ZINT Sigc_MsgComp(ZULONG dwCpmId, ST\_ZOS\_DBUF *pstOrgMsg,
                  ST\_ZOS\_DBUF **ppstCompMsg);
```

#### [Parameters]

Input parameters:

```
ZULONG dwCpmId
The compartement identifier.
```

```
ST\_ZOS\_DBUF *pstOrgMsg
The application message which is to be compressed.
```

Output parameters:

```
ST\_ZOS\_DBUF **ppstCompMsg
The compressed message.
```

#### [Return value]

Returns ZOK on success, or ZFAILED on failure.

#### [Example]

```
ZULONG dwCpmId;
ST\_ZOS\_DBUF *pstOrgMsg;
ST\_ZOS\_DBUF *pstCompMsg;
ZINT iRet;
.....
/* Compress an application message. */
iRet = Sigc_MsgComp(dwCpmId, pstOrgMsg, &pstCompMsg);
```

### 5.4 Sigc\_MsgDecomp

Decompressed a SigComp message.

```
ZINT Sigc_MsgDecomp(ZULONG dwUserId, ST\_ZOS\_DBUF *pstCompMsg,
                   PFN\_SIGCGETCPMID pfnGetCpmId, PFN\_SIGCNTFYNACK pfnNtfyNack,
                   ST\_ZOS\_DBUF **ppstNackMsg);
```

#### [Parameters]

Input parameters:

ZULONG dwUserId

The user ID.

[ST\\_ZOS\\_DBUF](#) \*pstCompMsg

The received SigComp message to decompress.

[PFN\\_SIGCGETCPMID](#) pfnGetCpmId

A callback function which is used to get a compartment identifier.

[PFN\\_SIGCNTFYNACK](#) pfnNtfyNack

A callback function which deals with [NACK](#) messages.

Output parameters:

[ST\\_ZOS\\_DBUF](#) \*\*ppstNackMsg

If a decompression failure has occurred, this will point to a NACK message which will be sent to the sender.

[Return value]

Returns ZOK on success, or ZFAILED on failure.

[Example]

```
ZULONG dwUserId;
```

```
ST\_ZOS\_DBUF *pstCompMsg;
```

```
PFN\_SIGCGETCPMID pfnGetCpmId;
```

```
PFN\_SIGCNTFYNACK pfnNtfyNack;
```

```
ST\_ZOS\_DBUF *pstNackMsg;
```

```
ZINT iRet;
```

```
.....
```

```
/* Decompress a SigComp message. */
```

```
iRet = Sigc_MsgDecomp(dwUserId, pstCompMsg, pfnGetCpmId, pfnNtfyNack, &pstNackMsg);
```

## 6. Config Interfaces

These interfaces are included in sigc\_cfg.h.

### 6.1.1 Sigc\_CfgSetLogLevel

Sets the log level. Before setting, it will check whether the config has been initialized; if not, it will initialize the configuration first.

```
ZINT Sigc_CfgSetLogLevel(ZULONG dwLevel);
```

[Parameters]

**Input parameters:**

ZULONG dwLevel  
The level.

**Output parameters:**

None.

**[Return value]**

Returns ZOK.

**[Example]**

```
ZULONG dwLevel;  
.....  
/* Set the log level */  
Sigc_CfgSetLogLevel(dwLevel);
```

### 6.1.2 Sigc\_CfgInitByteCode

Initializes the byte code. It will check whether the SigComp config has been initialized; if not, it will initialize the configuration first.

```
ZINT Sigc_CfgInitByteCode(ZUCHAR ucType, ZUCHAR *pucCode);
```

**[Parameters]****Input parameters:**

ZUCHAR ucType  
The compressor type. The type can be EN\_SIGC\_COMP\_LZ77, EN\_SIGC\_COMP\_LZSS, EN\_SIGC\_COMP\_LZW, EN\_SIGC\_COMP\_DEFLATE, EN\_SIGC\_COMP\_LZJH, EN\_SIGC\_COMP\_MDEFLATE, EN\_SIGC\_COMP\_DEFLATED, or EN\_SIGC\_COMP\_MAX.

ZUCHAR \*pucCode  
The byte code to initialize.

**Output parameters:**

None.

**[Return value]**

Returns ZOK.

**[Example]**

```
ZUCHAR ucType;  
ZUCHAR *pucCode;  
  
.....  
/* Initialize the byte code. */  
Sigc_CfgInitByteCode(ucType, pucCode);
```