Institut für Datentechnik
und Kommunikationsnetze

Technische Universität
Braunschweig

OSIRIS
Command Language
Token Code Compression

Revision 1.0

IDA-OCL-0004

January 2003

Prepared by
Tim Wittrock

Table of Contents

1 Scope 3

1.1 Purpose of this Document 3

1.2 Change Record 3

1.3 Abbreviations 3

1.4 Documentation Conventions 3

2 Compression 5

2.1 Background 5

2.2 Algorithm 5

3 Compression Table Generator 6

3.1 Usage 6

4 Compression Table Format 7

5 The Decompression Table Format 8

1 Scope

1.1 Purpose of this Document

This document gives a short overview over the compression algorithm used for OCL token code compression and decompression, describes the format of the compression and decompression tables and explains the handling of the compression table generator tool.

1.2 Change Record

Date	Revision	Author	Affected Sections
01/10/2003	1.0	Wittrock	All sections

1.3 Abbreviations

RTL	Runtime Library
OCL	Osiris Command Language
OCT	Osiris Command Token

UDP	User-defined Procedure
POP	Persistent Osiris Program

1.4 Documentation Conventions

The following text will specify the typing conventions that are used to explain the OSIRIS command language syntax.

- **`Keywords`** have to be typed exactly as shown in the documentation. ¿ represents an end of line (new line).
- ***References*** refer to other elements. It can be for example a user-defined name or a complex expression defined in the documentation.
- **:=** following a ***reference*** specifies the definition of the ***reference***.
- **[** ***Expressions*** **]** enclosed in square brackets are optional and may be omitted.
- Enclosing **(** ***expressions*** **)** in brackets is used for grouping.
- **(** ***Expressions*** **)+** in brackets followed by a plus may appear one or more times.
- **(** ***Expressions*** **)*** in brackets followed by an asterisk may be omitted or appear one or more times.
- ***Statements*** are lists of expressions separated by semicolon.

- The vertical rule **|** between two expressions indicates an alternative. You may choose either the expression on the left or on the right side.

```
Examples are printed this way   // comments are placed behind “//”
```

2 Compression

2.1 Background

UDP code consists mainly of token sequences. Each token consists of 4 different 32 bit values, which contain

- the line number of the corresponding source code (24 bits),
- the token (8 bits),
- three parameters, each 32 bits long.

Most parameters and the line numbers are small values with the upper three bits set to 0.

As one can see, nearly half of the token code consists of zero bytes. Therefore, these zeros should be represented by a very short bit sequence. Combining 16 bit allows a compression of the upper two zero bytes from 16 bit into 1 bit. The lower 16 bits, which contain small values (local addresses of variables) in most cases, can also be compressed by quite short bit sequences. Since zero bytes are such immense part of the token code, the compression of these zeros is the most relevant factor. Comparison between full 8-bit Huffman compression and 16-bit Huffman compression of only the most frequent 256 16-bit values (all other values were represented by a lead-in sequence followed by the value itself) showed that 16-bit compression is more effective for UDP code.

2.2 Algorithm

The OCL compression algorithm is a modified Huffman algorithm. It codes 16-bit values into bit sequences of different length, the most frequent values are represented by the shortest bit sequence, the least frequent ones by the longest sequence.

Since on-board memory is rare, it is not possible to use a table with specific entries for every single 16-bit value (this table would be at least 65535 lines long, each line at least 16 bits long, i.e. 131070 bytes), so only the most frequent values (e.g. 256) are part of the table. Only these values are coded by short bit sequences, all the others are represented by a special lead-in bit sequence which is followed by the value itself. This allows using a table which consumes only 2048 bytes of on-board memory with no significant loss of compression rate.

The decompression table is not part of the compressed token code packets. That means, even short token packets can be compressed by a factor similar to large token packets. If the built-in compression/decompression tables do not allow a sufficient compression rate, it is possible to use user generated tables instead.

3 Compression Table Generator

The compression table generator should be used to generate a user defined compression table for the OCL compiler and the corresponding decompression table for the UDP manager.

3.1 Usage

The generator can be called as follows:

`TableGen` **`[`** **`-n`** ***number*** **`]`** **`[`** **`-in`** ***inputfile*** **`]`** **`[`** **`–cpr`** ***compressiontable*** **`]`** **`[`** **`–decpr`** ***decompressionbinary*** **`]`** **`[`** **`-decprtxt`** ***decompressiontext*** **`]`**

The generator reads in the file named ***inputfile*** and examines it for the ***number*** most frequently used values. This ***number*** should be not too big, since the decompression table (which has to be uploaded and stored on-board) grows with the amount of values; the default for ***number*** is 254 (+ table entry for lead-in of uncompressed 16-bit values + code for repetition of 32-bit values = 256 table entries). To achieve best results, ***inputfile*** should be representative for your token code.

The compression table for the OCL compiler is written into the file ***compressiontable***, the decompression information which is required by the UDP manager on-board is written into the file ***decompressionbinary***. This file can be uploaded to the UDP manager directly. If you specify ***decompressiontext***, the generator writes the decompression information into an ASCII file of this name. This file can be used as source code for the built-in table. Normally, **`-decprtxt`** ***decompressiontext*** should be omitted, since this file is not useful for the user.

All parameters may be omitted. In this case, the generator reads in the file **`input.bin`**, writes the compression table into **`CompressionTable.txt`** and writes the decompression table as binary data ready for upload into **`DecompressionTree.bin`**.

4 Compression Table Format

Normally, the compression table generator program should be used to create the compression table file and the corresponding decompression table. The OCL compiler can read in compression tables which are stored in an ASCII file using the following format:

```
MagicNumber: hex_value ¿
// comment_line ¿
( { value ,  sequence_length , sequence }, ¿ )+
  { 0x1ffff , sequence_length , lead_in },
[ { 0x2ffff , sequence_length , sequence }, ]
```

The first line specifies the magic number of this compression table. This number is used to prevent errors which may be caused by decompressing uploaded token codes with improper decompression rules.

The ***comment_line*** may not be omitted (but a blank line is ok), since the OCL compiler ignores the second line of the compression table file, regardless of it's content.

All 16-bit ***value***s (0x0000 up to 0xffff) have to be in ascending order. The over-all length in bits of the bit ***sequence*** which encodes the ***value*** is specified by ***sequence_length***.

The value `0x1ffff` indicates the compression rule for all 16-bit values which have no compression sequence for their own; `lead_in` specifies the bit sequence which precedes the uncompressed value in the bit stream (***sequence_length*** specifies the over-all size of the ***lead_in*** plus the 16-bit of the value itself).

The value `0x2ffff` specifies the sequence for 32-bit value repetition. The ***sequence*** specified here has to be followed in the compressed bit-stream by the 32-bit value and a 16-bit repetition counter. This compression feature is not implemented yet.

5 The Decompression Table Format

The binary decompression table consists of multiple lines; each line contains information for two cases:

- The current bit of the sequence is not set (***Action*** $_0$)
- The current bit is set (***Action*** $_1$).

These lines are followed by the corresponding compression table's magic number and an over-all-checksum.

Table* := (*Table_line*)+ *Magic_number Checksum

A 'line' of the table is built by two 32-bit values:

Table_line* := *Action $_0$ ***Action*** $_1$

Where each ***Action*** value consists of the following bits:

Action* := *Continue Base Bytes Repeat Unused Value

The meaning of each of these parts is described in the following table:

	Bits	Meaning
Continue	2	0: End of sequence reached, decompresses to ***Value*** from table. The number of valid bits depends on ***Base*** (0 or 1). 1: End of lead-in reached, decompresses to the following value from bit stream. The number of bits read from bit stream depends on ***Base***. 2: Get next bit and continue at line ***Value*** of this table. All other bits in this rule are ignored.
Base	2	0: 8-bit value encoding 1: 16-bit value encoding 2: 24-bit value encoding 3: 32-bit value encoding
Bytes	2	Size of repetition counter (if ***Repeat*** = 1) or number of units to set to current value 0: 8 bits (1 unit) 1: 16 bits (2 units) 2: 24 bits (3 units) 3: 32 bits (4 units)
Repeat	1	0: no repetition 1: repeat current value ***n*** times, where ***n*** is specified in the compressed bit

		stream (behind the value, if ***Continue*** = 1)
Unused	9	
Value	16	Decompressed value

Although the decompression table format allows quite flexible compression, like changing between 8-bit and 16-bit Huffman encoding and n-times repetition of values, the current OCL compiler uses 16-bit compression only and none of the other features. The built-in decompression table of the UDP manager supports 16-bit compression and the repetition of 32-bit values only.