Alcatel NSS Gateway User Guide

Gateway Release: 3.3.1

Date: 26 October 2007

Contents

CONTENTS 2

REFERENCES 4

GLOSSARY 4

PREFACE 5

ABOUT THIS GUIDE 5

CONVENTIONS 5

1. OVERVIEW 6

1.1 THE GATEWAY FRAMEWORK 6

1.2 ALCATEL NSS GATEWAY OVERVIEW 6

1.2.1 Network Details 6

1.2.2 Data Version Support 7

1.2.3 Data/File Formats 7

1.2.4 File Naming Convention 10

1.2.5 Architectural extensions 11

1.2.6 Log Messages 11

2. ENGINE RULES AND CONFIGURATION 11

2.1 ALCATEL_RCP 11

2.2 ALCATEL_OCB 12

2.3 ALCATEL_LM 12

2.4 ALCATEL ENGINE RULE CONFIGURATIONS 12

2.4.1 Common Rule Entries 12

2.4.2 Alcatel RCP Rule Configuration 13

2.4.3 Alcatel OCB Rule Configuration 16

2.4.4 Alcatel LM Rule Configuration 19

3. POST PARSER RULES AND CONFIGURATION 25

3.1 HEADER_2_RECORD_TRANSFER 25

3.1.1 Rule Configuration 25

3.1.2 Sample Usage 26

3.2 TRANSPOSE 27

3.2.1 Rule Configuration 27

3.2.2 Sample Usage 28

3.3 ROTATE 29

3.3.1 Rule Configuration 29

3.3.2 Sample Usage 29

4. TECH PACK SUPPORT 32

References

[Gateway Framework User Guide]

Glossary

PIF	Parser Intermediate File
LIF	Loader Input File
HLR	Home Location Register
VLR	Visitor Location Register
SSB	Service Switching Point
RCP	Routing Core Platform
MMC	Man-Machine-Communication
SCCP	Signalling Connection Control Part

Preface

About this Guide

This guide details the vendor specific information on the Gateway release for Alcatel NSS. It contains the following information:

- *Chapter 1. Overview.* This chapter gives a brief description of the vendor gateway and the raw data format it parses.
- *Chapter 2: Engine Rules and Configuration.* This chapter details the vendor specific rules for parsing the raw data and their configuration.
- *Chapter 3: Post Parser Rules and Configuration.* This chapter describes any vendor specific Post Parser rules and their configuration.
- *Chapter 4: Tech Pack Support.* This chapter describes any standard support for Tech Packs included with the Gateway.
- *Chapter 5: Installation specific information.* This chapter contains the customer installation specific information.

Conventions

The following conventions are used in this guide:

`fixed width`	Highlights a block of example code, a configuration entry, or a command line instruction

1. Overview

1.1 The Gateway Framework

The Alcatel NSS Gateway uses the Gateway Framework as a container for the execution of its transfer, engine and post parser stages. The Gateway Framework and the Alcatel NSS Gateway are decoupled into two separate installations. The Gateway Framework consists of a library of perl modules that provide functionality such as:

- a container for the execution of the Vendor Engine and Post Parser rules for of data transformation
- Intermediate (PIF) and output data (LIF) storage and management
- logging utilities
- cleanup and crash recovery
- statistics gathering

The Alcatel NSS Gateway plugs into the Gateway Framework and extends this functionality to provide the Gateway functionality that processes the vendor data. More information on the standard Gateway configuration is contained in the [Gateway Framework User Guide].

Only vendor specific configuration details will be described in this document.

1.2 Alcatel NSS Gateway Overview

This section contains an overview of the Alcatel NSS Gateway and the network components, whose data the Gateway processes.

1.2.1 Network Details

The Alcatel NSS Gateway is design to transfer, parse, and further process data from Alcatel NSS components, for loading in the Tivoli performance management systems. The specific components of interest are the HLR/RCP, SSB/OCB and LM contained in the Alcatel NSS system.

This Alcatel NSS Gateway is specific to the French switch of the Alcatel NSS components.

1.2.1.1 HLR/RCP

The Gateway processes performance data from the Alcatel 900 and 1800 NSS format for the HLR Routing Core Platform (RCP) R5, R6 and R8 component.

1.2.1.2 OCB/SSP

The Alcatel OCB Service Switching Point (SSP) series of switches are used for PSTN and transit application and they have given the name S12 for their switches, which are being using in wireless network mainly GSM. The Alcatel NSS Gateway processes performance data from the Alcatel OCB v24.

1.2.1.3 LM

The Alcatel NSS Gateway processes performance data from the LM switch for version 7 and version 9.

1.2.2 Data Version Support

This gateway supports data from the following Alcatel NSS switches.

- Alcatel NSS HLR/RCP R5, R6 and R8
- Alcatel NSS OCB/SSP R24
- Alcatel NSS LM R7 and R9, Flexible ADL Measurement

1.2.3 Data/File Formats

All data from HLR/RCP, SSP/OCB and LM are in binary format. Their respective layouts are described here.

1.2.3.1 Data Layout – RCP

The binary data contained in the RCP binary files is organized into blocks of data. Each block of data consists of a header record and multiple data records. The header record is described in table 1.

Counter name	Size	Description
Year	1 byte integer	Date year; i.e. 04
Month	1 byte integer	Date month; i.e. 12
Date	1 byte integer	Date day of month; i.e. 29
Hour	1 byte integer	Date hour; i.e. 10
Minute	1 byte integer	Date minutes; i.e. 15
Second	1 byte integer	Date seconds; i.e. 00
Hd_disp	1 byte integer	
Hd_noofticket	1 byte integer	Header number of ticket
Hd_ticketno	1 byte integer	Header ticket number

Table 1 – RCP Header Record fields description

RCP data records are either 10 or 12 bytes, depending on the size of the rank counter. This is determined by the value of the counter type integer (3rd byte). RCP records are described in table 2.

Counter name	Size	Description
Observation Record family ID	2 byte unsigned integer.	ID that determines the family group, the record belongs to. Used to generate the counter name.
Counter type	1 byte	Determines rank ID size. 1,2 is a 2 byte rank. 3,4 is a 4 byte rank.
Rank ID within family	2 or 4 byte hex string depending on counter type.	
Counter value	4 byte big endian integer	

Counter validation value	1 byte char	

Table 2 – RCP Data Record field description

1.2.3.2 Data Layout – OCB

Each OCB file is composed of a series of block structures. Each block is identified by a block header entry, as described in table 3. The block header is follow by a series of data records, each starting with a record header. The header record is described in table 4.

Counter name	Size	Description
Block length	Two byte big endian integer	Block length
Reserved	Two bytes	

Table 3 – OCB block header record fields description

Counter name	Size	Description
Record length	Two byte big endian integer	Header record length
Reserved	Two bytes	
Record code	1 byte integer	Header class ID
Year	Two byte big endian integer	Date year day; i.e. 0419
Month	1 byte integer	Date month; i.e. 12
Hour	1 byte integer	Date hour; i.e. 10
Minute	1 byte integer	Date minutes; i.e. 15
Second	1 byte integer	Date seconds; i.e. 00
Exchange number	1 byte integer	Exchange number

Table 4 – OCB header record fields description

OCB data record byte allocation is dependent on the record class ID. Table 5 is an example of a byte allocation for a permanent observation class C record.

Counter name	Size	Description
CI (in tenths of an ERLANG)	Four byte big endian integer	
CD (in tenths of an ERLANG)	Four byte big endian integer	
CA (in tenths of an ERLANG)	Four byte big endian integer	
CT (in tenths of an ERLANG)	Four byte big endian integer	
CC (in tenths of an ERLANG)	Four byte big endian integer	
CV (in tenths of an ERLANG)	Four byte big endian integer	

Table 5 – OCB class C record fields description

1.2.3.3 Data Layout – LM

The arrangement of data structure in LM record is illustrated as below:

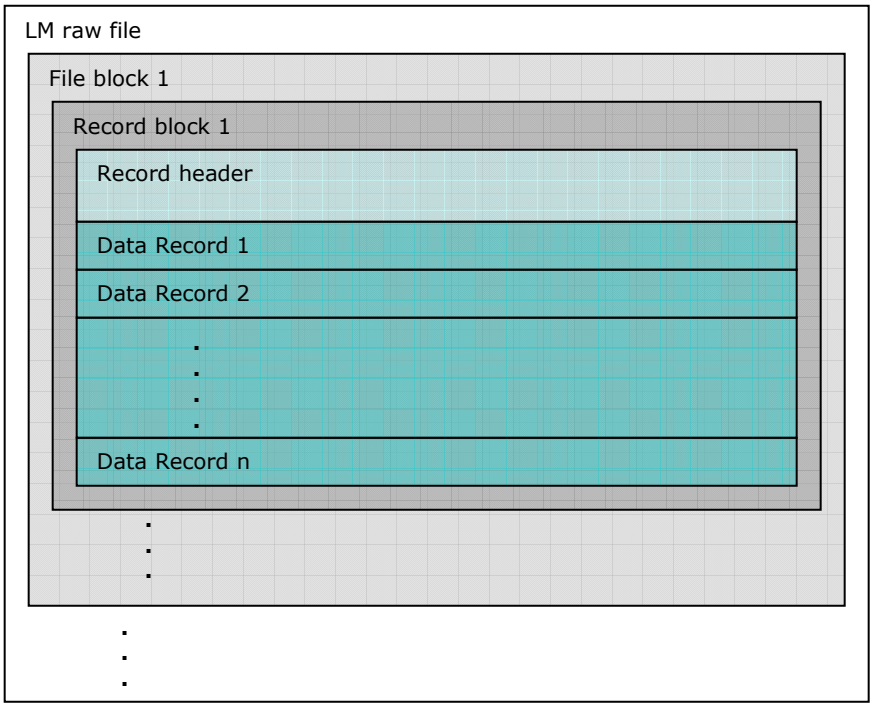

Diagram 1: LM raw data layout

An LM raw file must have one or more file blocks (with fixed length) in it. Multiple record blocks are sharing a single file block. Within a record block, there will be always a record header with at least one data record.

Table 6 shows the field descriptions of a file block header. Note that the Block length does not include the first 6 bytes of this file block header.

Counter name	Size	Description
Block number	Two byte little endian integer	Block number
Reserved	Two bytes	Reserve byte and Spanned flag
Last used byte	Two byte little endian integer	Block length

Table 6 – LM file block header field description

A record block always contains a record header, followed by multiple data records. Table 7 shows the example of a record header field description that belongs to record block type SCPHIST of file type SCCP.

Counter name	Size	Description
Record length	Two byte little endian integer	Header record length

MMC number	Two byte little endian integer	Record type
Reserved	26 bytes	
Year	Two byte little endian integer	Date Year; i.e. 2004
Month	1 byte little endian integer	Date month; i.e. 12
Day	1 byte little endian integer	Date day; i.e. 25
Hour	1 byte little endian integer	Date hour; i.e. 10
Minute	1 byte little endian integer	Date minutes; i.e. 15
Reserved	6 bytes	

Table 7 – Sample LM record header field description (SCCP SCPHIST header)

Each different MMC number (from record header) has its own unique data record byte allocation. It is possible to have multiple data records within a single record block. Table 8 is an example of data record for record block type SCPHIST.

Counter name	**Size**	**Description**
Reserved	8 bytes	
SSN	One byte integer	
Threshold	One byte integer	
Hour	One byte integer	
Minute	One byte integer	
Second	One byte integer	
Tenth_of_second	One byte integer	

Table 8 – SCPHIST data record byte allocation

1.2.4 File Naming Convention

1.2.4.1 HLR/RCP and OCB/SSP File Naming

For HLR/RCP and OCB/SSP Alcatel NSS raw files must follow the naming convention as below:

<OMC>.<NODE>

where,

<OMC> : 3 digit hexadecimal followed by the OMC.
<NODE> : this is the switch name who's generated the raw file.

1.2.4.2 LM File Naming

For LM Alcatel NSS raw files must follow the naming convention as below:

<MSCID>.<TYPE>

where,

<MSCID> : the MSC ID and name.

<TYPE> : this the type (file type) of the raw file.

1.2.5 Architectural extensions

There are no specific architectural extensions to the Alcatel NSS Gateway, such as XML parsing etc.

1.2.6 Log Messages

The log messages will be recorded in the log file as configured inside the properties. Basically there are 3 general types of log messages.

a. Informative messages
   - General informative description about parser current state. Example: the information about current block read (byteReadCount, blockSize, recordLength)

- Warning messages
  - Informing the abnormalities during parsing. The warning messages are not critical and can safely be ignore at point of existence. The messages mostly used during debugging process if there are problem arises during parsing activities.
    Example: Warning: -get_pif_key - record_type missing from binary data.
    (In this case, the record_type [MMC number] is missing when the gateway trying to use it as a PIF key record)

- Error messages
  - Pointing out the problem during parsing. This is critical messages as it describes the reason of why the parser failed to parse the data. Most of the time the parser will the stop the processing and skip the raw data.
    Example: Unexpected end of file or error reading file.

2. Engine Rules and Configuration

This section describes the Alcatel NSS engine rules and rule configurations.

2.1 ALCATEL_RCP

The binary parsing of the Alcatel RCP data is handled by the ALCATEL_RCP engine rule. The rule is designed to parse through each individual RCP performance data file and produce unique PIF files for each block of data.

The ALCATEL_RCP engine is designed to be a flexible tool, which can be used to map every byte in the binary RCP file. The header and record byte mappings for each block are contained in the configuration file entries HEADER_MAPPING and

RECORD_MAPPING respectively. These two record mapping configurations are sufficient to map and parse the entire file. If a change in the record design occurs, the new mapping can be achieved by updating the configuration accordingly.

2.2 ALCATEL_OCB

This is the engine rule to parse Alcatel NSS OCB data. This is a one-to-many relationship where a single raw OCB file will generate many PIF files. PIF files will be generated whenever an OCB 'class_id' matches the configured OBSERVATION_RECORDS for the rule.

Note that the header information shall be shared among these records, if there is only one header available for multiple records (it can be multiple sections in the raw file, where each section has its own header information). For OCB, each record has its own vital information in the record header.

2.3 ALCATEL_LM

ALCATEL_LM parses LM data. Each input file will generate multiple PIF files depending on the record type it contains. Similar to ALCATEL_OCB, a PIF file will be generated whenever the LM 'class_id' matches one of the entries in OBSERVATION_RECORDS.

Some of the input files contain records that do not have a record header, for example MNEM record type. Only the first record block has a header. All subsequent record blocks will share the header information in the PIF output files.

2.4 Alcatel Engine Rule Configurations

The following details the Alcatel engine RCP, OCB and LM rule configurations. The configurations are divided up into entries common to all rules and individual rule specific entries.

Entries such as HEADER_MAPPINGS and RECORD_MAPPINGS are common to all rules, but the configured value structures are different, and so these entries are documented in both the Alcatel RCP, Alcatel OCB and Alcatel LM sections.

2.4.1 Common Rule Entries

The following details proprietary rule entries common to all ALCATEL_RCP, ALCATEL_OCB and ALCATEL_LM rules.

- HEADER_FIELDS_FOR_PIF_NAME: This is a mandatory entry with an array value. The values for the array of header counter names that are contained in the array shall be included in the output PIF filename. The values appear in the filename in accordance with the sequence of counter names configured.

```
HEADER_FIELDS_FOR_PIF_NAME =>
                    [ qw(OMC NODE hd_ticketno StartDate StartTime ) ],
```

- HEADER_FIELDS_FOR_UNIQUE_PIF: This is a mandatory entry containing an array value. The header counter values for these counter name entries are used to create a key for the data records. This key serves the purpose of identifying

uniqueness of each of the PIF records. If the key is always unique, then all raw data shall be processed and outputted. If it is duplicated, only the final record in the duplication will be retained.

```
HEADER_FIELDS_FOR_UNIQUE_PIF =>
                    [ qw(OMC NODE hd_ticketno StartDate StartTime ) ],
```

- PIF_OBJ_ID: A mandatory entry containing an array value. The counter names specified here are used to configure what counter names in the current data record contain the observation record family object ID. This object ID is then used to search the configured group record ID's configured in the OBSERVATION_RECORDS entry to see if a match is found. IF a match is found the group name is used in the data output. If not the record is discarded.

```
(RCP) PIF_OBJ_ID => ['COUNTER_FID']
(OCB) PIF_OBJ_ID => ['class_id'],
(LM)  PIF_OBJ_ID => ['class_id'],
```

- PIF_RECORD_ID: A mandatory entry containing an array value. The counter names specified here are used to configure what counter values in the current data record are used as the current PIF record key.

```
(RCP) PIF_RECORD_ID => ['rank']

(OCB) PIF_RECORD_ID =>
['class_id', 'rec_num', 'routeid', 'rank', 'intrecno', 'index'],

(LM)  PIF_RECORD_ID => ['class_id'],
```

- ENABLE_FILENAME_TIMESTAMP: An optional entry with a 'True' or 'False/0' value (True – to turn it on and False/0 to turn it off). This entry allows the pif filename to be appended with a binary file processing timestamp. Enabling this entry will ensure the uniqueness of the pif filename during pif creation.

2.4.2 Alcatel RCP Rule Configuration

The section details rule entries specific to the Alcatel RCP rule.

- OBSERVATION_RECORDS: A mandatory entry containing a hash value. Each hash key entry is the name of a family of objects ID's. Each object family name key points to another hash, which contains the ID's, either as discrete numbers, or regular expression number ranges. If the record COUNTER_FID (record object ID) in each data record does not match any of the values specified in the OBSERVATION_RECORDS COUNTER_GROUPS, it shall be ignored.

```
OBSERVATION_RECORDS => {
             VLR_SS => {
                 COUNTER_GROUPS => ['2300[1,2]'],
             },
```

```
                VLR_E212 => {
                    COUNTER_GROUPS => ['20454'],
                },
                VLR_E164 => {
                    COUNTER_GROUPS => ['20606'],
                },
                VLR_GLOBAL => {
                    COUNTER_GROUPS => ['2030[1-8]',
                                       '20402',
                                       '2040[4-9]',
                },
}
```

- HEADER_MAPPING: This is a mandatory entry containing an array of hashes. The array will contains a sequence of entries, each of which represents a byte (or bytes) mapping for a field of the header record. Each hash in the array shall contain a mandatory key called COUNTER_TYPE. The value of COUNTER_TYPE decides the byte(s) description and byte(s) allocation. The codes used to describe the byte data are those defined in the Perl template for the pack/unpack function. These codes determine how the byte(s) are translated, or unpacked. For example, "C" represents an unsigned character, whereas "N" represents a 4 byte big-endian integer. There are two optional keys in the hash entry. They are ATTR and COUNTER_NAME.

  - ATTR is used to describe the function of the data, and this is currently only used to describe if the byte is part of a date definition. The ALCATEL_RCP collects all "Date" bytes and further processes into a suitable date format.

  - If the COUNTER_NAME is included in the hash entry for each byte(s) allocation, then the entity will be outputted as a full counter name/value in the PIF record. The absence of the COUNTER_NAME suggests that the byte(s) entity will be further processed elsewhere, or else is of no interest and is ignored.

```
HEADER_MAPPING => [
        { COUNTER_TYPE => 'C', ATTR => "Date" }, # year
        { COUNTER_TYPE => 'C', ATTR => "Date" }, # month
        { COUNTER_TYPE => 'C', ATTR => "Date" }, # day
        { COUNTER_TYPE => 'C', ATTR => "Date" }, # hour
        { COUNTER_TYPE => 'C', ATTR => "Date" }, # min
        { COUNTER_TYPE => 'C', ATTR => "Date" }, # sec
        { COUNTER_TYPE => 'C', COUNTER_NAME => 'hd_disp' },
        { COUNTER_TYPE => 'C', COUNTER_NAME => 'hd_noofticket'},
        { COUNTER_TYPE => 'C', COUNTER_NAME => 'hd_ticketno' },
    ],
```

- RECORD_MAPPING: This is a mandatory entry, containing an array of hashes, and works in the same way as header mapping for records in terms of a sequence of byte(s) allocation using COUNTER_TYPE, and optionally COUNTER_NAME to describe a complete counter name/value pair. RECORD_MAPPING includes an extra feature for COUNTER_TYPE, as it allows for dynamic allocation of bytes for a single entity in the RECORD_MAPPING array

sequence, based on counter values. This is done by configuring the COUNTER_TYPE value as a hash entry with the following entries:

- Default: This hash entry describes the default mapping of byte(s) for the current hash entry in terms of counter type mapping and counter name, and is used for initial sizing of the record when parsing the binary data.

- Index: The value of Index contains the parsed record array subscript, whose value contains the integer that will determine the current entities dynamic counter type and byte allocation. This integer must exist in the COUNTER_TYPE hash for a successful mapping.

- 1,2,3,4 etc.: These hash entries describe the mapping of byte(s) for the current RECORD_MAPPING hash entry, which the counter value retrieved by Index, will output, in terms of counter type mapping and counter name, and is used for the final sizing of the record when parsing the binary data.

This dynamic mapping is designed to handle the fact that the rank counter, as described in table 2, is either a 2 byte or a 4 byte integer, depending on the value of the counter type record field.

```
RECORD_MAPPING => [
    { COUNTER_TYPE => 'S', COUNTER_NAME =>'COUNTER_FID' },
    { COUNTER_TYPE => 'C' },
    { COUNTER_TYPE => {
        Index => 1,
        Default => {COUNTER_TYPE =>'H16',COUNTER_NAME =>'rank'},
        1 => {COUNTER_TYPE =>'H16',COUNTER_NAME =>'rank'},
        2 => {COUNTER_TYPE =>'H16',COUNTER_NAME =>'rank'},
        3 => {COUNTER_TYPE =>'H32',COUNTER_NAME =>'rank'},
        4 => {COUNTER_TYPE =>'H32',COUNTER_NAME =>'rank'},
        },
    },
    { COUNTER_TYPE => 'N' },
    { COUNTER_TYPE => 'C' },
]
```

- HEADER_RECORD_PROCESSING: An optional entry whose value points to a configured Perl subroutine. The purpose of this entry is to add custom processing for a deployment, such as add a new counter. But it could be any processing on the header record. This entry only deals with counter name/value pairs in the current header record and is invoked for each header record encountered.

  The ALCATEL_RCP default rule configuration uses the HEADER_RECORD_PROCESSING to perform the following tasks on the header record:
  - Hard code the duration time in seconds for the measurement period.
  - Insert the FILE_START_DATE counter, if not already present.
  - Insert the FILE_START_TIME counter, if not already present.

```
HEADER_RECORD_PROCESSING => sub {
   my ($rec_ref, $op_hash_ref) = @_;
   my @time = split (/:/, $op_hash_ref->{StartTime} );
```

```
    if (!exists($op_hash_ref->{FILE_START_DATE})) {
        $op_hash_ref->{FILE_START_DATE} =
                    $op_hash_ref->{StartDate};
        $op_hash_ref->{FILE_START_TIME} = $time[0].":00";
    }
    $op_hash_ref->{Duration} = 60;
    return 0;
},
```

- DATA_RECORD_PROCESSING: An optional entry whose value points to a configured Perl subroutine. The purpose of this entry is to add custom processing for a deployment, such as add a new counter. But it could be any processing on the data record. This entry only deals with counter name/value pairs in the current data record and is invoked for each data record.

  The ALCATEL_RCP default rule configuration uses the DATA_RECORD_PROCESSING to perform the following tasks on each data record:

  - Creates a new counter name by concatenating "C" and the record FID_COUNTER field value.
  - Assign the new counter name to the counter value contained in the record.
  - Creates a new counter name by concatenating "VAL_C" and the record counter, validation field value.
  - Assign the new counter name to the validation counter field value contained in the record.

```
DATA_RECORD_PROCESSING => sub {
      my ($rec_ref, $op_hash_ref) = @_;
      $op_hash_ref->{"C$rec_ref->[0]"} =  $rec_ref->[3];
      $op_hash_ref->{"VAL_C".$rec_ref->[0]} = $rec_ref->[4];
      return 0;
},
```

- HEADER_DATA_TO_REMOVE: An optional entry whose value points to a configured Perl subroutine. The purpose of this entry is to add custom processing for a deployment such as stripping off extra header. FIELD_TERMINATOR is used to identify the field to be stripped off.

```
HEADER_DATA_TO_REMOVE => [
      { FIELD_NAME => 'ERROR_CODE', FIELD_TERMINATOR => "\n", },
      { FIELD_NAME => 'SYSTEM_NAME', FIELD_TERMINATOR => "\n", },
      { FIELD_NAME => 'LENGTH', FIELD_TERMINATOR => "\n",
                                RECURRING  => 'True',
                                BLOCK_LENGTH => 'True',
      },
],
```

2.4.3 Alcatel OCB Rule Configuration

The section details rule entries specific to the Alcatel OCB rule.

- OBSERVATION_RECORDS: A mandatory entry containing a hash value. This has the same role as the ALCATEL_RCP OBSERVATION_RECORDS, but with an added feature. Erroneous raw data block can be detected earlier by using two optional

entries, HEADER_LENGTH, and RECORD_LENGTH. When a block of raw data is read, it should contain the length of the block. To check whether this block is correct, the formula below should give a result of 0:

(raw_data_block_length – HEADER_LENGTH) mod RECORD_LENGTH

Related to CONSOLIDATE_FILES, an optional entry RECORD_GROUP is introduced here. This entry notifies the engine as to how many records should be expected for a certain record type.

```
OBSERVATION_RECORDS => {
      PERM_OBS_CLASS_R => { COUNTER_GROUPS => ['7'],
            HEADER_LENGTH  => 25,
            RECORD_LENGTH  => 16,
      },
      PERM_OBS_CLASS_T => { COUNTER_GROUPS => ['8'],
            HEADER_LENGTH  => 13,
            RECORD_LENGTH  => 16,
            RECORD_GROUP   => 64,
      },
}
```

- HEADER_MAPPING: This is a mandatory entry, containing an array of hashes, and works in the same way as header mapping in the ALCATEL_RCP rule in terms of a sequence of byte(s) allocation using COUNTER_TYPE, and optionally COUNTER_NAME to describe a complete counter name/value pair. ALCATEL_OCB also includes dynamic byte allocation for the header record, based on the value of a particular counter in the current header, which is used to index into the HEADER_MAPPING COUNTER_TYPE hash to fetch the correct byte mapping for the current header record. This is done by configuring the COUNTER_TYPE value as a hash entry with the following entries:

    - Default: This hash entry describes the default mapping of byte(s) for the current hash entry in terms of counter type mapping and counter name, and is used for initial sizing of the record when parsing the binary data.

    - Index: The value of Index contains the parsed record array subscript entry, whose value contains the integer that will determine the current entities dynamic counter type and byte allocation. This integer must exist in the COUNTER_TYPE hash keys for a successful mapping.
    - 1,4,5,6,7 etc.: The numeric record codes contained in the current header record are actually OCB class ID's and are configured here as hash keys. The corresponding hash values are arrays of hashes, reflecting the variable nature of OCB headers. Some of these mappings will be single entity hash mappings, where others will have multiple hash entities in the array.

```
HEADER_MAPPING => [
    { COUNTER_TYPE => 'n', RECORD_LENGTH => 'True', },
    { COUNTER_TYPE => 'n' },
    { COUNTER_TYPE => 'C', COUNTER_NAME => 'record_code' },
    { COUNTER_TYPE => 'n', ATTR => 'Date' }, # year day
    { COUNTER_TYPE => 'C', ATTR => 'Date' }, # hour
```

```
        { COUNTER_TYPE => 'C', ATTR => 'Date' }, # minute
        { COUNTER_TYPE => 'C', ATTR => 'Date' }, # second
        # Permanent Observation Measurement Mappings (OBS)
        { COUNTER_TYPE => {
            Default => { COUNTER_TYPE =>'C' },
            Index => 2,
            # PERM_OBS_START
            1 => [{ COUNTER_TYPE => 'C',COUNTER_NAME => 'xchg_num'},],

            # PERM_OBS_CLASS_A
            6 => [  {COUNTER_TYPE => 'C',COUNTER_NAME => 'xchg_num' },
                    { COUNTER_TYPE => 'n' },
                    { COUNTER_TYPE => 'n' },
                    { COUNTER_TYPE => 'n' },
                 ],

            # PERM_OBS_CLASS_C
            4 => [{ COUNTER_TYPE => 'C',COUNTER_NAME => 'xchg_num'},],
```

- RECORD_MAPPING: This is a mandatory array entry. Similar to ALCATEL_RCP but with an extra feature. A new key named COUNTER_OPERATION is introduced. This entry will tell the engine to generate a sequence number for each record of a particular record type. However, this entry only accepts two values, i.e. '+' and '++'. When '+' is used, the sequence number shall be reset back to 1 whenever a single raw data block is processed, whereas '++' is reset after the raw file is processed.

```
RECORD_MAPPING=> [
    { COUNTER_TYPE =>
        {
            Default => { COUNTER_TYPE =>'C' },
            33 => [
                  { COUNTER_TYPE => 'C', COUNTER_NAME => 'rank'},
                  { COUNTER_TYPE => 'C' },
                  { COUNTER_TYPE => 'n' },
                  { COUNTER_TYPE => 'N', COUNTER_NAME => 'ine_i'},
                  { COUNTER_TYPE => 'N', COUNTER_NAME => 'inr_i'},
                  { COUNTER_OPERATION => '+',
                    COUNTER_NAME => 'intrecno' },
                 ],
            8 =>  [
                  { COUNTER_TYPE => 'N', COUNTER_NAME => 'ta' },
                  { COUNTER_TYPE => 'N', COUNTER_NAME => 'tt' },
                  { COUNTER_TYPE => 'N', COUNTER_NAME => 'td' },
                  { COUNTER_TYPE => 'N', COUNTER_NAME => 'ti' },
                  { COUNTER_OPERATION => '++',
                    COUNTER_NAME => 'index' },
                 ],
            },
      },
]
```

- PARTIAL_FILE_DIR: This is an optional entry containing a directory entry. In some cases the records for a group are not within a single raw file. When this

happens, the partial PIF cannot be processed to LIF yet. It is being stored in the directory path specified by this entry.

```
PARTIAL_FILE_DIR => '../../partial',
```

- CONSOLIDATE_FILES: This is an optional scalar entry. As mentioned above, when a partial PIF is generated, and this entry is set to 'true', the engine shall go to match the other portion of PIF in PARTIAL_FILE_DIR. If there are matching partial PIF, they will be joined together and consolidates counters if necessary.

```
CONSOLIDATE_FILES => 'True'
```

- INDEX_HOLDER: This is an optional array entry. When COSOLIDATE_FILES is set to 'true', there is an option to consolidate any sequence number within the PIFs. This array holds all possible counter names contain sequence number.

```
INDEX_HOLDER => [ qw( rank rec_num intrecno index) ]
```

- CHARACTER_MAP: This is an optional entry. This is used to map the character in the raw file to a specific character map, i.e. from EBCDIC to ASCII. It points to an array of characters.

```
CHARACTER_MAP => char_map()
```

2.4.4 Alcatel LM Rule Configuration

The section details rule entries specific to the Alcatel LM rule.

Mandatory Entries

- OBSERVATION_RECORDS: A mandatory entry containing a hash value. This entry is similar to Alcatel RCP. The extra configuration for Alcatel LM is the UNIQUE_HEADER_KEY. Some of the record types have different sets of header/record structure, depending on the indicator in the raw data itself. This has to be determined at runtime. This optional entry (UNIQUE_HEADER_KEY) tells the engine where to find the value of the indicator. The indicator tells which header/record structure to take from the configuration. Below shows the example of the MTP record type. Note that the counter 'repetitions' is defined in the HEADER_MAPPING for the MTP record type.

```
OBSERVATION_RECORDS => {
    N7T1 => {
        COUNTER_GROUPS => ['1565'],
        UNIQUE_HEADER_KEY => 'repetitions',
    },
    N7T2 => {
        COUNTER_GROUPS => ['1566'],
        UNIQUE_HEADER_KEY => 'repetitions',
    },
    N7T3 => {
        COUNTER_GROUPS => ['1567'],
        UNIQUE_HEADER_KEY => 'repetitions',
    },
    N7T4a => {
        COUNTER_GROUPS => ['1568'],
```

```
            UNIQUE_HEADER_KEY => 'repetitions',
        },
        N7T4b => {
            COUNTER_GROUPS => ['1589'],
        },
        N7T5 => {
            COUNTER_GROUPS => ['1569'],
        },
        N7T6 => {
            COUNTER_GROUPS => ['1570'],
        },
    },
```

- HEADER_MAPPING: This is a mandatory entry containing an array of hashes. The usage is exactly the same as for the Alcatel OCB rule. Below shows the example mapping for the CPU record type:

```
HEADER_MAPPING => [
    { COUNTER_TYPE => 'v', RECORD_LENGTH => 'True' }, # rec len
    { COUNTER_TYPE => 'v', COUNTER_NAME => 'record_type' },
    { COUNTER_TYPE => {
        Index => 1,
        Default => {COUNTER_TYPE => 'C'},

        1560 => [ # CPU
            { COUNTER_TYPE => 'a16', }, # unused
            { COUNTER_TYPE => 'v', ATTR => "Date",}, #year
            { COUNTER_TYPE => 'C', ATTR => "Date" }, # month
            { COUNTER_TYPE => 'C', ATTR => "Date" }, # day
            { COUNTER_TYPE => 'C', ATTR => "Date" }, # hour
            { COUNTER_TYPE => 'C', ATTR => "Date" }, # min
            { COUNTER_TYPE => 'C', COUNTER_NAME => "end_hour" },
            { COUNTER_TYPE => 'C', COUNTER_NAME => "end_min" },
            { COUNTER_TYPE => 'a6',  }, # unused
        ],
      },
    },
],
```

Below shows the example mapping for the TRUNKGROUP record type for Flexible ADL. Gateway will do counter name mapping from counter id given in the data file.

```
            7403 => [ # TRUNKGROUP
                { COUNTER_TYPE => 'a12', }, # unused
                { COUNTER_TYPE => 'v', ATTR => "Date",}, #year
                { COUNTER_TYPE => 'C', ATTR => "Date" }, # month
                { COUNTER_TYPE => 'C', ATTR => "Date" }, # day
                { COUNTER_TYPE => 'a4', }, # unused
                { COUNTER_TYPE => 'C', ATTR => "Date" }, # hour
                { COUNTER_TYPE => 'C', ATTR => "Date" }, # min
                { COUNTER_TYPE => 'C', COUNTER_NAME => "end_hour" },
                { COUNTER_TYPE => 'C', COUNTER_NAME => "end_min" },
                { COUNTER_TYPE => 'a2',  }, # unused
                { COUNTER_TYPE => 'C', COUNTER_NAME => 'ind'  },
                { COUNTER_TYPE => 'a5' }, # unused
```

```
                { COUNTER_TYPE => 'v', NAME_REF => 'COUNTERNAME1' },
                { COUNTER_TYPE => 'v', NAME_REF => 'COUNTERNAME2' },
                { COUNTER_TYPE => 'v', NAME_REF => 'COUNTERNAME3' },
                { COUNTER_TYPE => 'v', NAME_REF => 'COUNTERNAME4' },
                { COUNTER_TYPE => 'v', NAME_REF => 'COUNTERNAME5' },
                { COUNTER_TYPE => 'v', NAME_REF => 'COUNTERNAME6' },
                { COUNTER_TYPE => 'v', NAME_REF => 'COUNTERNAME7' },
                { COUNTER_TYPE => 'v', NAME_REF => 'COUNTERNAME8' },
                { COUNTER_TYPE => 'v', NAME_REF => 'COUNTERNAME9' },
                { COUNTER_TYPE => 'v', NAME_REF => 'COUNTERNAME10'},
                { COUNTER_TYPE => 'a40',  }, # unused
            ],
```

- RECORD_MAPPING: This is a mandatory entry containing an array of hashes. The usage is similar to HEADER_MAPPING described above, but it is for data record field mappings. Below shows the example mapping for the CPU record type:

```
RECORD_MAPPING => [
    { COUNTER_TYPE => {
        Default => {COUNTER_TYPE =>'C'},

        1560 => [ # CPU
            { COUNTER_TYPE =>'a2', }, # unused
            { COUNTER_TYPE =>'v', COUNTER_NAME => 'CE_ID' },
            { COUNTER_TYPE =>'V', COUNTER_NAME => 'INTERRUPTS' },
            { COUNTER_TYPE =>'V', COUNTER_NAME => 'EVENTS' },
            { COUNTER_TYPE =>'V', COUNTER_NAME => 'PROC_SE' },
            { COUNTER_TYPE =>'V', COUNTER_NAME => 'IDLE' },
            { COUNTER_TYPE =>'v', COUNTER_NAME => 'CPU_LOAD' },
        ],
      },
    },
],
```

  Below shows the example mapping for the TRUNKGROUP record type for Flexible ADL. Gateway will do counter name mapping from counter id given in the data file.

```
        `7403_IN' => [ # TRUNKGROUP
            { COUNTER_TYPE =>'v', COUNTER_NAME=> 'OBJECTINDEX' },
            { COUNTER_TYPE => 'a2' },
            { COUNTER_TYPE =>'N', NAME_REF => 'COUNTERVALUE1' },
            { COUNTER_TYPE =>'N', NAME_REF => 'COUNTERVALUE2' },
            { COUNTER_TYPE =>'N', NAME_REF => 'COUNTERVALUE3' },
            { COUNTER_TYPE =>'N', NAME_REF => 'COUNTERVALUE4' },
            { COUNTER_TYPE =>'N', NAME_REF => 'COUNTERVALUE5' },
            { COUNTER_TYPE =>'N', NAME_REF => 'COUNTERVALUE6' },
            { COUNTER_TYPE =>'N', NAME_REF => 'COUNTERVALUE7' },
            { COUNTER_TYPE =>'N', NAME_REF => 'COUNTERVALUE8' },
            { COUNTER_TYPE =>'N', NAME_REF => 'COUNTERVALUE9' },
            { COUNTER_TYPE =>'N', NAME_REF => 'COUNTERVALUE10' },
        ],
```

- FILE_BLOCK_SIZE: this is a scalar entry. It tells the engine the size of a single file block (refer Diagram 1) in bytes. The vendor engine will read the amount of

bytes from the raw file specified by this entry for each iteration, until the whole raw file is processed.

```
FILE_BLOCK_SIZE => 2048,
```

Optional Entries

- SKIP_UNKNOWN_RECORD: This is a scalar entry. It is an entry that decide whether to skip the unknown block when the parser already reach at the end of the record as describe in the record header. The calculation on the number of bytes to skip is based on block data size and current record length. The trigger for the skipping is based on errors when parsing the next record block. Without this entry or setting it to false, the parser will assume that the raw data file is corrupted.

  ```
  SKIP_UNKNOWN_RECORD => 'TRUE',
  ```

- RECORD_BLOCK_LENGTH_BYTE_SIZE: This is a scalar entry. In 'overflow' cases, where a single 2KB file block is not sufficient to contain all data records, these data records will overflow to the next file block. The first record block in the next file block will not contain any record header, but only record length information (the length of the overflowed data record). The engine needs to skip these bytes in order to continue reading the data record. This entry tells the engine how many bytes to skip for the length information. It is defaulted to 2 bytes if this entry is not set.

  ```
  RECORD_BLOCK_LENGTH_BYTE_SIZE => 2,
  ```

- ADD_RUNNING_SEQUENCE: This is a scalar entry. Most of the LM data records do not contain any unique key. This entry, if set to true, will result in a counter name called 'OBJECT', with a numeric value incremented for each record, in the output PIF data records.

  ```
  ADD_RUNNING_SEQUENCE => 'True',
  ```

- CONFIG_FILE: This is a hash entry. An external file is required when the indicator of a dynamic header/record structure is not within the raw file. These indicators are stored within an external file. The Alcatel LM rule provides this facility where the data (content of external file) is read into its internal data structures. This information will be read at the beginning of the rule execution and updated back to the file at the end of the execution. Any new entry added during execution shall be added to these external files. The key of this hash is string containing the directory path and filename of the external configuration file.

  Within the hash, DELIMITER and KEY_INDEX are optional. If they do not exist, the engine defaults the value of DELIMITER to ' ' (space) and the KEY_INDEX to 0 (first index of array). Note that the DELIMITER value is a Perl regular expression.

  COLUMN_NAMES is a mandatory array entry. It lists out the sequence of column names in the external configuration file. Note that there is always a 'key' in this array. The index of its position must match with KEY_INDEX (or defaults at index 0).

```
CONFIG_FILE => {
    './GSconfigfile' => {
        DELIMITER => ',',
        KEY_INDEX => 0,
        COLUMN_NAMES => ['key','num_objs','is_nsta'],
    },
    './attrfile' => {
        DELIMITER => '\s+',
        KEY_INDEX => 0,
        COLUMN_NAMES => ['key','name','group'],
    },
},
```

- COMMON_HEADER_COUNTERS: This is an array entry. As some of the record header types (MNEM file type) do not have date time information, it needs to take that information from another record header, i.e. only the very first record header has the information. This entry tells the engine to copy these common header counters to the remaining record headers that require it.

```
COMMON_HEADER_COUNTERS => [ qw(StartDate StartTime) ],
```

- HEADER_RECORD_PROCESSING: This is the same entry as described in the Alcatel RCP rule. The only difference for Alcatel LM is that there are three parameters passed into this subroutine instead of the two in the Alcatel RCP rule. The added parameter is the reference to the object parent. This gives the subroutine access to some of the information needed from the engine code for processing.

```
HEADER_RECORD_PROCESSING => sub {
    my ($parent_ref, $rec_ref, $op_hash_ref) = @_;

    # format end time
    if ( exists($op_hash_ref->{end_hour}) &&
         exists($op_hash_ref->{end_min}) ) {
        $op_hash_ref->{'end_time'} =
            sprintf("%02d:%02d",
                    $op_hash_ref->{end_hour},
                    $op_hash_ref->{end_min});
        delete $op_hash_ref->{end_hour};
        delete $op_hash_ref->{end_min};
    }

    # Handover (4684), header byte 13 to determine blocks
    if ( $op_hash_ref->{class_id} eq '4684' ) {
        $parent_ref->{HOD_Ind} = $op_hash_ref->{indicator};
        $parent_ref->{HOD_cnt1} = $op_hash_ref->{cnt1};
        $parent_ref->{HOD_rec_count} = 0;
    }

    return 0;
},
```

- DATA_RECORD_PROCESSING: This is same as described above, but it is for the data record part. The example subroutine is as below:

```
DATA_RECORD_PROCESSING => sub {
    my ($parent_ref, $rec_ref, $op_hash_ref) = @_;

    # change record block when cnt1 is hit
    if ($parent_ref->{HOD_rec_count}==$parent_ref->{HOD_cnt1}) {
        if ( $parent_ref->{current_class_id} eq '4684' ) {
            if ( $parent_ref->{HOD_Ind} == 0 ||
                 $parent_ref->{HOD_Ind} == 1 ) {
                $parent_ref->{current_class_id} = '4684_block';
            }
            else {
                $parent_ref->{current_class_id} = '4684_notused';
            }
        }
    }

    # increase record count
    if (exists($parent_ref->{HOD_rec_count})) {
        $parent_ref->{HOD_rec_count}++;
    }

    return 0;
},
```

3. Post Parser Rules and Configuration

In order to produce the LIF that matches the requirement for SSP/OCB, two vendor specific, post parser rules are created. They are HEADER_2_RECORD_TRANSFER and ROTATE.

3.1 HEADER_2_RECORD_TRANSFER

This rule transfers the selective values of header/record from an existing PIF to a new PIF. Maximum value of a counter is populated into the new PIF if specify. Hard coded value is also made available by specifying it in the rule.

3.1.1 Rule Configuration

- FILENAME_PREFIX: This is an optional scalar entry. When this is specified, it will become the prefix of the new PIF filename.

```
FILENAME_PREFIX => 'MAX'
```

- OUTPUT_BLOCK_NAME: This is an optional scalar entry. If this is not specified, the block name of the existing PIF shall be used in the new PIF.

```
OUTPUT_BLOCK_NAME => 'PERM_OBS_CLASS_R_MAX'
```

- HEADER_DATA_FOR_TRANSFER: This is a mandatory array entry. This array contains the header counter name of the existing PIF. Values of these counters shall be copied to the new PIF's record part.

```
HEADER_DATA_FOR_TRANSFER => []
```

- RECORD_DATA_FOR_TRANSFER: This is a mandatory array entry. This array contains the record counter name of the existing PIF. Values of these counters shall be copied to the new PIF's record part.

```
RECORD_DATA_FOR_TRANSFER => ['rpa_i', 'rra_i', 'rec_num','NODEID']
```

- MAX_SEQUENCE_COUNTER: This is an optional scalar entry. This is the counter name in the existing PIF that holds the running sequence number. When this is specified, all other records in the existing PIF are ignored except the record with maximum running number. This will only happens if SET_MAX_COUNTER is specified.

```
MAX_SEQUENCE_COUNTER => 'rec_num'
```

- SET_MAX_COUNTER: This is an optional hash entry. This hash contains the new counter names (to new PIF) and the existing counter name (from existing PIF). If the value specify in this hash cannot be found (i.e. no such counter name in existing PIF), it shall be treated as hard coded value.

```
SET_MAX_COUNTERS => { rpx => 'rpa_i',
                      rrx => 'rra_i',
                      NODEID => 'NODEID',
```

```
                    REFER_TO => 'PERM_OBS_CLASS_R',
                  },
```

A full sample configuration is included below:

```
{
    RULE_TYPE                => 'HEADER_2_RECORD_TRANSFER',
    RULE_DESC                => 'Get summary info for CLASS_R',
    INPUT_FILE_DESCRIPTION
                         => '^(PERM_OBS_CLASS_R-#-.*-#-I)\.pif',
    PRODUCE_PIF              => 'True',
    FILENAME_PREFIX          => 'MAX',
    OUTPUT_BLOCK_NAME        => 'PERM_OBS_CLASS_R_MAX',
    HEADER_DATA_FOR_TRANSFER => [],
    RECORD_DATA_FOR_TRANSFER =>
                          ['rpa_i', 'rra_i', 'rec_num','NODEID'],
    MAX_SEQUENCE_COUNTER     => 'rec_num',
    SET_MAX_COUNTERS         => { rpx => 'rpa_i',
                                  rrx => 'rra_i',
                                  NODEID => 'NODEID',
                                  REFER_TO => 'PERM_OBS_CLASS_R',
                                },
}
```

3.1.2 Sample Usage

Given the following PIF:

```
## Parser Intermediate File
##START|HEADER
OMC|record_code|StartTime|class_id|NODE|Date|StartDate|xchg_num
760OBTS0|7|22:00|7|SSP01|13722025|2004May16|1
##END|HEADER
##START|PERM_OBS_CLASS_R
NODEID|OBJ_GROUP_NAME|rpa_i|rrd_i|class_id|rra_i|rpd_i|rec_num
SSP01|PERM_OBS_CLASS_R|109|0|7|0|3304192244|1
SSP01|PERM_OBS_CLASS_R|0|0|7|3253860597|0|2
SSP01|PERM_OBS_CLASS_R|3304192245|3879747648|7|121|0|3
SSP01|PERM_OBS_CLASS_R|550172|0|7|0|0|4
##END|PERM_OBS_CLASS_R
```

Using the sample configuration, the outputted data is as below:

```
## Parser Intermediate File
##START|HEADER
OMC|record_code|StartTime|class_id|NODE|Date|StartDate|xchg_num
760OBTS0|7|22:00|7|SSP01|13722025|2004May16|1
##END|HEADER
##START|PERM_OBS_CLASS_R_MAX
max_seq|rpx|NODEID|rrx|REFER_TO
4|550172|SSP01|0|PERM_OBS_CLASS_R
##END|PERM_OBS_CLASS_R_MAX
```

3.2 TRANSPOSE

The objective of this rule is to transpose the data matrix by exchanging the rows from PIF data block to columns. The counter names for the header and value will be defined in the configuration.

3.2.1 Rule Configuration

- NEW_NAME_FOR_COUNTER_NAMES: This is a mandatory scalar entry. This shall be the counter name that will be use to group counter names which will come in rows.

```
NEW_NAME_FOR_COUNTER_NAMES  => 'HEADERS',
```

- COUNTER_NAME_FOR_DATA_ROWS: This is a mandatory scalar entry. This shall be the counter name that will be use to group all the counter values which map according its counter name.

```
COUNTER_NAME_FOR_DATA_ROWS => 'VALUE',
```

- COLUMN_DATA_TO_REPLACE_CTRS: This is an optional scalar entry. The value/data for the named counter will be promote as a counter name to group all the counter value.

```
COLUMN_DATA_TO_REPLACE_CTRS => 'OBJ_GROUP_NAME',
```

- REDUNDANT_COLUMNS: This is an optional array entry. The named columns will be removed from the data block before transposing the data matrix. The features similar to REDUNDANT_DATA_COUNTERS entry in other post parser rule.

```
REDUNDANT_COLUMNS => 'CA_OR',
```

A full sample configuration is included below:

```
      {
        RULE_TYPE => 'TRANSPOSE',
        RULE_DESC => 'Transpose the ADL result to produce counter/value
                      pair',
        INPUT_FILE_DESCRIPTION      => [
            '^\w+-#-\w+_A900_ADL-#-\w+-#-\d{2}\w{3}\d{4}-#-\d{2}:\d{2}-
#-\d{2}:\d{2}-#-\d+-#-I.pif',
                                    ],
        NEW_NAME_FOR_COUNTER_NAMES  => 'HEADERS',
        COUNTER_NAME_FOR_DATA_ROWS  => 'VALUE',
        COLUMN_DATA_TO_REPLACE_CTRS => 'OBJ_GROUP_NAME',
        REDUNDANT_COLUMNS           => ['CA_OR'],
        PRODUCE_PIF                 => 'True',
        PRODUCE_LIF                 =>  0,
        OUTPUT_FORMAT               =>  0,
      },
```

3.2.2 Sample Usage

Given the following PIF record (there are 20 columns):

```
## Parser Intermediate File
##START|HEADER
StartTime|record_type|MSC_AREA|end_time|class_id|TYPE|StartDate|du
ration|MSCID
23:00|8362|ASTRAKHAN|00:00|8362|VFS_A900_ADL|28Feb2007|60|ASTRAKHA
N
##END|HEADER
##START|STUNDLICHE
ORIGIN_TERM_TRF|OBJ_GROUP_NAME|SEQUENCE|ORIGINATING_OUTGOING_TRF|I
NCOMING_TERM|CC_OG|CALL_ATTEMPTS|class_id|MOB_MOB_TRF|CA_MORTR_MSC
|INCOMING_TERM_MOBILE_TRF|CA_OR|CA_MICTR_MSC|INCOMING_OUTGOING_TRF
|CA_TERM|ORIG_TRF_MOBILE_TERM_TRF|MS_OUTGOING_TRF|BEHAVIOUR_INDICA
TOR|PA_START|MOBILE_TERM_TRF
0|STUNDLICHE|0|0|0|55608|34654|8362|36630|52826|32026|0|19232|4301
2|0|0|52759|83|44078|0
##END|STUNDLICHE
```

Using the sample configuration, new PIFs is generated according to the configuration.

```
## Parser Intermediate File
##START|HEADER
end_time|MSC_AREA|record_type|duration|MSCID|StartTime|class_id|TY
PE|StartDate
00:00|ASTRAKHAN|8362|60|ASTRAKHAN|23:00|8362|VFS_A900_ADL|28Feb200
7
##END|HEADER
##START|STUNDLICHE
HEADERS|STUNDLICHE
ORIGIN_TERM_TRF|0
SEQUENCE|0
ORIGINATING_OUTGOING_TRF|0
INCOMING_TERM|0
INCOMING_TERM_MOBILE_TRF|32026
CC_OG|55608
CA_OR|0
CALL_ATTEMPTS|34654
CA_MICTR_MSC|19232
class_id|8362
INCOMING_OUTGOING_TRF|43012
MOB_MOB_TRF|36630
CA_TERM|0
CA_MORTR_MSC|52826
ORIG_TRF_MOBILE_TERM_TRF|0
MS_OUTGOING_TRF|52759
BEHAVIOUR_INDICATOR|83
PA_START|44078
MOBILE_TERM_TRF|0
##END|STUNDLICHE
```

3.3 ROTATE

As the name suggested, this rule turns the row of records of a PIF into columns. The counter names of the new PIF shall be suffix with running number. The FILENAME_PREFIXES shall be populated into the new PIF as 'prefix'.

3.3.1 Rule Configuration

- FILENAME_PREFIXES: This is a mandatory array entry. It consists the prefix of each new PIFs that are going to be generated.

```
FILENAME_PREFIXES => ['ALINCT','ALORGO','ALTRA','ALINT']
```

- COLUMNS_TO_ROTATE: This is a mandatory array entry. This array contains the counter names in the existing PIF that is going to be rotated. Note that the sequence of the counter name must match the one specified in FILENAME_PREFIXES.

```
COLUMNS_TO_ROTATE => ['td',    'tt',    'ti',   'ta'   ]
```

- NEW_COLUMN_PREFIXES: This is a mandatory array entry. This shall be the prefix of the counters in new PIF. Again, the sequence of this array must match the one specified in FILENAME_PREFIXES and COLUMNS_TO_ROTATE.

```
NEW_COLUMN_PREFIXES => ['t',     't',     't',    't'    ]
```

- SEQUENCE: This is an optional scalar entry. This is the counter name in the existing PIF that holds the running number of records. When this is specified, the value of this counter shall be combined with the NEW_COLUMN_PREFIXES to form the complete counter name. If this is not specified, an internal running number is used to make the counter name in new PIF unique.

```
SEQUENCE => 'index',
```

A full sample configuration is included below:

```
{
    RULE_TYPE              => 'ROTATE',
    RULE_DESC         => 'Turn rows of rec in CLASS T to columns',
    INPUT_FILE_DESCRIPTION => '^PERM_OBS_CLASS_T-#-.*-#-I\.pif',
    PRODUCE_PIF            => 'True',
    PRODUCE_LIF            => '0',
    FILENAME_PREFIXES      => ['ALINCT','ALORGO','ALTRA','ALINT'],
    COLUMNS_TO_ROTATE      => ['td',    'tt',    'ti',   'ta'   ],
    NEW_COLUMN_PREFIXES    => ['t',     't',     't',    't'    ],
    SEQUENCE               => 'index',
},
```

3.3.2 Sample Usage

Given the following PIF record (there are 64 rows):

```
## Parser Intermediate File
##START|HEADER
OMC|record_code|StartTime|class_id|NODE|Date|StartDate|xchg_num
760OBTS0|8|22:00|8|SSP01|13722025|2004May16|1
```

```
##END|HEADER
##START|PERM_OBS_CLASS_T
NODEID|td|tt|index|OBJ_GROUP_NAME|ti|class_id|ta
SSP01|0|0|11|PERM_OBS_CLASS_T|0|8|0
SSP01|24744|1473|12|PERM_OBS_CLASS_T|746|8|1
SSP01|17|133|13|PERM_OBS_CLASS_T|2|8|0
SSP01|0|0|30|PERM_OBS_CLASS_T|0|8|0
SSP01|21|223|14|PERM_OBS_CLASS_T|0|8|0
SSP01|0|0|31|PERM_OBS_CLASS_T|0|8|0
SSP01|0|0|15|PERM_OBS_CLASS_T|0|8|1
SSP01|0|0|32|PERM_OBS_CLASS_T|0|8|0
SSP01|361|939|16|PERM_OBS_CLASS_T|3|8|0
SSP01|0|0|33|PERM_OBS_CLASS_T|0|8|0
SSP01|0|0|17|PERM_OBS_CLASS_T|0|8|0
SSP01|0|0|50|PERM_OBS_CLASS_T|0|8|0
SSP01|0|0|34|PERM_OBS_CLASS_T|0|8|0
SSP01|27854|25216|18|PERM_OBS_CLASS_T|12548|8|0
SSP01|0|0|51|PERM_OBS_CLASS_T|0|8|0
SSP01|0|0|35|PERM_OBS_CLASS_T|0|8|0
SSP01|0|0|19|PERM_OBS_CLASS_T|0|8|0
SSP01|0|0|52|PERM_OBS_CLASS_T|0|8|0
SSP01|0|0|36|PERM_OBS_CLASS_T|0|8|0
SSP01|0|0|53|PERM_OBS_CLASS_T|0|8|0
SSP01|0|0|37|PERM_OBS_CLASS_T|0|8|0
SSP01|0|0|54|PERM_OBS_CLASS_T|0|8|0
SSP01|0|0|38|PERM_OBS_CLASS_T|0|8|0
SSP01|0|0|55|PERM_OBS_CLASS_T|0|8|0
SSP01|8|0|39|PERM_OBS_CLASS_T|0|8|0
SSP01|0|0|56|PERM_OBS_CLASS_T|0|8|0
SSP01|0|0|57|PERM_OBS_CLASS_T|0|8|0
SSP01|0|0|58|PERM_OBS_CLASS_T|0|8|0
SSP01|0|0|59|PERM_OBS_CLASS_T|0|8|0
SSP01|52503|44315|1|PERM_OBS_CLASS_T|12381|8|108
SSP01|0|0|2|PERM_OBS_CLASS_T|0|8|0
SSP01|0|0|3|PERM_OBS_CLASS_T|0|8|0
SSP01|89845|82724|4|PERM_OBS_CLASS_T|29446|8|109
SSP01|159228|82724|5|PERM_OBS_CLASS_T|29446|8|110
SSP01|0|82571|6|PERM_OBS_CLASS_T|29446|8|0
SSP01|81035|70500|7|PERM_OBS_CLASS_T|20293|8|110
SSP01|0|0|8|PERM_OBS_CLASS_T|0|8|0
SSP01|0|0|9|PERM_OBS_CLASS_T|0|8|0
SSP01|0|0|20|PERM_OBS_CLASS_T|0|8|0
SSP01|0|0|21|PERM_OBS_CLASS_T|0|8|0
SSP01|6293|6331|22|PERM_OBS_CLASS_T|1736|8|0
SSP01|754|1240|23|PERM_OBS_CLASS_T|582|8|0
SSP01|0|0|40|PERM_OBS_CLASS_T|0|8|0
SSP01|0|11|24|PERM_OBS_CLASS_T|0|8|0
SSP01|0|0|41|PERM_OBS_CLASS_T|0|8|0
SSP01|0|0|25|PERM_OBS_CLASS_T|0|8|0
SSP01|0|0|42|PERM_OBS_CLASS_T|0|8|0
SSP01|0|0|26|PERM_OBS_CLASS_T|0|8|0
SSP01|0|0|43|PERM_OBS_CLASS_T|0|8|0
SSP01|0|0|27|PERM_OBS_CLASS_T|0|8|0
SSP01|0|0|60|PERM_OBS_CLASS_T|0|8|0
SSP01|53|1605|44|PERM_OBS_CLASS_T|1230|8|0
SSP01|0|0|28|PERM_OBS_CLASS_T|0|8|0
SSP01|0|0|61|PERM_OBS_CLASS_T|0|8|0
```

```
SSP01|0|132|45|PERM_OBS_CLASS_T|0|8|0
SSP01|0|0|29|PERM_OBS_CLASS_T|0|8|0
SSP01|0|0|62|PERM_OBS_CLASS_T|0|8|0
SSP01|0|4|46|PERM_OBS_CLASS_T|0|8|0
SSP01|0|0|63|PERM_OBS_CLASS_T|0|8|0
SSP01|46593|912|47|PERM_OBS_CLASS_T|209|8|0
SSP01|33|189|64|PERM_OBS_CLASS_T|7|8|0
SSP01|0|0|48|PERM_OBS_CLASS_T|0|8|0
SSP01|0|0|49|PERM_OBS_CLASS_T|0|8|0
SSP01|0|0|10|PERM_OBS_CLASS_T|0|8|0
##END|PERM_OBS_CLASS_T
```

Using the sample configuration, 4 new PIFs are generated for each specified counter names:

```
## Parser Intermediate File
##START|HEADER
OMC|record_code|StartTime|class_id|NODE|Date|StartDate|xchg_num
760OBTS0|8|22:00|8|SSP01|13722025|2004May16|1
##END|HEADER
##START|PERM_OBS_CLASS_T_ALINCT
t46|t47|t48|t49|t10|t11|t12|t13|t14|t15|t16|t17|t50|t18|t51|t19|pr
efix|t52|t53|t54|t55|t56|t57|t58|t59|t20|t21|t22|t23|t24|t25|t26|t
27|t60|t28|t61|t29|t62|t63|t64|t30|t31|t32|t33|t34|t35|t36|t37|t38
|t39|t01|t02|t03|t04|t05|t06|t07|t08|t40|t09|t41|t42|t43|t44|t45
0|46593|0|0|0|0|24744|17|21|0|361|0|0|27854|0|0|ALINCT|0|0|0|0|0|0
|0|0|0|0|6293|754|0|0|0|0|0|0|0|0|0|33|0|0|0|0|0|0|0|0|8|52503
|0|0|89845|159228|0|81035|0|0|0|0|0|0|53|0
##END|PERM_OBS_CLASS_T_ALINCT

## Parser Intermediate File
##START|HEADER
OMC|record_code|StartTime|class_id|NODE|Date|StartDate|xchg_num
760OBTS0|8|22:00|8|SSP01|13722025|2004May16|1
##END|HEADER
##START|PERM_OBS_CLASS_T_ALORGO
t46|t47|t48|t49|t10|t11|t12|t13|t14|t15|t16|t17|t50|t18|t51|t19|pr
efix|t52|t53|t54|t55|t56|t57|t58|t59|t20|t21|t22|t23|t24|t25|t26|t
27|t60|t28|t61|t29|t62|t63|t64|t30|t31|t32|t33|t34|t35|t36|t37|t38
|t39|t01|t02|t03|t04|t05|t06|t07|t08|t40|t09|t41|t42|t43|t44|t45
4|912|0|0|0|0|1473|133|223|0|939|0|0|25216|0|0|ALORGO|0|0|0|0|0|0|
0|0|0|0|6331|1240|11|0|0|0|0|0|0|0|0|189|0|0|0|0|0|0|0|0|0|443
15|0|0|82724|82724|82571|70500|0|0|0|0|0|0|1605|132
##END|PERM_OBS_CLASS_T_ALORGO

## Parser Intermediate File
##START|HEADER
OMC|record_code|StartTime|class_id|NODE|Date|StartDate|xchg_num
760OBTS0|8|22:00|8|SSP01|13722025|2004May16|1
##END|HEADER
##START|PERM_OBS_CLASS_T_ALTRA
t46|t47|t48|t49|t10|t11|t12|t13|t14|t15|t16|t17|t50|t18|t51|t19|pr
efix|t52|t53|t54|t55|t56|t57|t58|t59|t20|t21|t22|t23|t24|t25|t26|t
27|t60|t28|t61|t29|t62|t63|t64|t30|t31|t32|t33|t34|t35|t36|t37|t38
|t39|t01|t02|t03|t04|t05|t06|t07|t08|t40|t09|t41|t42|t43|t44|t45
```

```
0|209|0|0|0|0|746|2|0|0|3|0|0|12548|0|0|ALTRA|0|0|0|0|0|0|0|0|0|
1736|582|0|0|0|0|0|0|0|0|0|0|7|0|0|0|0|0|0|0|0|0|12381|0|0|29446
|29446|29446|20293|0|0|0|0|0|0|1230|0
##END|PERM_OBS_CLASS_T_ALTRA

## Parser Intermediate File
##START|HEADER
OMC|record_code|StartTime|class_id|NODE|Date|StartDate|xchg_num
760OBTS0|8|22:00|8|SSP01|13722025|2004May16|1
##END|HEADER
##START|PERM_OBS_CLASS_T_ALINT
t46|t47|t48|t49|t10|t11|t12|t13|t14|t15|t16|t17|t50|t18|t51|t19|pr
efix|t52|t53|t54|t55|t56|t57|t58|t59|t20|t21|t22|t23|t24|t25|t26|t
27|t60|t28|t61|t29|t62|t63|t64|t30|t31|t32|t33|t34|t35|t36|t37|t38
|t39|t01|t02|t03|t04|t05|t06|t07|t08|t40|t09|t41|t42|t43|t44|t45
0|0|0|0|0|0|1|0|0|1|0|0|0|0|0|0|ALINT|0|0|0|0|0|0|0|0|0|0|0|0|0|
0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|0|108|0|0|109|110|0|110|0|0|0|
0|0|0|0|0
##END|PERM_OBS_CLASS_T_ALINT
```

4. Tech Pack Support

Tech pack support is included in the Alcatel NSS Gateway for the following Performance Manager solutions:

- Alcatel RCP
- Alcatel OCB
- Alcatel LM (GSM Alcatel NSS LM9 TP v1.2.13)

The EngineConfig.pm and UserConfig.pm configurations for these Tech Packs are located in the tech_pack_support kit sub directory.