

# **Nokia ASCII Gateway User Guide**

Release: 3.4.0

Date: January 29, 2008

---

## Contents

<b>CONTENTS</b>	<b>2</b>
<b>REFERENCES</b>	<b>4</b>
<b>GLOSSARY</b>	<b>4</b>
<b>PREFACE</b>	<b>5</b>
ABOUT THIS GUIDE	5
CONVENTIONS	5
<b>1. OVERVIEW</b>	<b>6</b>
1.1 THE GATEWAY FRAMEWORK	6
1.2 NOKIA ASCII OVERVIEW	6
1.2.1 Network Details	6
1.2.2 Data Types	6
1.2.3 Data Version Support	7
1.2.4 Data/File Formats	7
1.2.5 Architectural extensions	9
<b>2. ENGINE RULES AND CONFIGURATION</b>	<b>9</b>
2.1 NOKIA_ASCII	9
2.1.1 PIF Naming	9
2.1.2 Rule Configuration	10
2.2 NOKIA_MML	14
2.2.1 PIF Naming	14
2.2.2 Rule Configuration	15
<b>3. POST PARSER RULES AND CONFIGURATION</b>	<b>18</b>
3.1 CNAME_MANIP	18
3.1.1 Rule Configuration	18
3.1.2 Sample Usage	18
3.2 VALIDATE_AGGREGATE	19
3.2.1 Rule Configuration	19
3.2.2 Sample Usage	20
<b>4. TECH PACK SUPPORT</b>	<b>21</b>
<b>5. HIERARCHY INFORMATION</b>	<b>21</b>



## References

<b>Name</b>	<b>Description</b>
Gateway Framework User Guide	This use guide describes in detail the functionality of the Gateway Framework, and the standard suite of tools available.

## Glossary

## Preface

### About this Guide

This guide details the vendor specific information on the 3GPP XML Gateway. It contains the following information:

- *Chapter 1: Overview.* This chapter gives a brief description of the 3GPP XML 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 Nokia ASCII uses the Gateway Framework as a container for the execution of its engine and post parser stages. The Gateway Framework and Vendor 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 Vendor Gateway plugs into the Gateway Framework and extends this functionality to provide the final Gateway that parses 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 Nokia ASCII Overview

#### 1.2.1 Network Details

This Gateway supports raw performance and hierarchy data from a number of different sources:

- NSS, BSS and SGSN performance data from the Nokia NetAct Performance Management system, using the NOKIA\_ASCII.pm engine.
- SGSN configuration data, provided in ASCII MML report format, which is parsed by the NOKIA\_MML.pm engine.

#### 1.2.2 Data Types

The types of data supported are:

- For NSS, BSS and SGSN performance data the Gateway supports performance data from the Nokia NetAct Performance

Management System. This includes an extensive list of measurement object types within each network type.

- For SGSN configuration data, the MML type output is supported.

### 1.2.3 Data Version Support

- For NSS, BSS and SGSN performance data. the version supported are T12 and OSS3.1
- For SGSN configuration data, version information is not relevant.

### 1.2.4 Data/File Formats

#### 1.2.4.1 NSS, BSS and SGSN performance data

For the performance data, the standard performance data format from the Nokia ASCII netact system is supported. The performance data consists of:

- A File Header: The first line in the raw file, the file header is formatted as follows:

`<NE_TYPE><NE_ID>, <meas_type>, <meas time>, <duration>;`

where

NE\_TYPE – the type of network element, such as HLR or MSC for NSS data.

NE\_ID – The identifier of the network element, a 6 digit number.

Meas type – the type of measurement that follows.

Meas time – starting time of the measurement period.

Duration – length of measurement period in minutes.

The following format is then repeated:

- The Object ID: This identifies the measurement target, which is made up of 0 or more numeric object identifications. The combination of this line and the NE is unique for all elements in the network.
- Counter name, counter value: Each subsequent line, until delimited by a ";" to indicate the end of record consists of a counter name and counter value associated with the object ID.

A sample of the file header and one measurement object:

```
BSC30936, 90, 200403311900, 60;  
36, 11,  
INT_ID, 21,
```

```

PERIOD_START_TIME, 20040331190000,
PERIOD_STOP_TIME, 20040331200000,
PERIOD_REAL_STOP_TIME, 20040331200113,
PERIOD_REAL_START_TIME, 20040331190000,
PERIOD_DURATION, 60,
SEGMENT_ID, 36,
QOS_PRIORITY_CLASS, 11,
NBR_OF_TBF_ALLOCATIONS, 11,
TOTAL_NBR_OF_RLC_BLOCKS, 11,
TOTAL_DURATION_OF_TBFS, 27,
DROPPED_DL_LLC_PDUS_OVERFLOW, 0,
DROPPED_DL_LLC_PDUS_LIFETIME, 0,
AVE_MS_BSSGP_FLOW_RATE_SUM, 0,
AVE_MS_BSSGP_FLOW_RATE_DEN, 1;

```

#### 1.2.4.2 SGSN Configuration data.

SGSN configuration is in the format of MML report output. The format of the report can be broken into:

- A report header, containing the date and time the report was generated.
- A set of header lines, which identify the hierarchy information for the following configuration entries. These are mapped to header output in the PIF.
- A set of data lines, which containing the individual configuration entries.

This set of header and data lines are then repeated for each network element.

A sample of the MML report format:

EJL:PAPU=0;

DX 220      16-23-0201                      2002-05-23    15:11:57

#### NETWORK CONFIGURATION DATA OUTPUTTED

NSEI-02750

    PAPU-00

        LAC-00252

            RAC-002

CI-23382	BVCI-10005	STATE-WO
CI-25152	BVCI-10018	STATE-WO
CI-07202	BVCI-10002	STATE-WO
CI-39872	BVCI-10059	STATE-WO
CI-25122	BVCI-10015	STATE-WO
CI-27802	BVCI-10022	STATE-WO
CI-31682	BVCI-10053	STATE-WO
CI-39662	BVCI-10055	STATE-WO
CI-31822	BVCI-10041	STATE-WO
CI-32202	BVCI-10045	STATE-WO
CI-39722	BVCI-10048	STATE-WO
CI-25062	BVCI-10009	STATE-WO



CI-25072 BVC1-10010 STATE-WO

### 1.2.5 Architectural extensions

None

## 2. Engine Rules and Configuration

### 2.1 NOKIA\_ASCII

The NOKIA\_ASCII engine process the performance data files for NSS, BSS and SGSN data. There are several steps involved in the processing of the raw file:

1. Process the file header, extracting the network element id, and measurement type. The appropriate configuration is then used to ensure that the measurement is supported. The configuration of the measurement types is contained in configuration files NOKIA\_BSS\_Config.pm, NOKIA\_NSS\_Config.pm and NOKIA\_SGSN\_Config.pm. The format of these files is described below.
2. The object id line is then parsed.
3. The secondary hierarchy data, from the hierarchy configuration files is inserted based on the key from the object id line.
4. The counter names and values are mapped to a single row in the output PIF.

#### 2.1.1 PIF Naming

The naming format of the PIF files is configurable. This section describes the standard naming supported by the default configuration. This should be sufficient unless there is a specific requirement to rename the PIF files differently.

The PIF naming consists of:

- the name of the measurement type, derived from a lookup on the object identifier in the configuration file. e.g. P\_NBSC\_PBCCH\_AVAIL.
- The startdate from the file header, e.g. 20040331.
- The NODEID derived from lookup of the hierarchy file.
- The starttime from the file header.
- The network type derived from the file header.
- The object instance derived from the file header.

- The object identifier derived from the file header.

An example of a PIF file name for measurement 90, P\_NBSC\_QOS.  
P\_NBSC\_QOS-#-20040331-#-TBD0004BSC1-#-19:00-#-BSC-#-30936-#-90-#-I.pif

Significant performance improvement can be achieved by using fewer keys for the file header. This should generate larger PIF files.

### 2.1.2 Rule Configuration

The configuration of the NOKIA\_ASCII rule is split into 2 separate files due to its complexity:

- EngineConfig.pm which as normal describes the main configuration of the rule.
- NOKIA\_<Network Type>\_Config.pm that contains the configuration of the measurement types handled for a particular Network Type e.g. Nokia\_BSS\_Config.pm. This configuration is then referenced within the appropriate rule in EngineConfig.pm. The configuration of these measurement objects will be described but in general should not need to be changed during installation.

#### 2.1.2.1 EngineConfig.pm

EngineConfig.pm contains the main configuration of the NOKIA\_ASCII rule. In addition to the standard entries, the NOKIA\_ASCII rule mandatory entries are:

- **HEADER\_LINE\_DESCRIPTION:** A regular expression detailing the format of the header line in the input file.

```
HEADER_LINE_DESCRIPTION => '^(BSC)\w+\, \w+\, \w+\, \w+\;.*$',
```

- **FIELD\_DELIMITER:** Describes the delimiter used within the configuration files from which the hierarchy information is read.

```
FIELD_DELIMITER => ', ',
```

- **HEADER\_FIELDS:** A list header field names which represents the number of fields in the HEADER\_LINE\_DESCRIPTION separated by FIELD\_DELIMITER.

```
HEADER_FIELDS => [ qw(ASCII_ID OMTYPE DATE DURATION) ],
```

- **HEADER\_PROCESSING:** The names and values of the header counters to be extracted from the line matched in the HEADER\_LINE\_DESCRIPTION.

```
HEADER_PROCESSING => {
  DATE => {
    STARTDATE => '^(\\d{8}) .+',
    STARTTIME => '^(\\d{8})(\\d{4}) ',
```

```

    },
    ASCII_ID => {
        OBJECT_INSTANCE => '^[a-zA-Z]+(\d+).*',
        NETWORK_TYPE => '^[a-zA-Z]+.*',
    },
},

```

- **HEADER\_INFO\_FOR\_PIF\_FILENAME:** An array, listing the fields derived from the header line that will be used to create the PIF filename:

```

HEADER_INFO_FOR_PIF_FILENAME => [ qw(STARTDATE NETWORK_TYPE
BSCID OMTYPE) ],

```

- **OBJECTID\_LINE\_DESCRIPTION:** At the beginning of each record is an object id description. This configuration describes the valid format of this line that will consist of up to 5 comma separated keys.

```

OBJECTID_LINE_DESCRIPTION      => '^(\\d*\\,*){0,5}.*',

```

- **RECORD\_DELIMITER\_DESCRIPTION:** At the end of each record is delimiter. This configuration describes the valid format of this line that will consist of a comma separated values with a semi column at the end.

```

RECORD_DELIMITER_DESCRIPTION  => '^\\w+\\, [\\w\\-]*\\;.*',

```

- **OBJECT\_TYPES:** A hash that describes the information associated with each different measurement type. This is quite a large hash as there can be many different objects for each network component. In this section the possible values configured for each are described.

The subroutine and configuration are stored in the module associated with that network type, NOKIA\_BSS\_Config.pm, NOKIA\_NSS\_Config.pm and NOKIA\_SGSN\_Config.pm.

See the next section on the format of these configuration files.

```

OBJECT_TYPES => nokia_nss(),

```

The optional entries are:

- **HEADER\_INFO\_FOR\_DATA\_RECORD:** For certain installations, input files can contain measurements for the same om type for several different BSCs etc. This option allows you to configure the header info that you want inserted per record rather than as normal in the header. For example NODEID, BSCID etc can be inserted in every record to create larger PIFs, which will be faster for post parser rules. It is configured empty by default.

```
HEADER_INFO_FOR_DATA_RECORD => []
```

- **CONFIGURATION\_DATA\_FOR\_HEADER:** The fields, extracted from the lookup in of the hierarchy data that should be inserted into the PIF header.

```
CONFIGURATION_DATA_FOR_HEADER => {
    NODEID => "NODEID",
},
```

- **IGNORE\_LINE\_IF\_NETWORK\_CONFIG\_DATA\_MISSING:** This scalar specifies whether or not to ignore lines for which the hierarchical data cannot be found. If it is set to true, then any data lines for which the lookup of key data fails will be removed from the PIF output. By default it is configured as false.

```
IGNORE_LINE_IF_NETWORK_CONFIG_DATA_MISSING => '0',
```

- **CONFIGURATION\_DETAILS:** A complex hash containing details on the network configuration files. Hierarchy data from these files is integrated into PIF output records. There are 4 possible entries:

HIERARCHY, TRXHIER, SGSN and OBJECT.

Each has entry contains the following configuration elements:

- **DIR:** The path name that is pointing to the hierarchy files.
- **FILENAME:** The regular expression of hierarchy filename. These files store the hierarchy data.
- **FIELDS:** An array mapping the fields in the input file to counter names.
- **KEYS:** The sets of simple and complex keys to generate for lookup. These keys are then used while processing the performance files, to insert the relevant hierarchy data into each PIF row.

```
CONFIGURATION_DETAILS => {
    HIERARCHY => {
        DIR      => '../hier/',
        FILENAME => [ '^hierarchy.*\.dat$', ],
        FIELDS   => [qw(BSCID BTSID BSCNAME BTSNAME BSNAME
LACID CELLID DEFTCH DEFCCH MSCNAME)],
        KEYS     => {
            LAC_CELL_KEY => [qw(LACID CELLID)],
            LAC_KEY      => 'LACID',
        },
    },
    OBJECT => {
        DIR      => '../hier/',
        FILENAME => [ '^objects.*\.dat$', ],
        FIELDS   => [qw(OBJECT_INSTANCE NAME)],
        KEYS     => {
            OBJ_KEY      => 'OBJECT_INSTANCE',
            OBJ_NAME_KEY => 'NAME',
        },
    },
},
```

```
    },
  },
```

- **DEFAULT\_CONCATENATION\_CHAR:** A single character scalar with the string to be used when concatenating values which are being used to create a single counter value in the output PIF.

```
DEFAULT_CONCATENATION_CHAR => '/',
```

- **USE\_PIF\_FILENAME\_COUNTER:** Is set a unique number will be inserted in the PIF filename to ensure that the output PIFs are unique and do not overwrite each other, especially when large raw files are being used, which have previously been concatenated together.

```
USE_PIF_FILENAME_COUNTER      => "True",
```

- **INPUT\_NULL\_VALUE:** A scalar with the string for null data value.

```
INPUT_NULL_VALUE => '-1',
```

#### 2.1.2.2 Network Measurement Object Configuration

As stated above, the Nokia\_<XXX>\_Config.pm files detail the mapping of each measurement object within each network type. Each file has the same format. A hash detailing the configuration for each network type.

Each element within this hash details the configuration for mapping from the measurement object to the PIF. The hash key is the measurement object value e.g. 160.

The configuration entries for each measurement object are detailed below:

- **NAME:** The actual name that the measurement type maps to. Used in the output PIF filename.

```
NAME      => 'P_NBSC_TRAFFIC',
```

- **OBJECT\_ID\_NAMES:** An array with the names of the expected entries in the object id line.

```
OBJECT_ID_NAMES => [qw(BTSID TRX_TYPE)],
```

- **RECORD\_KEY:** An array with the names of the keys that are to be used when searching the network configuration information.

```
RECORD_KEY      => [qw(BSCID BTSID)],
```

- **HIERARCHY\_KEY\_SET:** A scalar referring to the hierarchy key from the hierarchy configuration in the engine which is to be used for lookups for this OM type. This key set must be

configured in the hierarchy configuration in CONFIGURATION\_DETAILS for the rule.

```
HIERARCHY_KEY_SET => 'BSCID_BTSID_KEY',
```

- **OUTPUT\_COUNTERS:** The OUTPUT\_COUNTERS is a hash entry whose keys contain the network configuration counter names to be integrated into the current record. The corresponding hash values contain the names of the network configuration column counters or object ID line entries. This may be a single or a list of counter names. In the event of a list of names, the output counter value will be the concatenation of all these.

```
OUTPUT_COUNTERS => {  
    NODEID => 'NAME',  
},
```

## 2.2 NOKIA\_MML

Due to the report format of the MML output files parsed by the NOKIA\_MML rule, the parsing of the input files is based upon parsing of a file header and then multiple blocks each of which contains a block header and a list of data blocks.

It may be useful to examine the raw input files that are parsed by this rule while reading the description. They are contained in example/data/nokia\_mml directory of the release package.

The format of the file consists of:

- a report header line, extracted and mapped to the PIF header and filename.
- a repeating set consisting of:
  - report header lines, which detail hierarchy configuration information. This set of lines maps to the header in the output PIF.
  - Report detail lines, which detail the configuration data. Each line is mapped to a data row in the output PIF.

### 2.2.1 PIF Naming

The naming of the PIF file is controlled by 2 configuration entries, one detailing the elements extracted from the raw file and the other detailing the block information from the PIF header to be included in the PIF filename.

The PIF filename typically contains hierarchy information and the date and time the MML command was executed to ensure uniqueness.

### 2.2.2 Rule Configuration

This section details the non-standard entries for the NOKIA\_MML rule.

- **OUTPUT\_FILENAME\_START**: An optional string to prepend to the PIF filename

```
OUTPUT_FILENAME_START      => 'ZFWO',
```

- **BLOCK\_NAME**: The name of the block to use in the PIF

```
BLOCK_NAME                  => 'ZFWO_BLOCK',
```

- **FILE\_HEADER\_DESC**: This is an array of hash elements each of that describes an expected file header line in the input. Each entry consists of a **LINE\_DESC**, a **RE** to match the line on, and **FIELDS** that is an array of counter names to assign the extracted values to.

```
FILE_HEADER_DESC =>
[
  {
    LINE_DESC =>
      'DX\s+\d+\s+([\d\-\+])\s+([\d\-\+])\s+([\d\:\+]) ',
    FIELDS    => [qw (FILE_BSC MML_DATE MML_TIME)],
  },
],
```

- **FILE\_HEADER\_INFO\_FOR\_PIF\_FILENAME**: An array containing the names of the counters extracted in **FILE\_HEADER\_DESC** that are to be used in the PIF filename.

```
FILE_HEADER_INFO_FOR_PIF_FILENAME =>
[qw (FILE_BSC MML_DATE MML_TIME) ]
```

- **BLOCK\_INFO\_FOR\_PIF\_FILENAME**: Similar to above this describes the counters, extracted from the report headers that is to be used in the PIF filename.

```
BLOCK_INFO_FOR_PIF_FILENAME => [ qw (NSEI BSCU PCU) ],
```

- **FILE\_HEADER\_INFO\_FOR\_PIF\_HEADER**: This array contains the list of counters, extracted in the **FILE\_HEADER\_DESC** that are to be put in the PIF header.

```
FILE_HEADER_INFO_FOR_PIF_HEADER =>
[qw (FILE_BSC MML_DATE MML_TIME) ]
```

- **BLOCK\_INFO\_FOR\_PIF\_HEADER**: The names of the counters, extracted from the report block header to be written to the PIF header.

```
BLOCK_INFO_FOR_PIF_HEADER => [
  qw (NSVCI NSEI BSCU PCU NSVC_NAME)
  qw (NSVC_OP_STATE DLICI DLICI_OP_STATE CIR BEARER CHANNEL)
```

1,

- **BLOCK\_DESCRIPTION:** This hash describes the core processing of the report details lines in the MML input file.

It contains a number of sub-elements described below:

- **BEGIN\_DESCRIPTION:** A scalar RE, describing the RE to match which marks the start of a report block.  
**BEGIN\_DESCRIPTION => 'NSEI-\d+',**
- **LINE\_DESC:** An array of hash elements, each of which describes a report line to be matched during processing. Any lines that are not matched by one of the RE's in this list are ignored. With each **LINE\_DESC** element a number of configuration fields are used:
- **LINE\_DESC:** A RE to use to match the line. Once the RE is true for a line the other line configuration is used to control what happens to the line. These options are detailed below. Fields can be extracted from the matched RE either by bracketing them as normal in **LINE\_DESC**, or by specifying the **LINE\_SEPARATOR** (see below).  
**LINE\_DESC => '^NSEI-(\d+)\s+BCSU-(\d+)\s+PCU-(\d+)'**
- **LINE\_SEPARATOR:** The string used to split the fields in the matched line.  
**LINE\_SEPARATOR => '\s+',**
- **FIELDS:** The names of the counters that are extracted from the matched line. In the case ZFWO files multiple name-value pairs exist on a single report line and so one report line can map to multiple rows in the PIF.  
**FIELDS => [ qw(NSEI BSCU PCU) ],**
- **LINE\_TYPE:** This defines the type of report line which has been matched, which can be either **HEADER**, in which case it will be mapped into the PIF header, or **DATA** in which case the value is mapped into the next row written.  
**LINE\_TYPE => 'DATA'**
- **WRITE\_LINE:** (Only applies to **LINE\_TYPE 'DATA'**) A boolean used to define whether the matched line is to be written out as a line in the PIF. If not, the counter value will be stored to be used in the next rows written to the PIF.  
**WRITE\_LINE => 'TRUE',**

Given the report extract below:

```

NSEI-02750
  PAPI-00
    LAC-00252
      RAC-002
        CI-23382  BVCI-10005  STATE-WO

```



CI-25152 BVC1-10018 STATE-WO  
 CI-07202 BVC1-10002 STATE-WO

The NSEI, LAC, RAC are not written directly to the PIF but are used as counter values in the real data rows, which start with the CI marker.

A line description to match the main report details line above is:

```
{
  LINE_DESC => '^\\s+CI-(\\d+)\\s+BVC1-(\\d+)\\s+STATE-(\\w+)',
  FIELDS => [ qw (CI BVC1 STATE) ],
  LINE_TYPE => 'DATA',
  WRITE_LINE => '1',
},
```

- **NEW\_COUNTERS:** This is an array containing descriptions of new counters to be derived from the counters extracted from the input report. Each array element contains a hash with the following elements:

- **COUNTER:** The name of the counter to be transformed

```
COUNTER      => 'MML_ID',
```

- **NEW\_COUNTER:** The name of the new counter to insert.

```
NEW_COUNTER   => 'BSNAME',
```

- **COUNTER\_DESC:** A RE describing the fields to be extracted from the original counter value. The values to be extracted are bracketed ().

```
COUNTER_DESC  => '\\d{4}\\d{2}\\d{2}\\d{1}',
```

- **NEW\_COUNTER\_DESC:** A scalar to describe the new format of the counter, using the extracted values above to recreate the counter. The format is the same as the normal format used in REs to match text.

```
NEW_COUNTER_DESC => '$1-$2-$3/$4',
```

- **CONFIGURATION\_DETAILS:** Similar to NOKIA\_ASCII this consists of the configuration required for the insertion of the hierarchy data into each row of the pif. This hash consists of a series of hash entries, each of which defines a hierarchy file and key mapping.

The fields in each entry are:

- **DIR:** The name of the path that is pointing to the hierarchy file.

```
DIR => '../hierarchy/',
```

- **FILENAME:** An array of regular expressions of the hierarchy file name. The hierarchy information is stored in those files.

```
FILENAME => [ '^hierarchy.*\\.dat$', ],
```

- **FIELDS:** An array describing the list of fields contained on each row of the file.

```
FIELDS => [qw(BSCID BTS_ID BSCNAME BTSNAME BSNAME  
              LACID CELLID DEFTCH DEFCCH MSCNAME) ],
```

- **KEYS:** An array describing the keys used to map between the raw data and the hierarchy data.

```
KEYS => [ 'BSNAME' ],
```

- **FIELDS\_TO\_INSERT:** The fields from the hierarchy data to insert into the PIF when the key is matched.

```
FIELDS_TO_INSERT => [ qw(BTSNAME BSCNAME) ],
```

### 3. Post Parser Rules and Configuration

#### 3.1 CNAME\_MANIP

The CNAME\_MANIP is used for counter manipulation. Part of the Nokia ASCII requirements is that counter names must be prepended with certain values.

This rule will prepend either a default or custom string, based on the current counter name, to rename the counter in the output PIF.

##### 3.1.1 Rule Configuration

The non-standard configuration entries associated with this rule are:

- **DEFAULT\_PREPEND\_STR:** A scalar describing the default string to be used to prepend all *strings with*.

```
'DEFAULT_PREPEND_STR' => "BSC_",
```

- **CUSTOM\_PREPEND\_STR:** A hash mapping a counter name to an value to prepend it with.

```
'CUSTOM_PREPEND_STR' => {  
    'PERIOD_START_TIME' => 'PP_',  
    'NOPERIOD_' => 'QS_',  
    'PERIOD_STOP_TIME' => 'PS_',  
}
```

**NOTE:** Either DEFAULT\_PREPEND\_STR or CUSTOM\_PREPEND\_STR must be configured. If both are, DEFAULT\_PREPEND\_STR takes precedence and all counters will be prepended with that value.

##### 3.1.2 Sample Usage

Given the following input PIF:

```
## Parser Intermediate File
```

```
##START|HEADER
Network_FileType|BSCID|NODEID|STARTTIME|OMTYPE|DURATION|NETWORK_TYPE|STA
RTDATE
BSC|30936|TBD0004BSC1|19:00|90|60|BSC|20040331
##END|HEADER
##START|P_NBSC_QOS
AVE_MS_BSSGP_FLOW_RATE_SUM|INT_ID|SEGMENT_ID
1|3|5
1|7|9
```

each counter name will be prefixed with BSC to produce the following output:

```
## Parser Intermediate File
##START|HEADER
Network_FileType|BSCID|NODEID|STARTTIME|OMTYPE|DURATION|NETWORK_TYPE|STA
RTDATE
BSC|30936|TBD0004BSC1|19:00|90|60|BSC|20040331
##END|HEADER
##START|P_NBSC_QOS
BSC_AVE_MS_BSSGP_FLOW_RATE_SUM|BSC_INT_ID|BSC_SEGMENT_ID
1|3|5
1|7|9
```

The configuration required to produce this output is below:

```
{
  'RULE_TYPE'          => 'CNAME_MANIP',
  'RULE_DESC'          =>
    'Alternative counter names P_NBSC_LOAD',
  'INPUT_FILE_DESCRIPTION'=>
    ['^P_NBSC_LOAD-#-(\d{8})-#-.*-#-\d{2}:\d{2}-#-BSC-#-\d+-#-\d+)-#-
I.pif'],
  'PRODUCE_PIF'        => 'True',
  'PRODUCE_LIF'        => 0,
  'DEFAULT_PREPEND_STR' => "BSC_",
},
```

## 3.2 VALIDATE\_AGGREGATE

This rule is used to accumulate configured counters over a number of records within a PIF. It is also configured with validating counters that are used to identify from which PIF data row all other non-accumulated counter values should be taken. In this way as well as combining a number of rows via accumulation, the correct non-accumulated counter values can also be inserted into the output PIF data.

### 3.2.1 Rule Configuration

The non-standard configuration entries for this rule are:

- **COUNTERS\_TO\_ACCUMULATE:** An array list of counters to be accumulated over all records.

```
COUNTERS_TO_ACCUMULATE => ['LOC_AREAS_IN_USE'],
```

- **COUNTERS\_TO\_VALIDATE\_ON:** The names of the counters to use within each record to check for nulls, while looking for the valid record.  
`COUNTERS_TO_VALIDATE_ON => ['ROAM_DB_OP'],`
- **VALID\_NULL\_VALUES:** An array describing the values that are to be interpreted as null values within each row.  
`'VALID_NULL_VALUES' => [''],`

In this case the counters above would be considered nulls when they are equal to "".

- **COUNTERS\_TO\_SORT\_ON:** If specified the counters in this array will be used to sort the records and a new record will be created in the output pif for each unique key.  
`COUNTERS_TO_SORT_ON => ['CELLID']`

#### NOTE:

1. All counters must be non null for a record to be considered valid.
2. The accumulate counters are accumulated over all matching null and non null records.
3. If no record is found with all COUNTERS\_TO\_VALIDATE\_ON with non null values, a warning will be logged. The values in first record read in will be used as the output counters.

### 3.2.2 Sample Usage

Given the following PIF:

```
## Parser Intermediate File
##START|HEADER
Network_FileType|BSCID|NODEID|STARTTIME|OMTYPE|DURATION|NETWORK_TYPE
BSC|30936|TBD0004BSC1|19:00|90|60|BSC
##END|HEADER
##START|CELLEDATA
CELLID|ROAM_DB_OP|LOC_AREAS_IN_USE
333333||10
333333|45|20
333333||80
222222||800
222222|50|200
```

The following output is produced:

```
##START|HEADER
Network_FileType|BSCID|NODEID|STARTTIME|OMTYPE|DURATION|NETWORK_TYPE
BSC|30936|TBD0004BSC1|19:00|90|60|BSC
##END|HEADER
##START|CELL1
CELLID|ROAM_DB_OP|LOC_AREAS_IN_USE
333333|45|100
222222|50|1000
```

The counter LOC\_AREAS\_IN\_USE is accumulated for each CELLID, and the value of ROAM\_DB\_OP considered valid, in this case any non '' value, is output in the accumulated row.

The following rule configuration would achieve the above

```
{
    'RULE_TYPE'           => 'VALIDATE_AGGREGATE',
    'RULE_DESC'           => 'Validate MSC files',
    'INPUT_FILE_DESCRIPTION' => ['^P_MSC_VLR-#-(\d{8}-#-.*).pif'],
    'OUTPUT_BLOCK_NAME'   => 'CELL1',
    'PRODUCE_LIF'         => 'TRUE',
    'PRODUCE_PIF'         => 'TRUE',
    'COUNTERS_TO_VALIDATE_ON' => ['ROAM_DB_OP'],
    'VALID_NULL_VALUES'   => [''],
    'COUNTERS_TO_ACCUMULATE' => ['LOC_AREAS_IN_USE']
    'COUNTERS_TO_SORT_ON'  => ['CELLID']
},
```

NOTE: Using the PERLIZE rule, now included in the Gateway Framework, will be able to fulfil the requirement of DLCI data normalisation (for certain TechPacks).

## 4. Tech Pack Support

Gateway configurations currently available for the following Tech Packs:

- Nokia BSS S11.5

## 5. Hierarchy Information

Hierarchy information is produce by customer or vendor using a set of extraction tools. The extraction tools will query the NetAct database to extract the necessary config data. An example of the extraction tool is provided with this vendor gateway.