Siemens BSS Gateway User Guide

Release: 3.4.0

Date: 28 April 2008

Contents

References

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

Glossary

BSS	Base Station Subsystem
PIF	Parser Intermediate Format
LIF	Loader Input Format

Preface

About this Guide

This guide details the Vendor specific information on the Gateway release for Siemens BSS. 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 Siemens BSS Gateway 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 Siemens BSS Gateway Overview

1.2.1 Network Details

This Gateway processes:

- BSS performance data from network elements in the GSM/GPRS network.
- GPRS hierarchy data.

1.2.2 Data Types

The data types supported are:

- For BSS performance data, the Gateway supports Scanner type measurements exported from the Siemens' Radio Commander Performance Management system.
- For GPRS hierarchy data, the Gateway supports the parsing of MML command strings, extracted from the OMC.

1.2.3 Data Version Support

The versions supported for each type:

- For BSS performance data, version BR6 and BR7
- For GPRS hierarchy data, version BR5.5

1.2.4 Data/File Formats

The data/file formats are:

- For BSS performance data, the files are in ASCII format. Each line in the raw data contains several fields:
  1. The measurement identifier – a pair of digits which identifies the measurement object class.
  2. The date and time of the measurement collection.
  3. Identification of the monitored object instance. Examples of this are BSC, Cell, TRX identifiers.
  4. The release version of Radio Commander System which generated the data.
  5. A symbolic name for the measurement object class.
  6. The granularity period of the measurement. This can be 5, 15, 30 or 60 minutes.
  7. The object result number. This defines the number of elements for which counters are present. In the case of target cell or channel measurements there may be more than one set of counter data, as there may be more than one target cell for handover measurements.
  8. The optional value list. In the case of BSC measurements, it identifies the cell type. For handover measurements it may contain the old and new cell identifiers. For channel measurements it identifies the sub-channel within the channel object.
  9. The validity list. Details the validity of the counter data that follows.
  10. List of counters: The list of performance counter values.

A sample of a raw data, of class 0 (SCANBSC), sub-class 1. Some spaces have been removed for the sake of clarity.

```
   (0,1) 09/01/2004 10:00 bsc {12 0}  60 SCANBSC 60  1   0    0   27538
```

- For GPRS hierarchy data, the files are in ASCII format. Each line contains an independent command for the creation/update of hierarchy data on the OMC system. Entries within these MML commands consists:
  1. A command e.g. “SET BSC”.
  2. A series of entries for different pairs of names and values to be set, each separated by a comma.

A sample line from the raw data file:

```
   SET BSC PKGBSCC:NAME=BSC:0,MEDAFUST=0-36,MEDAFUPE=UPPE_1H;
```

1.2.5 Architectural extensions

None

2. Engine Rules and Configuration

2.1 SIEMENS_BSS

The SIEMENS_BSS engine processes the performance data files output by the Siemens’ Radio Commander Performance System. Each measurement object class from which performance data is required, is configured in the engine rule configuration. This determines the mapping to the output PIF, and the granularity of the PIF data created.

Multiple rows in the ASCII file are mapped to a single PIF row. For example, for SCANBSC, all measurement objects in class 0 are mapped to a single row. Class 0 contains 20 related measurement sub-classes, numbered 0 to 20, each of which is contained in a separate line in the raw data.

The counter data for a single BSC from each of these 20 sub-classes is mapped to one row.

Counter data within each row is identified using 3 elements:

- the measurement object
- the measurement sub-class

- the counter position in the data row.

For example the 6th counter for measurement class 0, sub-class 10, would map to C0_10_6 in the PIF data.

See the sample data supplied with the release for examples of the raw data.

2.1.1 PIF naming

This section describes the elements used in the creation of the PIF name in the standard configuration.

The PIF name consists of the following elements (in order):

- The symbolic measurement object name from the engine configuration e.g. SCANBTS.
- The BSS identifier from the raw filename.

And from the data in the raw file:

- The BSC_ID from the measurement object in the raw file.
- The date and time of the measurement.
- The hierarchy entries configured in MONITORED_OBJECT_KEYS in the rule configuration. This entry determines the number and granularity of the output PIFs produced. If the MONITORED_OBJECT_KEYS has granularity down to TRX level it will obviously create more files than a key at the BSC_ID level.

An example of a PIF file for a SCANBSC measurement class 0, for BSS_ID 02, and BSC_ID 2:

```
SCANBSC-#-02-#-2-#-04May2004-#-1400-#-2-#-PM-#-I.pif
```

2.1.2 Rule Configuration

The SIEMENS_BSS engine can be broadly broken into 2 sections:

- Configuration related to raw files and data mapping which applies to all measurement object classes. This defines information such as the PIF header data extracted from the filename, PIF name creation and network object naming.
- Configuration for each of the relevant measurement object classes. This details how a particular measurement class, e.g.

SCANBTS, is handled during processing. This configuration is contained in the OBJECT_CLASSES entry of the rule.

This section only details the non-standard configuration entries.

The mandatory entries are:

- HEADER_FIELDS_FOR_PIF_FILENAME: The fields extracted from the raw filename, and the MONITORED_OBJECT_KEYS in the OBJECT_CLASSES to use in the output PIF name.
```
HEADER_FIELDS_FOR_PIF_FILENAME =>  [ qw (BSS BSC_ID) ],
```

- MONITORED_OBJECTS: This hash defines the mapping of the network element names in the raw data, field 3, to counter names in the PIF output. For example the bsc element contains 2 network keys which are mapped to BSC_ID and BTSM_ID.
```
MONITORED_OBJECTS => {
    bsc =>    [ qw (BSC_ID BTSM_ID) ],
    bts =>    [ qw (BSC_ID BTSM_ID BTS_ID) ],
    trx =>    [ qw (BSC_ID BTSM_ID BTS_ID TRX_ID) ],
    chan =>   [ qw (BSC_ID BTSM_ID BTS_ID TRX_ID CHAN) ],
    ss7l =>   [ qw (BSC_ID LINKSET) ],
    tgtbts => [ qw (BTSM_ID BTS_ID) ],
    btsm =>   [ qw (BSC_ID BTSM_ID) ],
},
```

- OUTPUT_MEASURE_IDENTIFIER: Whether or not to output the full measurement name identifier in the PIF data. This is field 0 in the raw data, and will be mapped to measIdentifier_C<X>_<Y>, where X is the class name and Y is the sub class.
```
OUTPUT_MEASURE_IDENTIFIER => 0,
```

- KEEP_VALIDITY_VALUES: Determines what values in field 9 of the raw data are considered valid. If a data row contains a validity value not in this list the counter data is considered invalid.
```
KEEP_VALIDITY_VALUES => ['0', '1', '2', '3'],
```

- IGNORE_INVALID_VALUES: If set to true, counter values that are considered invalid will be discarded. If set to false, counter values that are considered invalid will be replace with DEFAULT_NULL_VALUE.
```
IGNORE_INVALID_VALUES => 0,
```

- ENABLE_PIF_CACHE: Determines when the internal PIF data cache in the engine rule is flushed. If set to true it will be written out when the raw file has been completely parsed. This has efficiency benefits as less PIF objects can be produced. If set to false, the cache will be flushed every time the object class changes in the raw data e.g. from SCANBSC to SCANBTS.

```
ENABLE_PIF_CACHE      => "True",
```

- DEFAULT_NULL_VALUE: The value to use for the counter value when the validity from the raw data indicates invalid counter data.

```
DEFAULT_NULL_VALUE => 'NULL'
```

The optional entries are:

- HEADER_DATA_RECORD_PROCESSING: This entry enables extra Perl code in the engine to do various manipulations to header and data records. This is the last process within the engine before the data is being output to the PIF files.

```
HEADER_DATA_RECORD_PROCESSING => sub {
    my ($blkname_ref, $h_ref, $d_ref) = @_;
    my $new_block_name = $$blkname_ref;

    if ( $new_block_name =~ /^SCANBSC_eBSC$/ ) {
        if ( exists $d_ref->{C0_13_96} ) {
            # 96 counters
            $new_block_name = "SCANBSC_eBSC_Extended";
        }
        elsif ( exists $d_ref->{C0_13_50} ) {
            # 50 counters
            $new_block_name = "SCANBSC_eBSC_Basic";
        }
        else {
            # 42 counters
            $new_block_name = "SCANBSC_eBSC";
        }
    }
    $$blkname_ref = $new_block_name;

    return 0;
},
```

The configuration for OBJECT_CLASSES is separated as it contains multiple sub entries.

- OBJECT_CLASSES: This complex hash contains the configuration for mapping each of the measurement class data objects to the PIF output. The name, e.g. SCANBSC, determines the symbolic name given to the class in the output PIF name.

The mandatory entries within each object class entry are:

- CLASS: The class for this measurement object for this group. This, in combination with MEASUREMENT_LIST, identifies the raw data counters for this object class.
```
      CLASS => 0,
```

- MEASUREMENT_LIST: The list of sub-classes that are part of this object class. This can be a single value, an array of values, or "*" to indicate all sub-classes.
```
      MEASUREMENT_LIST => '*',
```

- MONITORED_OBJECTS_KEYS: The set of keys to use as part of the PIF name. By default this is set to the BSC_ID, to reduce the number of PIFs created.
```
      MONITORED_OBJECTS_KEYS => [ qw(BSC_ID) ],
```

The optional entries within each object class entry are:

- INSERT_COUNTERS: A hash containing a list of counter names and their values to insert in each output row.
```
      INSERT_COUNTERS => {
          LINKTYPE => "DOWNLINK",
      },
```

- OPTIONAL_VALUE_COUNTER: If specified, field 8 of the raw data will be mapped to this counter name. This is used to identify different cell and channel types within the same monitored object key.
```
      OPTIONAL_VALUE_COUNTER => 'CHANNEL_SEL_BYTES',
```

- OPTIONAL_COUNTER_AS_PIF_KEY: If set to true, the counter in OPTIONAL_VALUE_COUNTER will be used as part of the output key for the PIF name. This is required for certain object classes, as the number of counter values for the same object sub class can be different depending on the value of the optional counter. For example for the SCANBTS object class, extended cells have twice as many counters as normal cells.
```
      OPTIONAL_COUNTER_AS_PIF_KEY => "True"
```

- OPTIONAL_COUNTER_AS_ROW_KEY: If set to true, the counter in OPTIONAL_VALUE_COUNTER will be used as part of the output key of the PIF data row. This is required for channel measurements such as SCANCHAN as the optional value counter identifies the sub-channel within a single channel key.
```
      OPTIONAL_COUNTER_AS_ROW_KEY => "True"
```

The example below shows the configuration for the SCANBSC and SCANBTS object classes:

```
OBJECT_CLASSES => {
    SCANBSC => {
        CLASS => 0,
        MEASUREMENT_LIST => '*',
        MONITORED_OBJECTS_KEYS => [ qw(BSC_ID) ],
    },
    SCANBTS => {
        CLASS => 1,
        MEASUREMENT_LIST => '*',
        MONITORED_OBJECTS_KEYS => [ qw(BSC_ID) ],
        OPTIONAL_VALUE_COUNTER => 'CELLTYPE',
        OPTIONAL_COUNTER_AS_PIF_KEY => 'True',
    },
},
```

2.2 SIEMENS_ACL

The SIEMENS_ACL engine processes MML command files generated by the OMC. These files contain commands used to create hierarchy data in the OMC.

For each CREATE row that is of interest an entry is created in the rule configuration. Within this entry the counters to extract and any manipulation of the counter names and values is configured.

The format of the file is relatively straightforward consisting of a MML "CREATE" command with a list of parameters. e.g:

```
CREATE SS7L:NAME=BSC:0/ss7l:0,LNKA=0-0,TSLA=0-25,SLC=0;
CREATE SS7L:NAME=BSC:0/ss7l:1,LNKA=1-0,TSLA=4-25,SLC=1;
```

Each different command type is mapped to a different PIF, with the associated set of counter names and values for that command.

All counters except for the NAME entry are mapped directly. The NAME counter is split into its constituent components. In the example above the counter:

```
NAME=BSC:0/ss7l:1
```

would be split into counters:

```
NAMEBSC|NAMESS7L
0|1
```

2.2.1 PIF naming

The PIFs are named from a combination of:

- Any fields extracted from the filename.
- The name of the create command object being extracted e.g. PCU, CHAN.

An example of a PIF name is:

```
dbBPK03D-#-BTS_PKGBTSB-#-I.pif
```

2.2.2 Rule Configuration

The section details the non-standard configuration entries for SIEMENS_ACL.

The mandatory entries:

- COMMAND_COUNTERS_FOR_HEADER: A list of counter names from the individual PIF data rows to insert in the PIF header.
  ```
  COMMAND_COUNTERS_FOR_HEADER => [ qw (NAMEBSC) ],
  ```
- COMMAND_CONFIG: This hash contains the MML entries that are to be extracted. The hash key, e.g. PCU, identifies the CREATE command to match. Within each entry there are several fields.
  - KEEP_COUNTERS: A list of REs to match the extracted counter names against. Any counter names extracted from the create command and matched in this list will be written out to PIF.
    ```
    KEEP_COUNTERS => [ qw(NAME.* NSEI) ],
    ```
  - NEW_COUNTERS: A hash map containing a new counter to create from a set of counters from those read in from the raw data. If more than one counter is being used to create the new counters, they will be joined using "/". In the example below, NAMEBTS is being created from NAMEPTPPKF.
    ```
    NEW_COUNTERS => {
        NAMEBTS => [ "NAMEPTPPKF"  ],
    },
    ```
  - MANIP_COUNTERS: A hash containing configuration of new counter values from extraction of sub-fields of existing counter values. In the example below the NSVLI is being reduced to its second set of digits after the "-".

```
MANIP_COUNTERS => {
    NSVLI => '\d+-(\d+)',
},
```

- EXTRACT_COUNTERS: A hash of hashes containing configuration that further extract counter value of specific field into hashes of counter name and value. Example below shows CELLGLID giving the value of "262"-"77"-6000-630, then it is further extracted into counters (MCC, MNC, LAC, CI). The keys of hash would represent the counter name while the value represent the regex for extraction of the counter value:

```
EXTRACT_COUNTERS => {
   CELLGLID => {
       # Extract the counters from the CELLGLID field
       # Eg. "262"-"77"-6000-630
       MCC => '^\"(\d+)\"-\"\d+\"-\d+-\d+$',
       MNC => '^\"\d+\"-\"(\d+)\"-\d+-\d+$',
       LAC => '^\"\d+\"-\"\d+\"-(\d+)-\d+$',
       CI  => '^\"\d+\"-\"\d+\"-\d+-(\d+)$',
   },
},
```

- OUTPUT_BLOCK_NAME: A scalar to specify the new block name for the new PIF generated. For example:

```
OUTPUT_BLOCK_NAME => 'CREATECHAN',
```

3. Post Parser Rules and Configuration

3.1 COUNT_ROWS

The COUNT_ROWS Post Parser rule counts the number of PIF rows in a secondary file, where counters of particular values match within the same PIF key. The count of the number of instances of each value matched is then inserted into the main PIF.

In the case of Siemens BSS data this rule is used to get the number of channels of each type associated with a particular TRX.

3.1.1 PIF naming

This section describes the non-standard configuration entries for COUNT_ROWS:

- INPUT_FILE_KEYS: The set of counters to use as the key. This is used to match each row in the main PIF with the related PIF key from the secondary PIF.
```
      INPUT_FILE_KEYS         => [ qw(NAMEBSC NAMEBTS) ],
```

- COUNT_FILE_DESCRIPTION: The secondary PIF file to count the numbers of rows in.
```
      COUNT_FILE_DESCRIPTION  => '(.*)-#-CHAN-#-I.pif',
```

- COUNT_FILE_KEYS: The set of counters to use as a key in the secondary file. This is used to co-relate the PIF rows from the secondary file with the primary file.
```
      COUNT_FILE_KEYS         => [ qw(NAMEBSC NAMEBTS) ],
```

- COUNTER_TO_COUNT: The name of counter to count the values of in the secondary PIF.
```
      COUNTER_TO_COUNT        => 'CHTYPE',
```

- COUNTER_VALUES_TO_MATCH: The counter values of the COUNTER_TO_COUNT to count the instances of. In the example below, the number of instances of the channel type counter values are being counted for each COUNT_FILE_KEY key.
```
      COUNTER_VALUES_TO_MATCH => [ qw(MAINBCCH SDCCH TCHFULL) ],
```

- OUTPUT_COUNTER_NAMES: The names to use in the main PIF when inserting the results of the count from the secondary PIF.
```
      OUTPUT_COUNTER_NAMES    =>
          [ qw(CHTYPE_MAINBCCH CHTYPE_SDCCH CHTYPE_TCHFULL) ],
```

- DEFAULT_NULL_VALUE: The value to insert for the new counters, if no match for the PIF key is found in the secondary PIF.
```
      DEFAULT_NULL_VALUE      => "NULL",
```

3.1.2 Sample Usage

Given the main PIF:

```
##START|BLOCK
C_5|NAMEBSC|C_6|NAMEBTS
5|1|3|1
3|1|6|2
4|1|7|3
##END|BLOCK
```

and a secondary PIF, containing the channel data:

```
##START|BLOCK
CHTYPE|NAMEBSC|NAMEBTS|C_1|C_2
SDCCH|1|1|1|2
MAINBCCH|1|2|1|2
MAINBCCH|1|3|1|2
TCHFULL|1|0|1|2
TCHFULL|1|1|1|2
TCHFULL|1|2|1|2
MAINBCCH|1|3|1|2
MAINBCCH|1|0|1|2
TCHFULL|1|1|1|2
TCHFULL|1|2|1|2
##END|BLOCK
```

produces the following output PIF:

```
##START|BLOCK
C_5|CHTYPE_MAINBCCH|C_6|NAMEBSC|CHTYPE_SDCCH|CHTYPE_TCHFULL|NAMEBTS
5|0|3|1|1|2|1
3|1|6|1|0|2|2
4|2|7|1|0|0|3
##END|BLOCK
```

In each row of the output PIF three new counters have been inserted. These counters contain the number of rows that matched the counter value in the secondary file.

For example for key NAMEBSC-NAMEBTS (1-1) on the first row of the PIF, there were 0 rows of CHTYPE_MAINBCCH, 1 of CHTYPE_SDCCH and 2 of CHTYPE_TCHFULL.

The configuration below was used to produce this output:

```
{
    RULE_TYPE                 => 'COUNT_ROWS',
    RULE_DESC                 =>
        'Count the number of chtypes matching bsc/bts key',
    INPUT_FILE_DESCRIPTION  => '(.*)-#-MAIN-#-I.pif',
    INPUT_FILE_KEYS         => [ qw(NAMEBSC NAMEBTS) ],
    COUNT_FILE_DESCRIPTION  => '(.*)-#-CHAN-#-I.pif',
    COUNT_FILE_KEYS         => [ qw(NAMEBSC NAMEBTS) ],
    COUNTER_TO_COUNT        => 'CHTYPE',
    COUNTER_VALUES_TO_MATCH => [ qw(MAINBCCH SDCCH TCHFULL) ],
    OUTPUT_COUNTER_NAMES    =>
        [ qw(CHTYPE_MAINBCCH CHTYPE_SDCCH CHTYPE_TCHFULL) ],
    DEFAULT_NULL_VALUE      => "NULL",
    PRODUCE_PIF             => 'TRUE',
    OUTPUT_FORMAT           => 'LIF_Writer',
}
```

3.2 COUNTER_MANIP

The COUNTER_MANIP rule is used to extract and create new counter names and values based on current counter names and values. It can be used in cases where the counter name contains performance data, and is required as a counter value for loading.

It extracts the matching counter names and their values and outputs them under the configured new counter names and values.

In the case of the Siemens BSS Gateway, its used to extract processor numbers contained in the counter names.

3.2.1 Rule Configuration

This section describes the non-standard rule configuration entries.

- COUNTER_NAMES_TO_EXTRACT: The list of counter names to match in the PIF data. These are the counters that will be extracted. Each entry will map to a new row in the PIF output.
```
COUNTER_NAMES_TO_EXTRACT => [ 'C0_13_[12]$','C0_13_[34]$']
```

- OUTPUT_COUNTER_PREFIX: This specifies the portion of the counters matched in COUNTER_NAMES_TO_EXTRACT to output.
```
OUTPUT_COUNTER_PREFIX    => '^(C0_13_\d+)',
```

- CREATE_UNIQUE_COUNTERS: If set to true the rule will create unique counter names and values by appending a number to the end of the new counter names and values.
```
CREATE_UNIQUE_COUNTERS   => 'True',
```

- NEW_NAME_PREPEND: The string to use as the first portion of the counter name for outputting the names of the COUNTER_NAMES_TO_EXTRACT counters.
```
NEW_NAME_PREPEND         => 'OBJ_ID',
```

- NEW_VALUE_PREPEND: The string to use as the first portion of the counter name for outputting the values of the COUNTER_NAMES_TO_EXTRACT counters.
```
NEW_VALUE_PREPEND        => 'VALUE',
```

- OUTPUT_BLOCK_NAME: The name to use for the output blocks.
```
OUTPUT_BLOCK_NAME        => 'PROCSPLIT',
```

3.2.2 Sample Usage

Given the input PIF:

```
##START|BLOCK
C0_13_1|C0_13_2|C0_13_3|C0_13_4|C0_13_5|C0_13_6
0.00|1.00|3.00|6.00|6.00|6.00
0.00|3.00|7.00|4.00|1.00|9.00
##END|BLOCK
```

The following processing is performed:

- The groups of counters C0_13_[12], C0_13_[34] and C0_13_[56] will be split onto separate lines.
- The groups of counter names are output as counter values, under the new counter names OBJ_ID1 and OBJ_ID2
- The values of the counters will be output in VALUE1 and VALUE2.
- The output block will be called PROCSPLIT.

This produces the output data below:

```
##START|PROCSPLIT
VALUE1|VALUE2|OBJ_ID1|OBJ_ID2
0.00|1.00|C0_13_1|C0_13_2
3.00|6.00|C0_13_3|C0_13_4
6.00|6.00|C0_13_5|C0_13_6
0.00|3.00|C0_13_1|C0_13_2
7.00|4.00|C0_13_3|C0_13_4
1.00|9.00|C0_13_5|C0_13_6
##END|PROCSPLIT
```

The configuration for this rule:

```
{
    RULE_TYPE                    => 'COUNTER_MANIP',
    RULE_DESC                    => 'Manip PROC measurements',
    INPUT_FILE_DESCRIPTION       => '.*-#-I.pif',
    COUNTER_NAMES_TO_EXTRACT     =>
        [ 'C0_13_[12]$', 'C0_13_[34]$', 'C0_13_[56]$'],
    OUTPUT_COUNTER_PREFIX        => '^(C0_13_\d+)',
    CREATE_UNIQUE_COUNTERS       => 'True',
    NEW_NAME_PREPEND             => 'OBJ_ID',
    NEW_VALUE_PREPEND            => 'VALUE',
    PRODUCE_PIF                  => "True",
    OUTPUT_BLOCK_NAME            => 'PROCSPLIT',
    PRODUCE_LIF                  => 0,
}
```

4. Tech Pack Support

The Gateway Configuration available with the Tech Pack supports Siemens BSS BR8 data version.