Siemens NSS Gateway User Guide

Gateway Release: 3.4.0

Date: 16 April 2008

Contents

CONTENTS 2

REFERENCES 3

GLOSSARY 3

PREFACE 4

ABOUT THIS GUIDE 4

1. OVERVIEW 5

1.1 THE GATEWAY FRAMEWORK 5

1.2 SIEMENS NSS GATEWAY KIT OVERVIEW 5

1.2.1 Data Types 5

1.2.2 Data Version Support 5

1.2.3 Data/File Formats 6

2. ENGINE RULES AND CONFIGURATION 7

2.1 RULE_TYPE 7

2.1.1 Rule Configuration 7

3. POST PARSER RULES AND CONFIGURATION 12

3.1 MERGELINES 12

3.2 CPXX_NORMALIZE 13

3.3 USMM_TRAFFIC 14

3.4 USSM_SSID 15

4. TECH PACK SUPPORT 15

5. INSTALLATION SPECIFIC INFORMATION 16

References

[Gateway Framework User Guide]

Glossary

LIF	Loader Input File
PIF	Parser Intermediate File
XML	EXtensible Mark-up Language

Preface

About this Guide

This guide details the Vendor specific information on the Gateway release for the Siemens NSS Gateway Kit. It contains the following information:

- *Chapter 1. Overview.* This chapter gives a brief description of the Vendor Gateway and the raw data formats 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.

1. Overview

1.1 The Gateway Framework

The Siemens NSS Gateway Kit uses the Gateway Framework as a container for the execution of its engine and post parser stages. The Gateway Framework and Siemens NSS Gateway Kit are de-coupled 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 Siemens NSS Gateway Kit and Post Parser rules for of data transformation
- Intermediate (PIF) and output data (LIF,CSV,XML) storage and management
- logging utilities
- cleanup and crash recovery
- statistics gathering

The Siemens NSS Gateway Kit simply plugs into the Gateway Framework and provides the functionality to parse the Siemens NSS ASCII format.

More information on the standard Gateway configuration is contained in the Gateway Framework User Guide.

Only Siemens NSS Gateway Kit configuration details will be described in this document.

1.2 Siemens NSS Gateway Kit Overview

1.2.1 Data Types

The *Siemens NSS Gateway Kit* can be configured to parse the Siemens NSS XML format.

1.2.2 Data Version Support

These are examples of systems currently using the Siemens NSS Gateway Kit

Vendor Subsystem	Release	Format
Siemens NSS	SR13, SR12	Variable-Width ASCII

1.2.3 Data/File Formats

The *Siemens NSS Gateway Kit* was developed to support the proprietary variable-width ASCII format of Siemens NSS based raw files.

The following measurement types are supported in the TP, which are from the byte positioning ascii format.

Measurement Group	**Measurement identification in filename**
VLR	USVM
HLR	USHM
MSC	USMM
AC	USAM
SSERV	USSM
EIR	USEM
IWE	USWM
LOCAREA	USLM
INTCELL (INTCREL)	USIM
INTCELL (HOTRFS)	USTM
MGW	USGW
Traffic Destination data	TS.DEST.xxx
Coordination Processor	TS.CPXX.xxx
Trunk Group Traffic	TS.TGRP.xxx
Announcement Group	TS.ANGR.xxx
CCS7 Traffic	TS.C7DM.xxx, TS.C7GO.xxx, TS.C7LL.xxx, TS.C7LS.xxx, TS.C7MU.xxx, TS.C7RS.xxx
Line Trunk Group	TS.LTGX.xxx
PBX	TS.PBXX.xxx
DLU	TS.DLUX.xxx
Remote Switching Unit	TS.RSUX.xxx

2. Engine Rules and Configuration

Each file format you want to parse with the Siemens NSS Gateway Kit must have its own Engine Rule entry containing the set of Engine Configuration options described below. Some Engine Configuration options are mandatory and some will be optional depending on the file format that the Engine Rule is based upon.

2.1 RULE_TYPE

Describes the Engine Rule (gateway) to use to parse a specific file format. If set to "SIEMENS_INTERFACE", the Engine Configuration tells the gateway framework to use the siemens nss gateway engine. After this is set correctly the other Engine Configuration Options listed below then apply to the siemens nss gateway only.

Any Single Rule Type entry calls a specific gateway engine to convert a particular input file format to the parser intermediate format (PIF). Each Configuration will be different for different input formats.

2.1.1 Rule Configuration

The following details the vendor specific rule entries for the Siemens Interface engine rule.

2.1.1.1 Mandatory Configuration Entries

- RULE_TYPE:

```
RULE_TYPE   =>    'SIEMENS_INTERFACE',
```

- RULE_DESC: Simple Free form text to describe what format this Rule Entry will cater for. Mandatory entry.

```
RULE_DESC   => '(free text) file format of this data',
```

- INPUT_FILE_DESCRIPTION: Specification for Input files in the specified property network raw data directory "IN_DIR" (e.g. '^.*.xml$') is a regular expression which denotes all files which end in the ".xml" extension which are present in the input directory. More than one set can be specified in the []'s

```
INPUT_FILE_DESCRIPTION  => [ '^PPM-\w+-\w+-\w+-\w+\.xml$' ],
```

- INPUT_DIR_DEPTH: Specification for how deep the gateway framework will search for files in directory trees in the specified property "IN_DIR". 0 is the default. Mandatory entry.

```
INPUT_DIR_DEPTH         => '0',
```

- NUMBER_OF_FILES_TO_PROCESS: Specification for how many files from "IN_DIR" the gateway framework will attempt to run through the gateway in any single run. E.g. 20 = 20 files at a time. This parameter is a tuning parameter. Mandatory entry.

```
NUMBER_OF_FILES_TO_PROCESS    => '20',
```

- ORDER_OF_FILES: Specification for what order the gateway framework will attempt to parse the data through the gateway. No ordering is fastest, oldest first is best for PM files due to NC re-parenting. Mandatory entry.

  Valid options: YOUNGEST_FIRST, OLDEST_FIRST, DIRECTORY_ORDER (comment out this configuration option to speed up the first stage of the parser at expense of data presentation contiguity to the loader). This parameter is a tuning parameter.

```
ORDER_OF_FILES    => 'YOUNGEST_FIRST',
```

- DIRECTORY_HEADER_FIELDS: Specification for the directory name or part of the directory name to appear as a counter in the PIF header.

```
DIRECTORY_HEADER_FIELDS => {
      NETWORK_ID => '\/\w+\/\w{7}_(\w+)\/\w{6}_\w+'
}
```

- FILENAME_HEADER_FIELDS: Specification for the raw filename or part of the raw filename to appear as a counter in the PIF header.

```
FILENAME_HEADER_FIELDS => {
      FILENAME => '^(.*)\.xml$',
}
```

- HEADER_FIELDS_FOR_PIF_FILENAME: Specification for value of counters in the PIF header to be appended to the initial PIF filename.

```
HEADER_FIELDS_FOR_PIF_FILENAME => [qw(NETWORK_ID REGION_ID)]
```

- DATA_FIELDS_FOR_PIF_FILENAME: Specification for value of counters in the PIF counter blocks to be appended to the initial PIF filename.

```
DATA_FIELDS_FOR_PIF_FILENAME => [qw(START_DATE START_TIME)]
```

- CONFIG_FILENAME: The location of the configuration filename for the Siemens NSS raw file.

```
CONFIG_FILENAME => "config-v13",
```

2.1.1.2 Optional Configuration Entries

This configuration file describes the layout of the raw data files presented to the parser. The configuration file is provided to the parser through a command line option.

The following general points apply to the configuration file format.

- The file is case sensitive. For example. The word `datab' is not equivalent to `DATAB'.
- The file is line orientated. Only one statement per line. Lines cannot be continued to the next line.
- All key words and options/arguments and separated by white space. Either a <space> or <tab>.
- The configuration file statement keyword must be the first thing on the line except for white space.
- Counter and block names can be no greater than 50 characters in length.
- There is no limit to how many statements are allowed in a block or configuration file.
- There must be a Named block title `HEADER'. This block describes the header block of the file and is a starting point for the engine to enter the configuration file.
- Comment and blank lines are allowed in the file. Comment lines begin with a '#'.

File types and versions are all found in the documents produced by Siemens.

The statements can be divided into two types, block and line statements. Block statements generally enclose a list of other statements and describe an action or set of actions to be undertaken on a data record. All configuration file statements are valid within any block statement, including nesting named blocks. The block statements are the Named block, LINE and REPEATEND.

Line statements describe an action to be taken when processing a data record. The line orientated statements are CONTROL, JOIN and the DATA statements.

<u>CONTROL statement</u>

Statement syntax: CONTROL buffer name1 name2 ...

where:

`buffer' is the join buffer or a counter that contains the name of the Named block where the next configuration files statements are stored. `name1 name2...' is an optional list of named blocks that are possible values for `counter' to take.

The CONTROL statement indicates that the description of the current data record is continued in the another Named block. The name of block is contained in `buffer'. If the list of acceptable named blocks is specified then the new value must be in that list. The CONTROL statement has one special case. In the Named block `HEADER', the CONTROL statement is used to specify the name of the Named block that contains the description of the data records in the rest of the file. No list of acceptable Named blocks is desirable on this CONTROL statement.

DATAB and DATABNP statements

Statement syntax: DATAB counter num_bytes data_type
DATABNP counter num_bytes data_type
where:

`counter' is the user specified counter name. `num_bytes' is the number of bytes occupied by the counter. `data_type' is the data type of the counter. Possible values are INT, REAL, STRING.

The DATAB statement specifies that the counter occupies the next `num_bytes' of the data record and is of type `data_type'. The `num_bytes' of data are to be read from the current position in the input record. The value will be output to the output file in the same order as it is read in.

There is only one difference between the DATAB and DATABN statements. The DATABN statement doesn't print anything out to the output file. It will only produce output if the -printall statement line option is specified to the engine. All the rest of this section is applicable to both statements.

DATAO and DATAON statements

Statement syntax: DATAO counter offset num_bytes data_type
DATAONP counter offset num_bytes data_type

where:
`counter' is the user specified counter name. `num_bytes' is the number of bytes occupied by the counter. `data_type' is the type of data that the counter. Possible values are INT, REAL, STRING.

The DATAO statement specifies that the counter occupies the bytes of the data record from (and including) `offset' for the next `num_bytes' bytes and is of type `data_type'. The `offset' is counted from the start of the current data record. The value will be out put to the output file in the same order as it is read in.

There is only one difference between the DATAO and DATAON statements. The DATABN statement doesn't print anything out to the output file. It will only produce output if the -printall statement line option is specified to the engine. All the rest of this section is applicable to both statements.

JOIN statement

statement syntax: JOIN buffer_name counter

where:

\`buffer_name' is the name of the join buffer to use. \`counter' is the name of the counter to be used. The JOIN statement is intended to allow the concatenation of counters to be used in the CONTROL statement. The statement joins the contents of \`counter' to the end of the join buffer named \`buffer_name'.

\`counter' must be a counter in the current statement block. It should have a unique name in the statement block. It must also appear before the JOIN statement in the state ment block.

There is only one join buffer. When another \`buffer_name' is used the existing buffer is cleared and renamed with the name \`buffer_name'. The contents of \`counter' are then copied to the buffer.

<u>LINE statement</u>

```
Statement syntax:    LINE {
                             statement list
                     } LINE
```

The LINE statement describes the contents of a record that is read from the input data file. The statement has been included to allow the processing of multi-line records and the omission of rubbish records if they occur in a regular manner (e.g. at the start of the file). The LINE statement has no effect on the output data format.

<u>REPEATEND statement</u>

```
Statement syntax:    REPEATEND {
                             statement list
                     } REPEATEND
```

The REPEATEND block repeats the list of statements in \`statement list' until it reaches the end of the current input data record. At the end of each REPEATEND block, an output record will be written to the output file.

3. Post Parser Rules and Configuration

Only standard Post Parser Rules Are Supplied. There are no vendor specific rules as the parser is not vendor specific, only the configurations of the Gateway engine are vendor specific.

3.1 MERGELINES

This rule allows you to merge two or more data lines to produce a single line of data.

You specify the lines of data you wish to merge by supplying the rule with a list of one or more counters. For each line of data the rule uses the counters to construct an identifier for each data line. Lines of data with the same identifier are then merged into one line of data by the rule.

For example, consider the following block of data.

```
##START|EXAMPLE
DATE|TIME|TRUNK_NO|CNTR1|CNTR2|CNTR3|CNTR4|CNTR5|CNTR6
31-05-98|00:00|0415|1|2|3|||
31-05-98|00:00|0415||||4|5|6
31-05-98|00:00|0500|1|2|3|||
31-05-98|00:00|0500||||4|5|6
31-05-98|01:00|0415|11|12|13|||
31-05-98|01:00|0415||||14|15|16
31-05-98|01:00|0500|11|12|13|||
31-05-98|01:00|0500||||14|15|16
31-05-98|02:00|0415|21|22|23|||
31-05-98|02:00|0415||||24|25|26
31-05-98|02:00|0500|21|22|23|||
31-05-98|02:00|0500||||24|25|26
##END|EXAMPLE
```

Let us say that in this example we want to merge the lines of data so that there is only one line of data for each unique combination of the counters DATE, TIME and TRUNK_NO.

The SIEMENS_MERGELINES rule would produce an output data block as follows:

```
##START|EXAMPLE
DATE|TIME|TRUNK_NO|CNTR1|CNTR2|CNTR3|CNTR4|CNTR5|CNTR6
31-05-98|00:00|0415|1|2|3|4|5|6
31-05-98|00:00|0500|1|2|3|4|5|6
31-05-98|01:00|0415|11|12|13|14|15|16
31-05-98|01:00|0500|11|12|13|14|15|16
31-05-98|02:00|0415|21|22|23|24|25|26
31-05-98|02:00|0500|21|22|23|24|25|26
##END|EXAMPLE
```

You can configure this rule to produce output files in two formats, the Parser Intermediate Format (PIF) and the Loader Format (LIF).

You can chose to produce output files in one or both of the formats from one rule instance. Parser Intermediate format files produced will have a file-name that ends with '-#-SM.pif'. Loader format files produced will have a filename that ends with '-#-SM.lif'.

The configuration options available for this module are as follows:

```
RULE_TYPE                'SIEMENS_MERGELINES' - The string the parser
                         uses to identify the type of this rule.
RULE_NAME                A string describing what the rule does.  This
                         string will be printed to the Audit/Log files.
INPUT_FILE_DESCRIPTION   A regular expression or a list of regular
                         expressions that describe the files to be
                         processed by this rule.
COUNTERS_TO_SORT_ON      A list of counters that is used to determine
                         which lines of data are merged together.
OUTPUT_BLOCK_NAME        The block name to use in the output files
OUTPUT_FILENAME_START    A prefix that the output file name will
                         start with.
REDUNDANT_COUNTERS       A list of counters to delete from the
                         resultant output.
PRODUCE_PIF = 'True'     Will produce the output as a PIF file
            = 0          Will not produce a PIF file
PRODUCE_LIF = 'True'     Will produce the output as a LIF file
            = 0          Will not produce a LIF file
```

Example:

```
    {
        'RULE_TYPE' => 'SIEMENS_MERGELINES',
        'RULE_NAME' => 'Trunk Group Statistics',
        'INPUT_FILE_DESCRIPTION' => [ "^TGRP.*"],
        'COUNTERS_TO_SORT_ON' => ['DATE', 'TIME', 'TGNO',],
        'PRODUCE_PIF' => 'True',
        'PRODUCE_LIF' => 'True',
    },
```

The following points should be noted about the example shown above:

- Only files with names starting with 'TGRP' will processed by this rule.
- The identifier for each line of data will be constructed from the counters DATE, TIME and TGNO.
- The output will be produced as a loader input file and a parser intermediate file.

3.2 CPXX_NORMALIZE

This rule has been designed to handle the special processing required for the CPXX measurement results. It MUST NOT be used for any other purpose.

This rule has two main function. The process that reports the CPXX measurement, can be configured to report at intervals smaller that the standard 15 minutes. Refer to the Siemens Documentation for details.

This rule sums the statistics so that there is one statistic reported per processor per 15 minute measurement period. The second major function this rule has is to provide a set of statistics that summarise the performance of all the processors in the switch.

The statistics reported have different units to those described in the Siemens documentation. To get back to the original units the the report statistics must be divided by the number of statistics that were added together to obtain the result.

The configuration options available for this rule are as follows:

```
INPUT_FILE_DESCRIPTION  a regular expression or a list of regular
                        expressions describing the names of files that
                        this rule should be applied to.
OUTPUT_BLOCK_NAME       The name that will be used for the block/section
                        name in the loader file
OUTPUT_FILENAME_START   A prefix that the output file name will
                        start with.
RULE_NAME               a string describing what the rule does.  This
                        string will be printed to the Audit/Log files.
```

Example:

```
        {'RULE_TYPE' => 'CPXX_NORMALIZE',
         'RULE_NAME' => 'CPXX Files',
         'INPUT_FILE_DESCRIPTION' => "CPXX.*.pif",
         'OUTPUT_BLOCK_NAME' => 'PROC',
        },
```

This example rule would process all files that names match 'CPXX.*.pif'. The resultant loader input file would have data blocks with the name 'PROC'.

3.3 USMM_TRAFFIC

This rule has been designed to handle the special processing required for the USMM measurement results. It should only be used to process the USMM measurement results. It MUST NOT be used for any other purpose.

For each data record in the USMM file this rule does the following functions:

- it groups the counters that are specific to the individual mobile traffic types into seperate blocks with the name 'TRAFF'. It removes the traffic type name off the end of all counter names in these blocks

The configuration options available for this rule are as follows:

```
INPUT_FILE_DESCRIPTION  A list of file types that are to be joined.
                        Each file description is a regular expression.
                        Each file description must have a set of
                        brackets in it enclosing a part of the regular
                        expression is common with the other files
                        that it is to be joined with.  Only one set
                        of brackets is allowed in each file name.
OUTPUT_BLOCK_NAME       The name that will be used for the block/section
                        name in the loader file
OUTPUT_FILENAME_START   A prefix that the output file name will
```

```
                        start with.
```

Example:

```
      {
        RULE_TYPE => 'USMM_TRAFFIC',
        INPUT_FILE_DESCRIPTION => "USMM-#-1929-#-(.*)-#-[0-9]+-#-I.pif",
        OUTPUT_BLOCK_NAME => ‘TRAFF’,
      },
```

This example processes the counters from the USMM. The options in this example are the ones that should be used for processing these file types in most cases.

3.4 USSM_SSID

This rule has been designed to handle the special processing required for the USSM measurement results. It should only be used to process the USSM measurement results. It MUST NOT be used for any other purpose.

This rule splits the counters up into service types. It produces a loader block containing the counters specific to a service and a counter specifying the service abbriviation.

The configuration options available for this rule are as follows:

INPUT_FILE_DESCRIPTION	a regular expression or a list of regular expressions describing the names of files that this rule should be applied to.
OUTPUT_BLOCK_NAME	The name that will be used for the block/section name in the loader file
OUTPUT_FILENAME_START	A prefix that the output file name will start with.
RULE_NAME	a string describing what the rule does. This string will be printed to the Audit/Log files.

Example:

```
       {
        'RULE_TYPE' => 'USSM_SSID',
        'RULE_NAME' => 'USSM Files',
        'INPUT_FILE_DESCRIPTION' => "USSM.*.pif",
        'OUTPUT_BLOCK_NAME' => 'USSM_SSID',
       },
```

This example rule would process all files that names match 'USSM.*.pif'. The resultant loader input file would have the data blocks with the name 'USSM_SSID.

4. Tech Pack Support

Tech pack support is included in the Siemens NSS Gateway for the following Performance Manager solutions.

- GSM Siemens NSS SR13

The EngineConfig.pm and UserConfig.pm configuration files for these Tech Packs are located in the tech pack.

5. Installation Specific Information

None.