Nortel BSS Gateway User Guide

Release: 3.4.0

Date: April 3, 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.
Nortel OCD V15.1	Wireless Service Provider Solutions Observation Counter Dictionary - PE/DCL/DD/0125 15.107/EN - Standard July 2005

Glossary

BSS	Base Station Subsystem
PIF	Parser Intermediate Format
LIF	Loader Input Format
GPO	General Permanent Observation
OFS	Observation Fast Statistics
OGS	Observation General Statistics
SDO	Server de Donnees de l’OMC-R (OMC-R Data Server)

Preface

About this Guide

This guide details the Vendor specific information on the Gateway release for Nortel 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 4: 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 Nortel BSS Gateway, also referred to as the Vendor 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 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 Nortel BSS Gateway Overview

1.2.1 Network Details

The Nortel SDO provides OMC-R data records and radio network configuration parameters reports in an ASCII readable format. The SDO produces data using a defined directory structure and file naming convention. The SDO has the ability to produce data in formats with a high degree of configurability which may yield many different SDO configurations for customers' data.

1.2.2 Data Types

The data pertinent to IBM service assurance applications is as follows:

- Performance data, referred to by Nortel as *General Permanent Observations* or *Observation data* (GPO, OFS files)

- Configuration data, referred to by Nortel as *Radio Network Configuration Parameters* (NETWORK files).

1.2.3 Data Version Support

Data versions supported by this vendor gateway are v13, v14.3, and v15 Nortel BSS raw and configuration data.

1.2.4 Data/File Format

The high degree of configurability available in the Nortel SDO may yield data files of many different SDO configurations. The configuration of a data file is defined in an associated format file. For this release of the Vendor Gateway for Nortel**, it is assumed that the format of a set of data is guaranteed to be stable whilst the Gateway is in operation.** It is feasible that several format files for given data appear in a same day when the SDO configuration has been modified during the day. Since format files are time stamped it is possible to distinguish the format file associated to each data file. However this case of an unstable SDO configuration is neglected for this release.

This constraint does not exclude the Gateway from parsing data of different formats, but it imposes that in this case, different rules of the Vendor Interface are necessary for each different format held.

Appendix A illustrates subsections of example data files with their associated format files. The classification of the format of the raw data is defined in its format file. The terminology used in the format files is briefly explained below. For a more detailed discussion the reader should refer to the Nortel documentation.

1.2.4.1 Performance Data Layout

The performance data format files have 3 or 4 sections

1. Common parameters
   - colSep, column separator
   - lineSep, record/line separator
   - fieldLength, parameter value is either "VARIABLE" or "FIXED
2. Object instance format describes all the attributes associated with the objects:
   objectFormat, object Length, classIdSep, classSep, className (see section **Error! Reference source not found.** for further details)
3. Observation format
   - counterName, parameter value is either "Q3" or "CUSTOM"

Q3 are counter names such as "hoRequired"
CUSTOM are counter names such as "1079 000 00 CUM"
The relationships between Q3 and CUSTOM counters are in file sdo_cntr_list.
i.e. this file has lines of the form:

```
hoRequired                # "1064 000  0 CUM"
```

There are also synthetic counters and this is defined in bss_nmc_formulas.dat (see below)

- **counterType**, parameter value is either "BOTH" or "RAW" or "SYNTHETIC"

  Synthetic counters are defined in the file bss_nmc_formulas.dat, this file has data of the form:

  ```
  hoRequestOutgoing bts = hoRequestOutgoingIntraBss +
                          hoRequestOutgoingInterBss +
                          hoRequestIntraBts;
  ```

- **factorized**, parameter value is either "ON" or "OFF"

  factorized=OFF is the V11 format e.g.

  ```
  bts 46/3/2,hoRequestIntraBts,24
  bts 46/3/2,hoExtractionIntraBts,20
  bts 46/3/3,hoRequestIntraBts,45
  bts 46/3/3,hoExtractionIntraBts,0
  ```

  factorized=ON is an optional new V12 format e.g.

  ```
  hoRequestIntraBts,hoExtractionIntraBts
  bts 46/3/2,24,20
  bts 46/3/3,45,0
  ```

- **validityField**, parameter value is either "ON" or "OFF"

  validityField =ON there is an additional field for each counter indicating if the counter is significant or not:
  If using factorized=OFF then the format with a validity field is:

  ```
  bts 46/3/2,hoRequestIntraBts,100,24
  bts 46/3/2,hoExtractionIntraBts,100,20
  bts 46/3/3,hoRequestIntraBts,100,45
  bts 46/3/3,hoExtractionIntraBts,0,0
  ```

  If using factorized=OFF then the format with a validity field is:

  ```
  hoRequestIntraBts,hoExtractionIntraBts
  bts 46/3/2,100,24,100,20
  bts 46/3/3,100,45,0,0
  ```

4. COLUMNS,

This section is not present for the factorized format (i.e.factorised,"ON") as it is impossible to describe the list of columns.
Otherwise it specifies the length of attributes: (-1 if fieldLength,"VARIABLE"):

```
        objectInstance, countername, counterValue
```

1.2.4.2 Properties of raw data files

There are two types of raw data files

- per record data
- daily data reporting for a 24 hourly periods

The raw data files contains information about two types of values:

- Object Instance Parameters
- Counters

1.2.4.2.1 Object Instance Parameters

The object instance parameter is a record identifier i.e. it is an identifier of an object to which a set of counter values belongs. The format file defines 5 specifications associated with the object instance's format:

- classIdSep,
- classSep,
- className
- objectFormat,
- objectLength,

A complete object instance definition is a sequence of (class, identifier) couples. The **class/id separator** (referred to as classIdSep in format file) is used to separate the class part and the identifier parts of the couple. The **class separator** (referred to as classSep in format file) is used to separate the (class, identifier) couples.

For example, "BSC 2/BSM 3/BTS 4" is an object of 3 couples where '/' is the class separator, ' ' is the class/id separator.

The last couple of the object may be used to unambiguously identify the **class** of the object instance, i.e. for the example "BSC 2/BSM 3/BTS 4" the class is "BTS".

This example employs the "COMPACT" **class name** (referred to as className in format file). The same example can equivalently be shown using the "FULL NAME" class name, i.e.

"bsc 2/btsSiteManager 3/bts 4" where the class of object is "bts".

The correspondence between COMPACT and FULLNAME of all the classes is given in **Error! Reference source not found.**.

COMPACT className	***FULL NAME className***
BSC	Bsc
BSM	btsSiteManager
BTS	Bts
PCM	pcmCircuit
ADJ	adjacentCellHandoverNortel
TRZ	transceiverZone

Table 1 Correspondence between COMPACT and FULL NAME class names

The **object format** (referred to as objectFormat in format file) of the (class, identifier) couples can take one of 3 formats, (EXTENDED, COMPACT-FIXED, COMPACT-VARIABLE). The example used thus far shows the object format as EXTENDED, an illustration of this object in the other formats is shown in **Error! Reference source not found.**.

Format	***Example***
EXTENDED	BSC 2/BSM 3/BTS 4
COMPACT-FIXED	BTS 2/3/4/-1
COMPACT-VARIABLE	BTS 2/3/4

Table 2 Example of object formats.

Note: the "-1" value in COMPACT-FIXED stands for meaningless

The **length of the object** (referred to as objectLength in format file) may be either VARIABLE or FIXED length.
The following table is a summary of the allowable values format parameters associated with the object instance:

parameter	***allowable values***
classIdSep	any character
classSep	any character
className	COMPACT, FULL NAME
objectFormat	EXTENDED, COMPACT-FIXED, COMPACT-VARIABLE
objectLength	FIXED, VARIABLE

Table 3 Allowable values for format parameters associated with object instance

1.2.4.2.2 Counters

As explained in **Error! Reference source not found.**, there are two types of counter names Q3 and CUSTOM.
The data file may have 1, 2, 24, or 48 values associated with each counter name. Table 4 shows a summary of how many values are

associated with each counter name and the number of fields expected per "line" of data for each different format.

factorized	validityField	RAWorSUM	Number of values asscoiated with counter name	Number of fields per "line"
ON	OFF	raw	1	N+1
OFF	OFF	raw	1	1+2
ON	ON	raw	2	2*N+1
OFF	ON	raw	2	2+2
ON	OFF	sum	24	24*N +1
OFF	OFF	sum	24	24 +2
ON	ON	sum	48	24*2*N+1
OFF	ON	sum	48	24*2+2

Table 4 Number of values associated with each counter name

Note: When factorized=ON each line has the object instance parameter value as well as the counter values (hence the +1); in this table it is assumed that there are N counter names per "line". When factorized=OFF each line has the object instance parameter value and the counter name (hence the +2).

1.2.4.3 Configuration Data Layout

The configuration data format files have 2 sections
Common parameters

- colSep, column separator
- lineSep, record/line separator
- fieldLength, parameter value is either "VARIABLE" or "FIXED COLUMNS,

Per line specification of the column name and column length (-1 if fieldLength,"VARIABLE") of each counter in the associated configuration data.
In accordance to the specification in the associated format file, the configuration data files is "lines" of values separated by delimiter "colSep".

1.2.4.4 File naming specification

As described in Nortel OMC-R Data server functional specification, the file naming convention for performance and the configuration data is as follows:

- Performance data:

Per record file –

/SDO/data/obs/raw/<Date>/<Network>/<BSC>/aOFS_<objClass>_<dUTC><HHMN>

/SDO/data/obs/raw/<Date>/<Network>/<BSC>/aGPO_<objClass>_<dUTC><HHMN>

Daily file –

/SDO/data/obs/sum/<Network>/<BSC>/aOFS_<objClass>.<Date>
/SDO/data/obs/sum/<Network>/<BSC>/aGPO_<objClass>.<Date>

- Configuration data:

Per record file –

/SDO/data/network/<Date>/<Network>/aNETWORK_<configParameter>.<HHMN>

where

<Value>	*Description*	*Format*	*Example*
<Date>	date	YYYYMMDD where YYYY=year, MN=month, DD=day	19991225
<Network>	network number of BSS	XXX	001
<BSC>	bsc number	XXX	019
<objClass>	object class	XXX (e.g BSC, BSM, BTS, ADJ, TRZ, PCM)	BSC
<configParameter>	radio network parameters	any string (e.g. BTS, SALGOBTS, RXDISWTS, RXLEVWTS, RXQUALWTS, PLMN_BTS, ADJCHO, ADJCR, MAFREQHSYST, POWERCM, HANDOCM, TRANSCVE, TRANSCV, TRANSCVZ, FHS, MAFHS, BSC, SIGP, SALGOSIGP, SIGL, SIGLS, PCMCIRCUIT, LAPDL, XTP, TRANSCD, TRANSCDB, ALGOUSED, BTSSM, PCMBTSSM, BSCMDINT, BSCCNTRLIST, COUNTELEM)	BTS
<dUTC>	difference between local time and GMT	SHHMN where S=P (plus) or M (minus), HH=hour, MN=minute	P0200
<HHMN>	time	HH=hour, MN=minute	1234

Table 5 Descriptions of parameters used in filenames

Associated with these 2 types of data (performance and configuration) are format files, these respectively have the following naming conventions:

- Performance data:

format_obs.YYYYMMDD_HHMN

- Configuration data:

format_<configParameter>.YYYYMMDD_HHMN

The PIF will be output with a name in the following format where '-#-' is used as a delimiter :

```
      <original filename > -#-<value of block name> <counter> -#-I.pif
```

where <counter> will either be nonexistent (as in the case when the first file of <original filename> is processed) or it will be an integer (1,2,3,..) (as in the case where a file with same name has file already parsed and whose the resulting PIF still exists in the intermediate directory)

1.2.4.5 File naming specification for V16 (ONL)

The following describes the file naming convention used for Nortel V16 (ONL):

- Performance data:

Per record file –

$INPUT_DIR/raw/<SDO>.<FileType>_<objClass>.<FileID>.<FileDate>.<FileTime>
$INPUT_DIR/raw/<SDO>.<FileType>_<objClass>.<FileID>.<FileDate>.<FileTime>.Z

- Configuration data:

Per record file –

$INPUT_DIR/<SDO>.<objClass>.CNF.<FileDate>
$INPUT_DIR/<SDO>.<objClass>.CNF.<FileDate>.Z

where

<Value>	*Description*	*Format*	*Example*
<SDO>	OMC-R Data Server	String value	OMC3
<FileDate>	date	DDMMYYYY where YYYY=year, MN=month, DD=day	02072007
<FileID>	BSC, BSM, BTS, or some similar ID number	XXX	001
<objClass>	object class	XXX (e.g BSC, BSM, BTS, ADJ, TRZ, PCM) or X (abbreviated) (e.g. B, A, T, P)	BSC or B
<FileTime>	time	HHMN HH=hour, MN=minute	1235

Table 6 Descriptions of parameters used in filenames

UserConfig.pm for v16 tech pack (ONL) has been configured to look for raw files under a subdirectory called 'raw' (or 'sum'). The following subdirectory path must be created and raw performance files must be placed in this directory for processing:
$INPUT_DIR/raw

Associated with these 2 types of data (performance and configuration) are format files. These respectively have the following naming conventions:

- Configuration data:

  format_<configParameter>.YYYYMMDD_HHMN

Format files **must** be placed in the same directory as the associated performance and configuration files.
The PIF will be output with a name in the following format where '-#-' is used as a delimiter :

```
      <original filename > -#-<value of block name> <counter> -#-I.pif
```

where <counter> will either be nonexistent (as in the case when the first file of <original filename> is processed) or it will be an integer (1,2,3,..) (as in the case where a file with same name has file already parsed and whose the resulting PIF still exists in the intermediate directory)

1.2.5 Architectural extensions

No external tools were used to parse the Nortel performance data.

2. Engine Rules and Configuration

There are 2 vendor interfaces for the Nortel SDO V12 BSS Gateway, one to parse raw data (NORBSS_data) and the other to parse configuration data (NORBSS_config).
For effective parsing up to 3 other types of file also need to be referenced:

1. format files for configuration data
2. format files for raw data
3. counter list configuration (only if Q3 counter names need to be converted)

The former is used by the vendor interface NORBSS_config, whereas the latter two may be used be the vendor interface NORBSS_data.

2.1 PreParserConfig

The configuration file PreParserConfig.pm allows the configuration of custom pre-parsers that can be applied in EngineConfig.pm. Each pre-parser is created as a piece of CODE in a subroutine. It will enable miscellaneous tasks of manipulating raw data prior to actual data processing. Example:

```
EXAMPLE => sub {
    my $memoryfile = shift;
    foreach (@{$memoryfile}) { $_ = "newfiled,".$_; }
        print @{$memoryfile};
        return 0;
    }
```

The example pre-parser above adds a new field named ‘newfield’ to each row in the raw input file before the EngineConfig parses the data.

2.2 NORBSS_data

This vendor interface parses raw data files into PIF format. Essentially only raw data files are parsed, but the process requires information from the raw data’s associated format file.

2.2.1 Rule Configuration

To evoke the NORBSS_data module the configuration option 'RULE_TYPE' must be set to 'NORBSS_data'. The configuration options 'RULE_TYPE', 'RULE_DESC', 'INPUT_FILE_DESCRIPTION', 'INPUT_DIR_DEPTH', 'FILENAME_HEADER_FIELDS', ‘DIRECTORY_HEADER_FIELDS’ are common to all productised parsers and are fully described in Gateways Framework User Guide; all other configuration options listed are specific to the NORBSS_data module.

- TIME, DATE, RAWorSUM: These are referred to as the module's 'musthave' options, their configuration is mandatory and must be in the form of a 2 element ARRAY. These options are meant to be used to specify respectively the time, date and whether data is raw of summary.

  The second component of the array must be either 'LITERAL', 'EXTRACT' or EXTRACT_DEL'. The configuration 'LITERAL' implies that the musthave option will take the literal value specified in the first component of its array. If 'EXTRACT' or 'EXTRACT_DEL' is configured then the first component of the array must be a key in the FILENAME_HEADER_FIELDS or DIRECTORY_HEADER_FIELDS configuration hash, i.e. the value of the 'musthave' parameter will be extracted from the filename or directory name by this key.

In the above example, the musthave option 'TIME' will be given the literal value '12:00'; whereas 'RAWorSUM' will be the string 'raw' or 'sum' extracted from the filename by the key rawORsum and there will be no countername rawORsum in the PIF header block; whereas 'DATE' will have the value extracted from the last 8 digits of the filename by the key 'FileDate' and the countername 'FileDate' will also be listed in the PIF header block.

```
TIME     => ['FileTime', 'EXTRACT_DEL'],
DATE     => ['FileDate', 'EXTRACT_DEL'],
RAWorSUM => ['rawORsum', 'EXTRACT_DEL'],
```

The values above may be extracted from either the raw filename or the directory path to the raw file. In the case where the gateway is configured to extract the compulsory attributes listed above from the 'DIRECTORY_HEADER_FIELDS', the raw files must be placed within a relevant subdirectory path. The exact path and naming convention is subject to the gateway configuration. For example:

Inside EngineConfig.pm

```
DIRECTORY_HEADER_FIELDS => {
             rawORsum => '(raw|sum)',
},
```

The raw files must be placed under a 'raw' or 'sum' directory inside the INPUT_DIR path.

```
$INPUT_DIR/raw/<raw-file>
$INPUT_DIR/sum/<raw-file>
```

- FMT_DEFINED: This is a mandatory HASH option, it described the specification of the format parameters of the data. It can have the following components:

  Mandatory components:

  - colSep: a delimiter used to distinguish data fields; allowable values : \W+
  - lineSep: a delimiter used to distinguish data fields; allowable values : \W+
  - objectFormat: a defined Nortel format for the class name; allowable values : EXTENDED, COMPACT-FIXED, COMPACT-VARIABLE
  - classIdSep: a delimiter used to separate the class and the identifier in class -identifier couples; allowable values : \W+
  - classSep: a delimiter used to separate class-identifer couples; allowable values : \W+

- className: a defined Nortel format for object instance; allowable values : COMPACT, FULL NAME
- counterName: a defined Nortel format for type of column names; allowable values : Q3, CUSTOM
- validityField: a defined Nortel format specifing whether validity value is associated with each field; allowable values : ON, OFF
- factorised: a defined Nortel format of file layout; allowable values : ON, OFF

Optional components:

- fieldLength: description of lengths of columns; allowable values : FIXED, VARIABLE
- objectLength: a description of length of object instance; allowable values : FIXED, VARIABLE
- countertype: a defined Nortel format for types of counters contained within data; allowable values : RAW, SYNTHETIC, BOTH
- numberCols: number of fields on a "line"; allowable values : 0 set for factorized ON; 3,4,26,50 set for factorized OFF

```
FMT_DEFINED => {
      colSep        => ',',
      lineSep       => '\n',
      fieldLength   => 'VARIABLE',
      objectFormat  => 'COMPACT-VARIABLE',
      objectLength  => 'VARIABLE',
      classIdSep    => ' ',
      classSep      => '/',
      className     => 'COMPACT',
      counterName   => 'CUSTOM',
      validityField => 'OFF',
      factorized    =>  'ON',
      counterType   => 'RAW',
},
```

- VALIDITY: This is an optional SCALAR configuration option. It is intended to be used when parsing validityField ON data. The scalar value VALIDITY can be set to any integer between -1 and 100 (inclusive). If VALIDITY is set to -1, all the data will be parsed including the validity values that will have counter names with suffix '_V'. If VALIDITY is set to a value between 0 and 100, then it is interpreted as a threshold value, only counter values with validity values above or equal to this threshold will be parsed to the PIF;

those below the threshold will have a blank counter value in the PIF. The default value is 100.

```
VALIDTIY      => -1,
```

- DURATION: This is an optional SCALAR configuration option. It is intended to be used to specify in PIF header the duration of the data it is parsing. The default value is 3600.

```
DURATION      => '3600',
```

- Q3COUNTERS: This is an optional HASH configuration option. It is intended to be used to lookup CUSTOM counter names so they can replace the Q3 counter names present in the data. The hash option has 3 mandatory components:
- CLIST_PATH: a SCALAR, specifying the full path to the lookup file sdo_cntr_list.cfg. This file must be readable and writable.
- FORMAT: a SCALAR, specifying the pattern of the lines of the file to be matched. The pattern should have at least 2 tagged expressions.
- COLUMNS: a 2 component ARRAY with the components which are the mutually exclusive strings Q3 and CUSTOM. The first component will label the data extracted from $1 (tagged expression in FORMAT), the second component will label the data extracted form $2. e.g. ['Q3', 'CUSTOM'] $1 refers to Q3 counters, $2 refers to CUSTOM counters and ['CUSTOM','Q3'] refers to the vice versa example.

```
Q3COUNTERS => {
      CLIST_PATH => '/parser_dir/sdo_cntrs_list.cfg',
      FORMAT     => '^(\w+)\s*#\s\"(.*)\"',
      COLUMNS    => ['Q3', 'CUSTOM']
}
```

- COUNTER_MANIP: This is an optional hash which defines the manipulation of the counter names to remove spaces and rearrange the format of the name. It consists of 3 entries:
  - COUNTER_MATCH: An RE used to match and extract the counter
  - NEW_COUNTER_NAME: An array containing the order of the counters in the new counter name. Note these are the order they are extracted from the RE match. So above the 4th match is moved to the front etc.

  - COUNTER_JOIN_CHAR: The character to use to join the entries matched to create the new counter name

In the configuration example below, the counter name "1000 000 0 CUM" would be manipulated to give tagged expressions $1=1000, $2=000, $3=0, $4=CUM, hence the counter name would be manipulated to be "CUM_1000_000_0"

```
COUNTER_MANIP => {
      COUNTER_MATCH     => '^(\S+)\s+(\S+)\s+(\S+)\s+(\S+)$',
      NEW_COUNTER_NAME  => [3,0,1,2],
      COUNTER_JOIN_CHAR => '_',
},
```

- FACTORIZED_DATA_HEADER: Hash which describes the format of the data headers expected in the factorized data header section

```
FACTORIZED_DATA_HEADER => {
      LEADTEXT => '\[CLASS_(\w+)\]',
      COL_HDR  => '\[COUNTER_LIST\]',
      VAL_HDR  => '\[COUNTER_VALUE\]'
},
```

- SUFFIX: The suffix used on counter names to pass validity values to PIF when VALIDITY =-1 option required

```
SUFFIX => 'V',
```

- CLASS_LIST_INSTANCE_MAP: List of object instance components for a given object class.

```
CLASS_LIST_INSTANCE_MAP => {
      BSC => ['BSC'],
}
```

- COMPACT_TO_FULL_MAP: Map of corresspondences between compact and full names

```
COMPACT_TO_FULL_MAP => {
      FRM => 'framer',
      CCH =>'cch',
      LPR => 'logicalProcessor',
}
```

- HEADER_DATA_RECORD_PROCESSING: This is an optional configuration entry for the Engine rule, which consists of a piece of CODE in a sub-routine. It will enable miscellaneous tasks of manipulating the header and data records before it is being output to the PIF files. This entry may reduce the use of certain post parsing rules. Example:

  In EngineConfig.pm :

```
HEADER_DATA_RECORD_PROCESSING => sub {
      my ($block_name, $header_ref, $data_ref) = @_;
```

```
        # Copy NODEID from header to data
        if ($h_ref->{NODEID}) {
              $d_ref->{NODEID} = $h_ref->{NODEID};
        }

        # Replace whitespace in NAME with underscore
        $d_ref->{NAME} =~ s/ /_/g;

        return 0;
  },
```

- DATE_INPUT_FORMAT: This is an optional configuration entry defining the format of the header date string. Currently, the following formats are supported:
  - DDMMYYYY
  - DDYYYYMM
  - MMDDYYYY
  - MMYYYYDD
  - YYYYDDMM
  - YYYYMMDD

Example:

In EngineConfig.pm :

```
DATE_INPUT_FORMAT => 'MMDDYYY'
```

If the DATE value is '12112007', then the date components will be read as:

```
DATE = 11, Month = 12, Year = 2007
```

- PRE_PARSE: An optional hash configuration defining a pre-parser process to apply to the raw input prior to processing and transferring the output to the result file.

  This option has two mandatory entries:
  - PARSER_NAME: The name of the pre-parser to be used. This pre-parser must exist in the PreParserConfig.pm configuration file.
  - PARSER_ARGS: An optional hash configuration defining the arguments that are to be passed to the pre-parser.

In EngineConfig.pm :

```
PRE_PARSE => {
    PARSER_NAME => 'ONL',
    PARSER_ARGS => {
        SHORT_OBJ_NAME => 1,
    },
},
```

The pre-parser to be used is 'ONL' and the argument 'SHORT_OBJ_NAME' with a value of '1' will be passed to the pre-parser along with the actual raw file data. It is up to the pre-parser to interpret and deal with the arguments that are passed. Different pre-parsers may require different PARSER_ARGS to be set.

- HEADER_INFO_TO_KEY_PIF_FILENAME: An optional list of header info to construct the output file name.

```
HEADER_INFO_TO_KEY_PIF_FILENAME => [ 'START_DATE', 'BSC_ID' ],
```

2.2.2 PIF naming convention

The PIF will be output with a name in the following format where '-#-' is used as a delimiter:

```
<header_info_to_key_pif_filename>-#-<original filename > -#-<value of
block name> <counter> -#-I.pif
```

where <counter> will either be nonexistent (as in the case when the first file of <original filename> is processed) or it will be an integer (1,2,3,..) (as in the case where a file with same name has file already parsed and whose the resulting PIF still exists in the intermediate directory).

2.3 NORBSS_config

This vendor interface parses configuration data files into PIF format. Essentially only configuration data files are parsed, but the process requires information from the configuration data's associated format file.

Not only does the format file contain details of the format of data, it contains a list of column names. The column names uniquely identify a name for each field in the configuration data.

Note: It will be difficult to distinguish this from raw data in an unfactorized format because both are just "lines" of values separated by delimiter "colSep". Therefore it is important that the configuration in INPUT_FILE_DESCRIPTION is used to distinguish configuration data and not process any raw unfactorized format data.

2.3.1 Rule Configuration

To evoke the NORBSS_config module the configuration option 'RULE_TYPE' must be set to 'NORBSS_config'. The configuration options 'RULE_TYPE', 'RULE_DESC', 'INPUT_FILE_DESCRIPTION', 'INPUT_DIR_DEPTH', 'FILENAME_HEADER_FIELDS' are common to all productised parsers and are fully described in Gateway Framework

User Guide; all other configuration options listed are specific to the NORBSS_config module.

- TIME, DATE, OBJECT: These are referred to as the module's 'musthave' options, their configuration is mandatory and must be in the form of a 2 element ARRAY. These options are meant to be used to specify respectively the time, date and configuration object of the data.

  The second component of the array must be either 'LITERAL', 'EXTRACT' or EXTRACT_DEL'. The configuration 'LITERAL' implies that the musthave option will take the literal value specified in the first component of its array. If 'EXTRACT' or 'EXTRACT_DEL' is configured then the first component of the array must be a key in the FILENAME_HEADER_FIELDS or DIRECTORY_HEADER_FIELDS configuration hash, i.e. the value of the 'musthave' parameter will be extracted from the filename or directory name by this key.

  In the above example, the musthave option 'TIME' will be given the literal value '12:00'; whereas 'OBJECT' extracted from the filename by the key Object and there will be no countername Object in the PIF header block; whereas 'DATE' will have the value extracted from the last 8 digits of the filename by the key 'FileDate' and the countername 'FileDate' will also be listed in the PIF header block.

```
      TIME   => ['12:00', 'LITERAL'],
      DATE   => ['Filedate', 'EXTRACT'],
      OBJECT => ['Object', 'EXTRACT_DEL'],
```

- FMT_DEFINED, FMT_READ: At least one of these two configuration options must be specified. If both are specified then FMT_READ is used in preference to FMT_DEFINED. If retrieving the format parameters via FMT_READ method fails, then in the case of both options being specified FMT_DEFINED is used instead.
- FMT_DEFINED: This is a HASH option, it described the specification of the format parameters of the data. It can have the following components.

  Mandatory components:
  - colSep: a delimiter used to distinguish data fields; allowable values \W+
  - lineSep: a delimiter used to distinguish data records; allowable values \W+

  Optional components:
  - fieldLength (optional) description of lengths of columns; allowable values : FIXED, VARIABLE COLUMNS (mandatory -

ARRAY of scalars) a list of column names of the configuration data to be parsed

- FMT_READ: This is a HASH option. When specified it will, upon parsing the file look in the current directory for a format file with a name matching FORMAT_DESCRIPTION, and read the file's contents and interpret them as the format parameters' specification, where '-#-' is used as a delimiter.

  The format file may be a compressed or uncompressed file, and have read and write permissions. Even if the format file is used in the process of parsing a file and the option DEBUG=nodebug, the format file will remain in the input directory, therefore you should arrange for a clean up of unnecessary format files periodically to avoid the input directory become full of irrelevant format files.

  This hash option has 3 components:

  - FORMAT_DESCRIPTION: (mandatory -ARRAY of scalars) format file is obtained by matching file names to the first matching string in this list of possible matches. A dynamic substitution into the matching string, denoted by (#<name>#), can be made for the values of the OBJECT, TIME, DATE that will be placed in PIF header e.g.

    configuration file name : aNETWORK_BTS.1234_20000104
    associated format file name : format_BTS.2000104_1234
    configuration settings:

```
FILENAME_HEADER_FIELDS => {
      'Time'     => '\.(\d{4})_\d{8}$',
      'Filedate' => '(\d{8})$',
      'Obj'      => 'aNETWORK_(\w+)\.'
},
OBJECT => ['Obj', 'EXTRACT_DEL'],
TIME   => ['Time', 'EXTRACT_DEL'],
DATE   => ['Filedate', 'EXTRACT_DEL'],
FMT_READ => {
      FORMAT_DESCRIPTION => [
      'format_(#OBJECT#).(#DATE#)_(#TIME#)'],
      ......
```

    In this case parser will extract following values:

```
(#OBJECT#) = BTS
(#TIME#) = 1234
(#DATE#) = 20000104
```

    And so the search pattern for a format file will be set to:

```
format_BTS.20000104_1234,
```

Hence it will find the associated format file necessary.

- FMT_LEATEXT: (mandatory -SCALAR) first line of format file must match this pattern
- FMT_DIVIDER: (optional -SCALAR) line of format file to match that divides format parameters from column specification.

  Note: if not set assumes value '\[COLUMNS\]'

```
   FMT_READ => {
         FORMAT_DESCRIPTION => [ 'format_(#OBJECT#)'],
         FMT_LEADTEXT       => '\[SYNTAX\]',
         FMT_DIVIDER        => '\[COLUMNS\]'
   },
```

- PIF_NAME_PREFIX_FILE: This is an optional configuration option which is an ARRAY of 2 components. It has a default value 'CONFIG'

  The second component of the array must be either 'LITERAL' or 'EXTRACT'. The configuration 'LITERAL' implies that the literal value specified in the first component of its array. If 'EXTRACT' is configured then the first component of the array must be a regular expression with tagged expressions. The tagged expression will then be used as a prefix to the PIF name delimited by '-#-'.

```
   PIF_NAME_PREFIX_FILE => ['^(\w+)\.(\d{3})\.\w+\.(CNF)', 'EXTRACT']
```

- PIF_NAME_PREFIX_DIR: Optional configuration entry, specifying the REs to extract from the input directory structure to use in the output PIF filename. Each RE to be extracted is bracketed (). The PIF_NAME_PREFIX_DIR is inserted in the output PIF filename before the FILE prefixes, if any. The example below is extracting the last entry in the directory tree

```
   PIF_NAME_PREFIX_DIR => '\/(\w+)$',
```

2.3.2 PIF naming convention

The naming uses the value of the configuration parameters 'PIF_NAME_PREFIX' and 'musthave' parameter 'OBJECT' along with the suffix 'I.pif'. The parameter 'PIF_NAME_PREFIX_FILE' and 'PIF_NAME_PREFIX_DIR' are optional and has a default value of 'CONFIG', however the user may want his own prefix by configuring a regular expression to be extracted from the filename or by stating his own literal value for the prefix. The PIF file name will be in the following format:

```
"<PIF_NAME_PREFIX_DIR>-#-<PIF_NAME_PREFIX_FILE>-#-<OBJECT>-#-I.pif"
```

where '-#-' is used as a delimiter. Examples:
Input file name : sdo1.001.BSC.CNF.20000224_1106

In EngineConfig.pm :

```
      OBJECT => [ 'BSC', 'LITERAL']
      PIF_NAME_PREFIX => ['ConfigurationFiles','LITERAL']
```

PIF file name: ConfigurationFiles-#-BSC-#-I.pif

In EngineConfig.pm :

```
      OBJECT => [ 'BSC', 'LITERAL']
      PIF_NAME_PREFIX => ['^(\w+)\.(\d{3})\.\w+\.(CNF)','EXTRACT']
```

PIF file name: sdo1-#-001-#-CNF-#-BSC-#-I.pif

In EngineConfig.pm :

```
      OBJECT => [ 'BSC', 'LITERAL']
```

PIF file name: CONFIG-#-BSC-#-I.pif

Note: NORBSS_config will overwrite any existing PIF in the intermediate directory that has the same name as the file it is intending to write, this is different to the behaviour demonstrated by NORBSS_data where it will not overwrite existing files of the same name.

Note : If the 'EXTRACT' method is chosen but the regular expression does not match the file name then the PIF name will simply be:

```
      CONFIG-#-<OBJECT>-#-I.pif
```

3. Post Parser Rules and Configuration

This section describes specific configurations needed to for the post parsing rules specific to the Nortel BSS parser.

3.1 ADD_NEWTIME

This is a rule used to add a new counter dependent upon the value of a header counter, (in the reference configuration this counter is TIME.) The value of the new counter placed in the file depends on which one of the range of values in NEW_TIMES the counter value match.

The non-standard configuration options are described below.

Examples of the full user config can be found in the EXAMPLES/pcu_join directory

3.1.1 Rule Configuration

- HEADER_NAME_USED_TO_ID_DATA_RECORD: (Mandatory) A scalar describing the counter name in the header which is being examined and manipulated by this rule.
  ```
  HEADER_NAME_USED_TO_ID_DATA_RECORD => 'TIME',
  ```

- NEW_COUNTER_NAME: (Mandatory) The name of the new counter which is to be inserted when a match is found for the HEADER_NAME_USED_TO_ID_DATA_RECORD counter. In the example below the new counter is called NEWTIME.
  ```
  NEW_COUNTER_NAME => 'NEWTIME',
  ```

- OLD_COUNTER_MAP: (Mandatory) This is a RE to match and extract the values from the old COUNTER For example the value below will extract the hour from the time and store it.
  ```
  OLD_COUNTER_MAP => '(\d{2}):\d{2}',
  ```

- NEW_COUNTER_MAP: (Mandatory) If a match is found using the OLD_COUNTER_MAP the new counter is created using the RE in this configuration. The value below will take the value derived from the previous RE match in OLD_COUNTER_MAP and use it to create the new counter. For example if the HEADER_NAME_USED_TO_ID_DATA_RECORD had a value of 23:45 this will create a new counter of 23:00 etc.
  ```
  NEW_COUNTER_MAP => '$1:00',
  ```

- KEEP_COUNTERS: (Optional) A list of counters which are to be copied from the input PIF to the output PIF.

```
    KEEP_COUNTERS => ['bsc']
```

3.1.2 Sample Usage

Given the following input PIF header:

```
##START|HEADER
RAWorSUM|BSC|FileType|Network|DURATION|DATE|TIME|Object|sdoName
raw|003|OFS|001|3600|31Mar2000|12:15|BTS|sdo1
##END|HEADER
```

The NEWTIME counter is added to the header block.

```
##START|HEADER
RAWorSUM|BSC|FileType|Network|DURATION|DATE|TIME|NEWTIME|Object|sdoName
raw|003|OFS|001|3600|31Mar2000|12:15|12:00|BTS|sdo1
##END|HEADER
```

The configuration is as follow:

```
      {
            RULE_TYPE => 'ADD_NEWTIME',
            INPUT_FILE_DESCRIPTION => [
                  '(.*).PCUL_B(.*)#-BTS(.*)-#-I-#-S-#-S-#-S-#-S.pif',
            ],
            RULE_DESC => 'Insert NEWTIME in each data record',
            OLD_COUNTER_MAP => '(\d{2}):\d{2}',
            NEW_COUNTER_MAP => '$1:00',
            HEADER_NAME_USED_TO_ID_DATA_RECORD => 'TIME',
            NEW_COUNTER_NAME => 'NEWTIME',
            PRODUCE_PIF => "True",
            PRODUCE_LIF => 0,
      },
```

3.2 COUNTER_EXTRACT

This rule performs an extraction on a specific set of counters names. The counter names contain information which is used to create a new key which is written along with the renamed counters to a new PIF.

The non-standard configuration options are listed below:

3.2.1 Rule Configuration

- EXTRACT_COUNTERS_DESCRIPTION: (Mandatory) An array containing a list of REs used to extract the names of the counters, and the keys. The purpose of this entry is:
  - to extract all the key values for the new counter
  - to extract and save the old counter name which will be used
  - to create the old counter values for processing.

NOTE: The counter name and key must be bracketed to return in $1 and $2 in the RE match as below.

```
EXTRACT_COUNTERS_DESCRIPTION => ['^(CUM_1084_00[034]_)(\d+)'],
```

For example for a counter for LAPD 84 such as CUM_1084_003_84, using the rule above $1 will contain CUM_1084_003_ and the key will be 84.

- EXTRACT_COUNTERS_NEW_VALUES: (Mandatory) A RE which is used to extract the the new counter name which is required to be written to the new PIF.

  ```
  EXTRACT_COUNTERS_NEW_VALUE => '^(CUM_1084_00[034])_\d+',
  ```

  In this case for counter CUM_1084_003_84, the new counter name will be CUM_1084_003.

- INVALID_COUNTER_VALUES: (Mandatory) A list of counter values which are invalid. If these are found in any of the input counters, the block will not be written to the PIF

  ```
  INVALID_COUNTER_VALUES => ['-1'],
  ```

- NEW_COUNTER_NAME: (Mandatory) The name of the new counter which is being created from the extraction of the keys from the counter names.

  ```
  NEW_COUNTER_NAME => 'LAPD',
  ```

- KEEP_COUNTERS: (Optional) A list of counters from the input PIF which are to be written to the output PIF/LIF

  ```
  KEEP_COUNTERS => ["bscName","BSC"],
  ```

3.2.2 Sample Usage

Taking the example of the input PIF with the following counter names and rows

```
CUM_1084_000_10|CUM_1084_003_10|CUM_1084_004_10|CUM_1084_000_99|CUM_1084
_003_99|CUM_1084_004_99
11|12|13|24|25|26
```

This would produce a PIF with 2 lines

```
LAPD|CUM_1084_000|CUM_1084_003|CUM_1084_004
10|11|12|13
99|24|25|26
```

In this case the new counter created is LAPD and the key values for the counter are extracted from the counter names. The counter names are also renamed to remove the key from their name.

The configuration is as follow:

```
{
      RULE_TYPE => 'COUNTER_EXTRACT',
      RULE_DESC => "Extract LAPD data from BSC file",
      INPUT_FILE_DESCRIPTION => [
            '(.*)\.BSC(.*)-#-BSC(.*)-#-I.pif'
      ],
      EXTRACT_COUNTERS_DESCRIPTION => [
            '^(CUM_1084_00[034]_)(\d{2})'
      ],
      EXTRACT_COUNTERS_NEW_VALUE => '^(CUM_1084_00[034])_\d+',
      INVALID_COUNTER_VALUES => ['-1'],
      NEW_COUNTER_NAME => 'LAPD',
      OUTPUT_BLOCK_NAME => 'LAPD',
      OUTPUT_FILENAME_START => 'LAPD',
      PRODUCE_PIF => 0,
      OUTPUT_FORMAT => "LIF_Writer",
      KEEP_COUNTERS => ["bscName","BSC"],
},
```

3.3 INFOINSERT_TCHANNEL

This rule is an extension of the INFOINSERT rule. The counter value which is inserted is generated by incrementing its value by a configurable amount when a certain value is seen in the info PIF. This will be further explained in the configuration below.

3.3.1 Rule Configuration

- INFO_FILE_DESCRIPTION: (Mandatory) A RE of the files to use for the extraction of the counter information to insert.
  ```
  INFO_FILE_DESCRIPTION => 'current(.*)-#-CHANNEL-#-I.pif',
  ```

- INFO_FILE_KEYS: (Mandatory) The counter names used as keys when storing the counter values to be inserted in the output PIF. This key must be matched by the INPUT_FILE header and record keys for the output counter to be written to the PIF.
  ```
  INFO_FILE_KEYS => ['bscId', 'btssm_no', 'btsId' ],
  ```

- INPUT_FILE_HEADER_KEYS: (Mandatory) The counters to be used from the header block of the file to create the PIF key used to match the INFO_FILE_KEYS above
  ```
  INPUT_FILE_HEADER_KEYS => [],
  ```

- INPUT_FILE_RECORD_KEYS: (Mandatory) The counters to be used from the input file to match the records against those read from the info file.
  ```
  INPUT_FILE_RECORD_KEYS => ['bsc', 'btsSiteManager', 'bts'],
  ```

Therefore in this example the key bscId-btssm_no-btsId from the INFO_FILE must match the INPUT_FILE key bsc-btsSiteManager-bts for the counter to be inserted in the output.

- WRITE_DATA_LINE: (Mandatory) If set to true then the data line will be written to the output PIF even in the case where a key match is not made between the two files

```
WRITE_DATA_LINE => 'True',
```

- MANIP_COUNTERS: (Mandatory) A list of the counters to manipulate, this is configured as an array of anonymous hashes. The required configuration in each hash element are:
  - COUNTER_NAME: (Mandatory) The name of the counter to be manipulated in INFO file

```
      COUNTER_NAME => 'channelType',
```

  - COUNTER_VALUES: (Mandatory) The counter values to match, which require further processing to create the output counter. This is an array of values.
  - COUNTER_INCREMENTS: (Mandatory) The amount to increment the output counter when the value in the map above is encountered. In the example below, when channelType has value 3, the output counter has value 8 etc.
  - NEW_COUNTER_NAME: (Mandatory) The name of the mapped counter in the output PIF

```
      'MANIP_COUNTERS' => [
            {
                  COUNTER_NAME => 'channelType',
                  COUNTER_VALUES => ['3','5','7','8'],
                  COUNTER_INCREMENTS => ['8','4','0','7'],
                  NEW_COUNTER_NAME => 'deftch',
            },
      ],
```

- REDUNDANT_HEADER_COUNTERS: (Optional) A list of header counters which are to be removed from the input PIF when creating the output PIF

```
REDUNDANT_HEADER_COUNTERS => ['DURATION'],
```

- REDUNDANT_COUNTERS: (Optional) A list of block counters which are to be removed from the input PIF when creating the output PIF

```
REDUNDANT_COUNTERS => [],
```

- OUTPUT_FILENAME_START: (Optional) If set, the output PIF filename will be prefixed by this string

```
    OUTPUT_FILENAME_START => 'TCH',
```

3.3.2 Sample Usage

This extract of the INFO PIF

```
##END|HEADER
##START|CNF
bscId|btssm_no|btsId|channelType
0|8|0|3
0|8|0|8
1|2|1|5
1|2|1|5
```

Given that the 1st 3 columns are the key the output counter value for deftch would be:

0-8-0 would map to deftch 15, (channelType 3, increment by 8, channelType 8 increment by 7)

1-2-1 would map to deftch 8, (channelType 5 increment by 4)

3.4 ADD_RECORDS

This is a rule used to add a new counter dependent upon the value of a header counter. The value of the new counter placed in the file depends on which one of the range of values in ADD_TIME hash the counter value match.

The non-standard configuration options are described below.

3.4.1 Rule Configuration

- HEADER_NAME_USED_TO_ID_DATA_RECORD: (Mandatory) A scalar describing the counter name in the header which is being examined and manipulated by this rule.

```
    HEADER_NAME_USED_TO_ID_DATA_RECORD => 'TIME',
```

- NEW_REC_ID_NAME: (Mandatory) The name of the new counter which is to be inserted when a match is found for the HEADER_NAME_USED_TO_ID_DATA_RECORD counter. In the example below the new counter is called NEWTIME.

```
    NEW_REC_ID_NAME => 'START_TIME',
```

- ADD_TIMES: (Mandatory) A hash listing the counter mapping to be used.

```
    ADD_TIMES => {
          '00:00' => ['00:00', '00:15', '00:30', '00:45'],
          '01:00' => ['01:00', '01:15', '01:30', '01:45'],
          '02:00' => ['02:00', '02:15', '02:30', '02:45'],
          ...
          '22:00' => ['22:00', '22:15', '22:30', '22:45'],
          '23:00' => ['23:00', '23:15', '23:30', '23:45'],
      }
```

3.4.2 Sample Usage

Sample extracts of PIFs

```
125.02072007.1815-#-I.pif
## Parser Intermediate File
##START|HEADER
START_DATE|TIME|FileID
14Jun2007|18:15|125
##END|HEADER
##START|125
BTSS|COUNTER_1
125|1
##END|125
```

The new intermediate PIF file produced by the ADD_RECORD rule:

```
125.02072007.1815-#-I-#-AR.pif
## Parser Intermediate File
##START|HEADER
START_DATE|FileID|TIME|START_TIME
14Jun2007|125|18:15|18:00
##END|HEADER
##START|BTS
BTS|COUNTER_1|TIME|START_TIME
125|1|18:15|18:00
125|2|18:15|18:00
125|3|18:15|18:00
125|4|18:15|18:00
##END|BTS
```

In this case, the START_TIME and TIME values are added to the data record. A matching START_TIME value is also added to the header. The configuration is as follow:

```
      {
          RULE_TYPE => 'ADD_RECORDS',
          RULE_DESC => 'Insert START_TIME in each data record',
          INPUT_FILE_DESCRIPTION => ['^(\d+\.\d{8}\.\d{2})\d{2}-#-
      I\.pif'],
          ADD_TIMES => {
              '18:00' => ['18:00', '18:15', '18:30', '18:45'],
          },
          HEADER_NAMES_USED_TO_ID_DATA_RECORD => 'TIME',
          NEW_REC_ID_NAME => 'START_TIME',
          PRODUCE_PIF => 'True',
          PRODUCE_LIF => 0,
      },
```

3.5 JOIN_15

This rule is an extension of the JOIN rule. The rule allows multiple 15 minute duration files to be joined into a 60 minute duration file.

The non-standard configuration options are listed below:

3.5.1 Rule Configuration

The mandatory entries are:

- COUNTERS_TO_JOIN_ON: The list of counters to join the input files on.

```
COUNTERS_TO_JOIN_ON => [qw(BTS BSM BSC)],
```

- OUTPUT_BLOCK_NAME: The name of the new data block in the output file.

The optional configuration entries are:

- REDUNDANT_HEADER_COUNTERS: A list of counters to remove from the header data block.
- REDUNDANT_DATA_COUNTERS: A list of counters to remove from the data block.
- OUTPUT_FILENAME_START: A prefix to use on the output file name.
- HEADER_COUNTERS_TO_USE_IN_OUTPUT_FILENAME: A list of counter names to use to construct the output file name.
- HOURS_TO_WAIT_FOR_PARTNER_FILES: This is the amount of time for a PIF file to wait for its partner file before it can be joined. The entry should be removed from the configuration or set to -1 if there is no partner file to wait for.
- TIME_JOINFILE_PRODUCED: If set to true, the rule will add time/date to the output joined file. The time/date is in the format

```
<DAY><DATE><MON><YEAR>_HH:MM:SS
```

- QUOTE_INPUT_PIFS: If set the rule will add the names of the input PIFs to the output LIF. Can be useful when trying to debug the joining of a large number of files or a complex rule.

3.5.2 Sample Usage

Sample extracts of PIFs

```
125.02072007.1800-#-I.pif
## Parser Intermediate File
##START|HEADER
START_DATE|TIME|FileID|START_TIME
```

```
14Jun2007|18:00|125|18:00
##END|HEADER
##START|125
BTS|START_TIME|TIME|COUNTER_1
125|18:00|18:00|1
##END|125

125.02072007.1815-#-I.pif
## Parser Intermediate File
##START|HEADER
START_DATE|TIME|FileID|START_TIME
14Jun2007|18:15|125|18:00
##END|HEADER
##START|125
BTS|START_TIME|TIME|COUNTER_1
125|18:00|18:15|2
##END|125

125.02072007.1830-#-I.pif
## Parser Intermediate File
##START|HEADER
START_DATE|TIME|FileID|START_TIME
14Jun2007|18:30|125|18:00
##END|HEADER
##START|125
BTS|START_TIME|TIME|COUNTER_1
125|18:00|18:30|3
##END|125

125.02072007.1845-#-I.pif
## Parser Intermediate File
##START|HEADER
START_DATE|TIME|FileID|START_TIME
14Jun2007|18:45|125|18:00
##END|HEADER
##START|125
BTS|START_TIME|TIME|COUNTER_1
125|18:00|18:45|4
##END|125
```

The new intermediate PIF file produced by the JOIN_15 rule:

```
BTS-#-125-#-02072007-#-1845-#-J.pif
## Parser Intermediate File
##START|HEADER
START_DATE|FileID|START_TIME
14Jun2007|125|18:00
##END|HEADER
##START|BTS
BTS|COUNTER_1
125|1
125|2
125|3
125|4
##END|BTS
```

In this case, the four 15 minute interval files are accumulated into one 1 hour interval file. The START_TIME and TIME counters are stripped from each data record. Similarly, the TIME counter is also stripped from the header.

The configuration is as follow:

```
{
    RULE_TYPE => 'JOIN_15',
    RULE_DESC => 'Join 15 minute durations into 1 hour',
    INPUT_FILE_DESCRIPTION => ['^(\d+\.\d{8}\.\d{2})\d{2}-#-
I\.pif'],
    PRODUCE_PIF => 'True',
    PRODUCE_LIF => 0,
    OUTPUT_BLOCK_NAME => 'BTS',
    COUNTERS_TO_JOIN_ON => ['BTS','START_TIME','TIME'],
    OUTPUT_FILENAME_START => 'BTS',
    HEADER_COUNTERS_TO_USE_IN_OUTPUT_FILENAME =>
['FileID','START_DATE','START_TIME'],
    REDUNDANT_HEADER_COUNTERS => ['TIME'],
    REDUNDANT_DATA_COUNTERS => ['START_TIME','TIME'],
    HOURS_TO_WAIT_FOR_PARTNER_FILES => -1,
},
```

3.6 AGGREGATE

This rule is used to aggregate data after applying a JOIN_15 rule.

The non-standard configuration options are listed below:

3.6.1 Rule Configuration

The mandatory configuration entries are:

- OUTPUT_BLOCK_NAME: the block name to use in the output PIF/LIF.
- COUNTERS_TO_SORT_ON: the set of counters to use as the key for accumulation.

```
COUNTERS_TO_SORT_ON => [qw(CELL)],
```

The optional configuration entries are:

- NON_ADDITIVE_COUNTERS: Counters that are not to be accumulated, but passed through in the output as they are found. For example date and time fields.
- REDUNDANT_DATA_COUNTERS: Data counters to remove from the output blocks.
- MAXIMUM_COUNTERS: the list of counters to derive the maximum values for.

```
MAXIMUM_COUNTERS => [qw(LOAD)],
```

- MINUMUM_COUNTERS – the list of counters to derive the minimum values for.

```
MINIMUM_COUNTERS => [qw(LOAD)],
```

3.6.2 Sample Usage

Given the following block of PIF data:

```
##START|DATA_BLOCK
CELL|COUNT1|COUNT2|COUNT3|LOAD|TRX
DF0001|1|2|3|100|1
DF0001|1|2|3|10|2
DF0001|1|2|3|50|3
DF0001|1|2|3|90|4
DF0002|1|2|3|60|1
DF0002|1|2|3|0|2
DF0002|1|2|3|30|3
DF0002|1|2|3|20|4
##END|DATA_BLOCK
```

The output after processing by AGGREGATE:

```
##START|AGR
CELL|COUNT1|COUNT2|AGR_NUM_IN_SUM|COUNT3|LOAD|TRX
DF0001|4|8|4|12|100|10
DF0002|4|8|4|12|60|10
##END|AGR
```

A sample configuration used in the sample application of the rule:

```
{
     RULE_TYPE => 'AGGREGATE',
     RULE_DESC => 'Aggregate Sample',
     INPUT_FILE_DESCRIPTION => ['.*'],
     COUNTERS_TO_SORT_ON => [qw(CELL)],
     OUTPUT_BLOCK_NAME => 'AGR',
     REDUNDANT_DATA_COUNTERS => [],
     PRODUCE_PIF => 'True',
     PRODUCE_LIF => 0,
     NON_ADDITIVE_COUNTERS => [],
     MAXIMUM_COUNTERS => [qw(LOAD)],
     MINIMUM_COUNTERS => [qw(LOAD)],
}
```

4. Configuration Support

Configurations listed below are available for the systems releases from Nortel BSS.

4.1 Sample Configuration

There are sample configurations used for Nortel BSS Gateway within the examples directory for the following system releases:

- GSM/BSS V13
- GSM/BSS V14.3

4.2 Tech Pack Support

Tech pack support is included in the Nortel BSS Gateway for the following data version.

- GSM/BSS V15
- GSM/BSS V16

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

5. Installation Specific Information

5.1 V16 Tech Pack Directory Structure for ONL

Raw performance files must be placed within a subdirectory inside the $IN_DIR named ‘raw’.

```
        $IN_DIR/raw/
```

Summary performance files must be placed within a subdirectory inside $IN_DIR named ‘sum’.

```
       $IN_DIR/sum/
```

Format files must be placed in the same subdirectory as the raw or configuration files that they describe. For example, if raw files are placed in a subdirectory called $IN_DIR/raw/, the format files describing the raw files must be placed in the same subdirectory. Similarly, if configuration files are placed in $IN_DIR, the corresponding format files must also be placed in $IN_DIR.

TransferConfig.pm must be properly configured to comply with the requirements above.