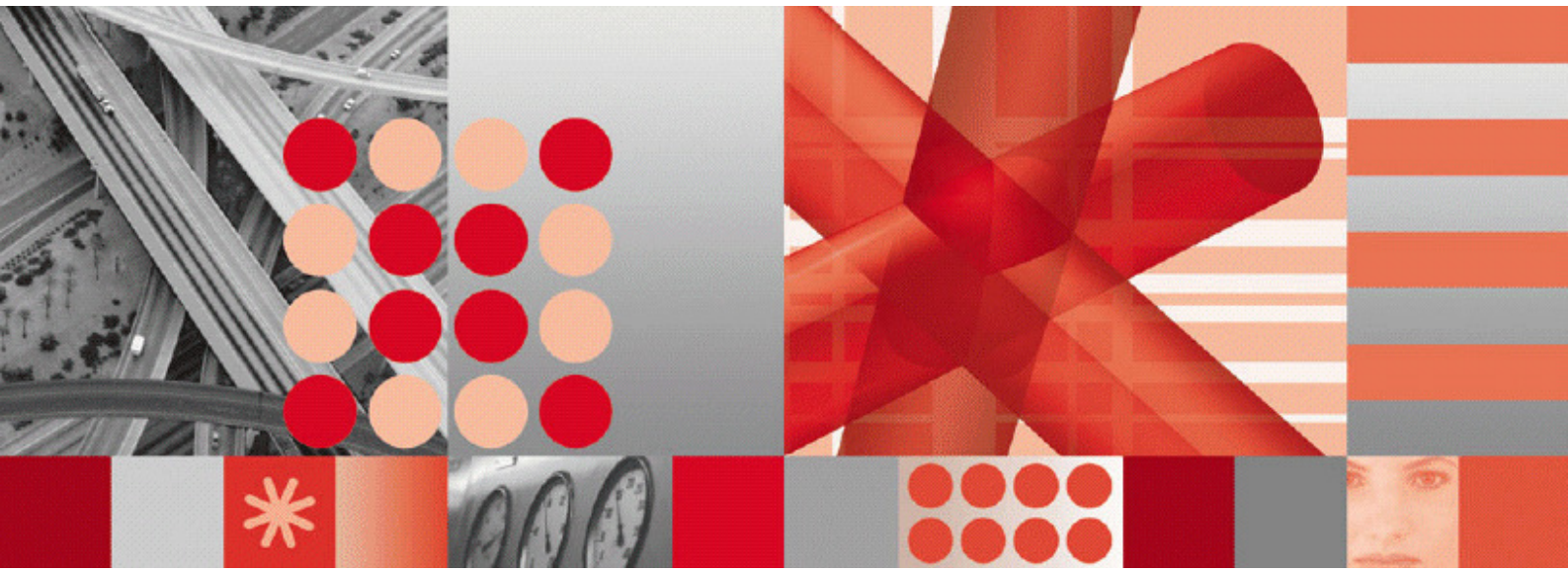
Tivoli Metrica Performance Manager Gateways - Perl

Version 2.2.13



Gateway Framework User Guide

Note: Before using this information and the product it supports, read the information in Notices on page 79.

This edition applies to Version 4.1 of IBM® Tivoli® Metrica Performance Manager and to all subsequent releases and modifications until otherwise indicated in new editions.

Table of Contents

Glossary

PM	Performance Management. The term used to describe the NPR and NGN suite of products for network performance measurement and monitoring.
raw performance data	Vendor performance/hierarchy data in the format in its exported format. The data is not in any standard form that can be loaded directly by the PM systems.
PIF	Parser Intermediate Format. Internal temporary format for the performance data, used internally by the Gateway to transfer data between different stages and rules
RE	Regular Expression. A pattern of characters and symbols describing the structure of a sequence of strings. Used extensively in the Gateway configuration to match input files, counter names and values. For example: \d{2}\w{3}\d{4} would match a date string such as 20Mar2004
LIF	Loader Input Format. The standard form for performance data for loading into the PM system. The loader process interprets this data and loads it into the appropriate tables/columns in the database.
ftp	File Transfer Protocol. Used for the transfer to/from remote servers and management of files on remote servers.
ssh	Secure SHell. Remote shell protocol using encryption to provide a secure link between the local and remote host.
scp	Secure Copy. Part of ssh, secure copy of files to/from remote servers.
rcp	Remote Copy. Non secure equivalent of scp
loadmap	A loadmap is a description of the mapping of counter data in LIF blocks to database tables and columns in the performance management system. It also supports conditional statements and simple data transformation requests.
OMC	Operations and Maintenance Centre. A management system used to configure, monitor and manage physical and logically disparate network elements within an operator's network.

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 About this Documentation

1.1 Audience

The target audience of this document is Vallent Performance Manager for Wireless customers. They should be familiar with telecommunication and IT principles and should also have a good understanding of Solaris.

IMPORTANT: Before attempting an installation of Performance Manager for Wireless you are strongly advised to read the release notes and any readme files distributed with your Performance Manager for Wireless software. Readme files and release notes may contain information specific to your installation not contained in this guide. Failure to consult readme files and release notes may result in a corrupt, incomplete or failed installation.

Note: Performance Manager for Wireless Administrators should not, without prior consultation and agreement from IBM, make any changes to the Index Organized tables or database schema. Changes to the Index Organized tables or database schema may result in corruption of data and failure of the Performance Manager for Wireless System. This applies to all releases of Performance Manager for Wireless using all versions of interfaces.

1.2 Required Skills and Knowledge

This guide assumes you are familiar with the following:

- General IT Principles
- Sun Solaris Operating System
- Oracle Database
- Windows operating systems
- Graphical User Interfaces
- Network Operator's OSS and BSS systems architecture

This guide also assumes that you are familiar with your company’s network and with procedures for configuring, monitoring, and solving problems on your network.

2 Overview and Architecture

2.1 Overview

The purpose of the Gateway Framework is to provide a standard framework for the transfer, parsing, manipulation and presentation of performance data from remote network elements to the Vallent service assurance applications.

The raw performance data is typically transferred from the operators OMC. This data is exported at fixed periods, for example 15 minutes. The format of this data is typically bulky and in a non-standard format.

The Framework consists of a set of processing stages that perform different functions to output the required performance data. It includes standard tools for logging and recovery, as well as configurable rules for the transformation of data.

The final output is then loaded to the service assurance application to provide end user reporting and monitoring services.

This document details the configuration and usage of version 3.1 of the Gateway Framework.

2.2 Architecture

This section gives a brief overview of the Gateway architecture and the main components. The Gateway consists of 3 main components, Gateway Framework, Vendor Gateways, and the Gateway Configurations.

2.2.1 Gateway Framework

The Gateway Framework provides standardized modules for the collection, parsing and presentation of vendor specific data formats. These modules interact mainly with various directories for reading, writing and storing of files. The Transfer module is split into a separate IN and OUT phase, run before and after parsing of the data respectively. Figure 2-1 gives an overview of the Gateway architecture.

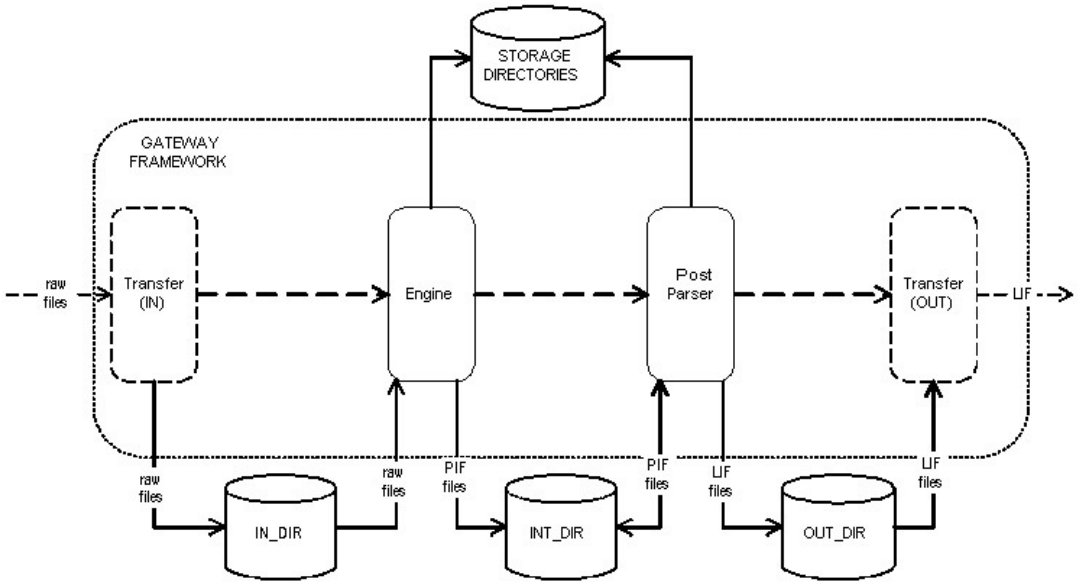


Figure 2-1: Gateway Framework architecture

The modules and stages within the parsing process are:

Transfer (IN): This optional stage allows the configuration of the transfer in of raw performance data from remote server(s), typically the OMC. It supports the scp, ftp, rcp and local cp file transfer mechanisms. Multiple instances can be configured, so files can be transferred from multiple destinations.

Engine: The Engine stage parses the raw performance data files, which is either a vendor or 3rd party standard format, producing the data in PIF format. This processing is performed by a vendor specific Engine rule, which allows the Gateway Framework to be reused for multiple vendors, using different engine rules to match the vendor data. The Engine stage writes out the PIF data to an intermediate directory, or to a common memory store.

For example, to parse Nokia ASCII format data, the NOKIA_ASCII engine rule is used, which produces PIF files.

Post Parser: The Post Parser stage processes this data through multiple standard and vendor specific post parser rules, to transform data for efficient loading in accordance with the PM systems loadmap. Examples of common functions performed within the Post Parser stage are:

- Joining of PIF files.
- Insertion of hierarchy data from a secondary data source e.g. insertion of GSM hierarchy data into performance data based on BTS and CELL keys.
- Accumulation of counter values across rows for example, accumulation of cell counters across a BSC.

The Post Parser rules can produce 2 types of output:

PIF files, which are used as input to the next Post Parser rule on the processing chain.

Output files, which are output to a separate directory. These are the final performance files for loading into the PM system. These can be one of the following:

- LIF files.
- CSV files
- XML files
- Prospect specific format

A vendor Gateway is usually released with a Post Parser configuration that meets the standard requirements for mapping of data to the PM system. Ownership of the complete customer configuration and solution, which usually consists of further data manipulation rules, is the responsibility of PS.

Transfer (OUT): Once the output files have been produced, if required they can be transferred to a remote server for loading. The Transfer engine once again handles this data transfer. It is configured for the transfer of files off the local server onto a remote server. The configuration of this stage is optional also, as this stage may be unnecessary or handled by other external tools.

Other general features contained in the Gateway Framework, which can be reused by Vendor Gateway releases are listed below:

Logging and Auditing: The Gateway Framework provides standard audit and logging tools for Vendor Gateways.

Crash Recovery: The Gateway Framework manages the recovery from a crash due to invalid input file, so the Gateway will not continually fail on a restart.

Backlog processing: The Gateway Framework can drip feed files in the oldest first order through the Gateway, allowing controlled recovery from a backlog situation.

Parallel Processing: Engine and Post Parser rules can be configured to process groups of files in parallel.

Memory Caching of Intermediate data: To further improve performance, intermediate data being passed through various stages of transformation may be cached in memory, rather than on disk.

Note that this now only applies to the post parsing rules. The engine portion of the Gateway will always generate PIF files into the intermediate directory. This facilitates multiple processes being applied to the engine stage, while retaining the benefits of PIF caching at the post parser stage.

Saving of parsed, intermediate and load data: Files from various stages of processing can be configured to be saved to directories for tracability.

Compression of input and stored data: Compression input data can be handled automatically, and stored data files can also be compressed to reduce storage requirements.

Automated Data Transfer: The Framework contains support for the transfer of data both to and from the local server using a number of standard transfer protocols.

Notification: The Gateway Framework provides a notification module for dispatching alarm and statistical event activity messages to the service assurance application notification system.

Statistics: Gathers File, Block and Counter statistics on the data processed by the vendor Gateway.

2.2.2 Vendor Gateways

The Vendor Gateway consists of a number of Engine and Post Parser modules to perform the manipulation of the vendor specific data. It is integrated into the Framework via the Gateway Configuration files.

2.2.3 Gateway Configurations

The Gateway Configuration defines the rules for parsing various data from specific vendor sub-systems. The directory main configuration files are the EngineConfig.pm and UserConfig.pm.

The use of these configuration files will be further described in the configuration and management section below.

3 Gateway Installation

This chapter describes the steps involved in the download and installation of a vendor gateway.

Before installing the Gateway Configurations, ensure that the required Perl version, Gateway Framework and Vendor Gateways are installed.

3.1 Installing Perl

The Gateway Framework release 3.4 requires Perl version 5.6.1 installed. The supported architecture of Perl is listed below:

Platform	Architecture
HP-UX	PA-RISC2.0
Sun Solaris	sun4-solaris
Tru64	alpha-dec_osf
RedHat Linux	i686-linux

Refer to the specific platform Perl Build Instruction to install Perl.

3.2 Installing Gateway

Before installing and configuring a Gateway Configuration, the required Vendor Gateways and Gateway Framework must be downloaded and installed.

Refer to the Gateways Installation Note on how to install the Gateway Framework and Vendor Gateways.

4 Gateway Configuration and Management

This chapter describes the configuration of the Gateway and its various stages, and the system management via the supplied scripts.

4.1 Setting Up Gateway

General setup of the Gateway is performed in the properties file. The properties file sets the general and system level configuration, such as the directories for the raw, intermediate and output data. The property entries can include environment variables in the form of ${var}

4.1.1 properties

The properties file configuration entries are in the following format:

<name>=<value>

where <name> is the name of the property, and <value> the value assigned to it. The property file entries are:

- LOG_FILE: The name and location to write the log file to. By default set to

  LOG_FILE=log.<vendor-subsys>_<version>

- LOG_LEVEL: The log level of the Gateway. Set to 5, the highest level by default. This is useful during installation and configuration of the Gateway, and the level should be reduced to 3 for the final commissioning.

  LOG_LEVEL=0

- AUDIT_FILE: The name of the file to write the audit trail messages to. By default set to:

  AUDIT_FILE=audit

- DISK_FREE: The Gateway can be configured to check for a minimum disk free space on all used directories on startup. If this space is not available the Gateway will log a warning with the required and available free space and exit. The free space required can be specified as a percentage:

  DISK_FREE=10%

  or number of MB free.

  DISK_FREE=300MB

- This setting can be used to prevent the Gateway failing during processing as it exhausts disk space while writing intermediate or output files.

  Set the value to 0 to ignore.

- IN_DIR: The root directory for the raw performance data files. The Vendor Engine rules will read this directory for files to process. Subdirectories can be created to separate different types of files, if appropriate.
- INT_DIR: The intermediate directory for PIF files. If PIF data is being cached to memory then only PIF files being kept between Gateway iterations will be visible in this directory (non debug mode).
- OUT_DIR: The output directory, for the final performance output files created by the Gateway.
- INPUT_STORAGE_DIR: The directory to store the raw files after processing. This is optional, set to 0 if not required.
- INTERMEDIATE_STORAGE_DIR: The directory to store the PIF files created during the Gateway execution. This is optional, set to 0 if not required. Even if in memory PIF caching is being used, they will be output as files to this directory, if it is configured.
- OUTPUT_STORAGE_DIR: The directory to store the performance LIF files.
- DEBUG: Switch on/off debugging. With debugging on, the raw and PIF files will not be removed from their directories, to allow the execution of the Gateway to be traced.

  DEBUG=debug

  switches on debugging

  DEBUG=0

  switches off debugging.

- COMPRESS: Compress data in the storage directories. The default is 0, no compression. The entry:

  COMPRESS=0

  turns off compression and:

  COMPRESS=true

  turns on compression

- COMPRESS_TOOL: The compression utility to use for compressing files. This should be set to the path where compress or gzip is installed.
- REMOTE_COMPRESS_TOOL: This entry is similar to the COMPRESS_TOOL, but it is meant for the remote server. The compression utility specified here must match with the one

specified in COMPRESS_TOOL. This will enable **cpio** channel to compress the data before transferring. Refer section 3.2 for bulk transfer information.

- PIF_MODULE: The PIF storage option to use for the storage of PIFs during processing. There are the following 2 options:

  PIF_MODULE=PIF_Handler

  This is the standard mode, where PIF data is written and read from files in the intermediate directory.

  PIF_MODULE=PIF_Cache

  The PIF data is written and read to local memory rather than disk. PIF files will still be written out to disk if required between Gateway iterations, and to the INTERMEDIATE_STORAGE_DIR if configured. In memory PIF caching has performance advantages over disk based PIF storage. Note that PIF caching no longer applies to the engine stage of processing and only starts at the Post Parser stage.

- MAX_PIF_LOCAL_MEMORY: The maximum local memory that should be used by the PIF_Cache. If it exceeds this value, then a warning will be printed in the log file. In general the memory based PIFs require about 2.3 times the amount of space of the disk based PIFs.

Initially in this file, all that must be set to get the Gateway up and running is the data directories. Other values can be configured as required. The properties value settings can also include environment variables in the form ${var}.

4.1.2 Launching the Gateway

Typically the Gateway is run at specified intervals, for example every 15 minutes. This is controlled using the standard unix tool, cron.

The Gateway has functionality to ensure that a second Gateway will not be launched to process the same set of raw performance files. This can occur in a backlog situation, where the amount of data to process takes longer than can be handled in the 15 minute period.

4.2 Transfer Configuration

The Transfer stage controls the transfer of files both onto and off the local server. Raw files are transferred into the local input directory for processing, and performance LIF files may be transferred from the local server to a remote destination for loading.

This stage is optional, and may not be necessary or may be performed by external tools. In this case the default TransferConfig.pm, which is empty, can be used.

Multiple transfer rules may be configured within this file. Raw files can be copied from multiple remote destinations, and similarly the performance data can be transferred to multiple remote destinations.

The mandatory configuration entries are:

- RULE_DESC: A description that will be logged to the audit file. This is useful for tracking the rule execution.
- DIRECTION: Set to IN or OUT. Specifies whether files are being transferred to/from a remote destination. Raw files for processing would be specified with

  ```
  DIRECTION => IN
  ```

  and performance files for transfer to a remote server

  ```
  DIRECTION => OUT
  ```

- PROTOCOL: The Protocol to use for transfer. This may be set to ftp, scp or rcp. If performing a transfer to/from the local server, set the protocol to rcp, and the TransferEngine will automatically detect that it is a local directory, and use the appropriate local commands.
- HOST: The hostname or IP address of the remote destination
- LOCAL_DIR: The local directory to copy files to when DIRECTION is IN, OR the local directory to copy files from when DIRECTION is OUT
- REMOTE_DIR: The remote directory to copy files from when DIRECTION is OUT, OR the remote directory to copy to when DIRECTION is IN.
- TIMESTAMP_FILE: The file to use to save the timestamp of last file copied in or out.

  NOTE: this file must be unique for each rule entry. See further notes on the timestamp file.
- DEPTH: The directory depth to follow down when copying files from a remote server. If DEPTH is set to 0, then only files in the base remote directory will be copied, if DEPTH is set to 1 then sub directories one down also be copied. Note this directory structure will be replicated in the local directory.
- INPUT_FILE_DESCRIPTION: A hash containing the regular expressions used to match different file types for copying. Only files matching the entries in this hash will be copied.

The optional configuration entries are:

- STATISTICS_FILENAME: The name of the file to output the statistics to. This file will be suffixed with the date and time the Transfer stage ran and will contain statistics on the total time, number of files copied, size of files, for each entry in INPUT_FILE_DESCRIPTION. Overall statistics also for the all files transferred are also outputted.
- OUTPUT_FORMAT: The format to use to write out the Transfer Statistics. Can be any of the supported output formats LIF_Writer, CSV_Writer, XML_Writer or Prospect_Writer.
- PRODUCE_PIF: If set to true, statistics will be output in PIF format. They may be then joined to File Statistics if required. ENABLE_PING: Whether or not to attempt to ping the remote host before initiating the transfer session.

- PING_PROTOCOL: The protocol to use for the ping request. This can be set to icmp, udp, or tcp depending on the ping type required. Note for icmp pings the Gateway must be running as root user.
- PING_RETRY_ATTEMPTS: The number of times to reattempt the ping if it fails. Default is 0.
- RETRY_INTERVAL: The number of seconds to wait between ping attempts. Default is 5 seconds.
- ENABLE_LOCAL_COMPRESSION: Whether or not to compress the local files copied in or out. Files transferred in will be compressed once they have been transferred. Files transferred out will be compressed before transfer to reduce the network load.
- NUMBER_OF_FILES_TO_PROCESS: If required, the files can be drip fed into the Gateway. If set to a value, the TransferEngine will not copy in more than this number of files on each iteration. Only applies to files being copied onto the local server.
- DELETE_ORIGINAL: Whether or not to delete the original file on the remote server, in the case of DIRECTION in, or on the local server, in the case of DIRECTION out. Defaults to 0, do not delete.
- TMP_FILENAME_EXTENSION: The temporary extension to use on files while they are being copied. This prevents incomplete files being read by a process during transfer. Defaults to ".pt".
- TIMEOUT: Numerical value in seconds for file transfer timeout. Defaults to 20 seconds if not set.
- OVERWRITE_FILES: Whether or not to overwrite the existing file on the remote server, in the case of DIRECTION out, or on the local server, in the case of DIRECTION in. Defaults to 0, do not overwrite. Set to ‘True’ to overwrite.

For the protocols supported there are number of protocol specific entries.

For the ftp protocol:

- USER: the username to use to login to the remote server.
- PASS: the password to use on login to the remote server.
- TRANSFER_MODE: The transfer mode can be either be configured to ‘ASCII’ or ‘BINARY’ only. The transfer mode is set to binary by default.
- LIST_SLEEP: the number of seconds to sleep after getting the list of files to be transferred. This is to allow any files that are still in the process of populating data to finish before they are copied over.

For the scp/rcp protocol:

- PROTOCOL_PATH: The location of the protocol installation. This should be the path to where the ssh/rsh installation is, usually **`/usr/local/bin`**. This is used to execute the ssh/scp and rsh/rcp commands used during the transfer process.
- BULK_TRANSFER: When this entry is set to 'TRUE' or '1', the **`cpio`** command will be used in transferring files. The **`cpio`** command will create a private channel between local host and remote host before transferring files, thus increasing the performance when there is a big quantity of files to transfer. To further increase the performance, a new entry in "properties" file for remote server compression utility is required. Refer section 3.1.2 for details. All destination files will be overwritten and drip feed mechanism will be ignored for bulk transfer. Do not set BULK_TRANSFER, or set to '0' to disable bulk transfer.

  NOTE: The total size of files to be transferred is limited to 2GB for bulk transfer. This is a limitation of cpio tool imposed by XPG/4 and POSIX 2 standard.
- ENABLE_REMOTE_COMPRESSION: Whether or not to compress the remote files copied in or out before the are transferred.

  NOTE: ENABLE_LOCAL_COMPRESSION and ENABLE_REMOTE_COMPRESSION will work exclusively, where ENABLE_REMOTE_COMPRESSION will take precedence when DIRECTION 'IN', and ENABLE_LOCAL_COMPRESSION will take precedence when DIRECTION 'OUT'.
- RETRY_ATTEMPTS: The number of times to reattempt the transfer if it fails. Default is 0.

For ssh installations, the ssh key exchange must be validated to ensure that the transfer process will run correctly without prompting for a user or password. This is checked by running:

```
> ssh <remote server>
```

This should login into the remote server without prompting for a user name/password, if the ssh keys have been properly configured. Below is the sample configuration of an input an output transfer rule from TransferConfig.pm:

```
# Incoming rule
{
  RULE_DESC => 'Copy raw files in for processing',
  PROTOCOL => 'ftp',
  DIRECTION => 'IN',
  HOST => 'nok_server',
  PING_PROTOCOL => 'udp', # only for unreliable connections
  PING_RETRY_ATTEMPTS => 2,
```

```
  RETRY_INTERVAL => 2,
  USER => 'virtuo',
  PASS => 'virtuo',
  DEPTH => 0,
  LOCAL_DIR => "../data_dirs/input_d",
  REMOTE_DIR => "/spool/nokia/dumped_files",
  TIMESTAMP_FILE => "./lock_dir/.itimestamp",
  INPUT_FILE_DESCRIPTION => {
    BSC => 'bsc_.*\.raw',
    BTS => 'bts_.*\.raw',
  },
  DELETE_ORIGINAL => 0,
  ENABLE_LOCAL_COMPRESSION => 'TRUE',
  OVERWRITE_FILES => 'TRUE',
},

# Outgoing rule
{
  RULE_DESC => 'Copy LIF files to PM server using scp',
  PROTOCOL => 'scp',
  HOST => 'pmserver',
  PROTOCOL_PATH => '/usr/local/bin',
  DIRECTION => 'OUT',
  LOCAL_DIR => "../data_dirs/output_d",
  REMOTE_DIR => "/spool/loader/nokia",
  TIMESTAMP_FILE => "./lock_dir/.otimestamp",
  INPUT_FILE_DESCRIPTION => {
    NOKIA => '.*.lif',
  },
  DELETE_ORIGINAL => 'TRUE',
  BULK_TRANSFER   => 'TRUE',
  STATISTICS_FILENAME => 'transfer_nokia', # output transfer stats
  OUTPUT_FORMAT   => 'LIF_Writer',
}
```

4.2.1 The timestamp file

This section explains the usage of the timestamp file within the different protocols. As stated above the timestamp file must be unique for each rule entry.

- ftp: When configured with the ftp protocol, the TransferEngine will create the timestamp file in the local directory on the local server. Within this file it stores the modification time of the last file, and the list of the last files copied. This is to ensure that if using drip feed of files, it will know which files with a certain modification time have already been copied.

  The timestamp file for ftp consists of the last modification time for each subdirectory that it traverses into depending on the DEPTH specified. Below is a sample timestamp file format for ftp:

```
$mod_time = {
  '/<remote_dir_1>' => {
    'LAST_TIME' => <last_file_mod_time>,
  },
  '/<remote_dir_2>' => {
    'LAST_TIME' => <last_file_mod_time>,
  },
};
```

  Where `<remote_dir_#>` is the full path of the directories on the remote server, and `<last_file_mod_time>` is the modification time of the last file transferred in POSIX time. The time should be consistent with the time from the previous timestamp file format.

- scp/rcp: When configured with the scp/rcp protocol, the TransferEngine will create the timestamp file in the remote directory on the remote server. The engine uses this file in combination with the "find" command to get a list of files which are newer than the timestamp file on the remote server. It then filters the files based on the INPUT_FILE_DESCRIPTION. The timestamp file does not contain any information, only the modification time of the file is used. Drip feed cannot be used with scp/rcp unless DELETE_ORIGINAL is also set to true.

  The timestamp file for scp/rcp is an empty touched file with the modification time set using the local time of the transfer for the Gateway run.

4.3 Engine Configuration

The configuration for the engine stage, where the raw performance data is parsed in the standardised PIF format, is contained in EngineConfig.pm.

This configuration can consists of a number of rule entries, depending on the complexity of the vendor data and the number of formats supported. The configuration entries and values within each of these depends on the Vendor Engine Rule.

This section describes the standard entries, mandatory and optional, that apply to all Vendor Engine rules.

The mandatory entries for all rules are:

- RULE_TYPE: The name of the Vendor Engine that is being executed. This must match the name of the Perl module in the parsersrc directory.

  ```
  RULE_TYPE => 'NOKIA_ASCII'
  ```

- INPUT_FILE_DESCRIPTION: A scalar or array specifying the regular expressions which the files must match. This is to ensure only the appropriate raw files are passed to the vendor engine rule for processing. Can be configured as single entry:

  ```
  INPUT_FILE_DESCRIPTION => 'BSC.*.raw'
  ```

  or an array of different filenames

  ```
  INPUT_FILE_DESCRIPTION => ['BSC.*\.raw', NSS.*\.raw']
  ```

The optional configuration entries are:

- RULE_DESC: A text string that will be logged to the audit file. This can be useful for tracing execution of the same engine rule.

  ```
  RULE_DESC => 'Parse Nokia ASCII BSC raw performance data
  ```

- NUMBER_OF_FILES_TO_PROCESS**:** The number of files to be processed at one time through the Engine and Post-parser. This is used to drip feed files through the system in a backlog scenario.
- ORDER_OF_FILES: The order in which files are to be processed. These options are:
  - YOUNGEST_FIRST
  - OLDEST_FIRST
  - DIRECTORY_ORDER
  - FILENAME_ASCENDING
  - FILENAME_DESCENDING
- FILENAME_HEADER_FIELDS: A set of new counters extracted from the filename, which are passed to the vendor engine rule. In the example below the date and time are being extracted from the filename.

  ```
  FILENAME_HEADER_FIELDS => {
      DATE => 'BSC\.(\d{8}).*'
      TIME => 'BSC\.\d{8}(\d{2}:\d{2})'
      }
  ```

- DIRECTORY_HEADER_FIELDS: Similar to above, with fields extracted from the directory path to the raw file.
- INPUT_DIR_DEPTH: The maximum depth in the raw input directory to follow. For example:

  ```
  INPUT_DIR_DEPTH => 1
  ```

  the engine will process all files in the raw directory, and files down one further directory level.

- DO_NOT_DELETE: Do not delete the raw files or move them to the storage directory.

4.4 Post Parser Configuration

Once the Engine stage is complete, the intermediate PIF files are passed to the Post Parser stage. The post parser stage is configured in UserConfig.pm. The rules are processed in the order that they appear in UserConfig.pm, so the output of one Post Parser rule can feed into the next. Two types of Post Parser rules may be configured:

- Standard: These are the Post Parser rules shipped as standard with the Gateway Framework. These rules fulfil most of the standard functions required to create the output performance data. Standard Post Parser rules contain a full description of these standard rules.
- Vendor Specific: Where the standard rules don't meet the user requirements for output, Vendor Specific Post Parser rules are designed to meet the specific needs of the Vendor data. These are shipped with the vendor Gateway.

The specific configuration entries for each Post Parser rule depend upon the functionality it offers. However all Post Parser rules support some standard entries which are configured for each rule.

- RULE_TYPE: The Post Parser rule to run. This must be the same as the name of the Perl module that implements the rule.

  ```
  RULE_TYPE => 'JOIN'
  ```

- INPUT_FILE_DESCRIPTION: The list of files to match against this rule.

  ```
  INPUT_FILE_DESCRIPTION => [
  '^P_NBSC_TRAFFIC-#-(\d{8}.*\d{2}:\d{2}-#-BSC-#-\d+-#-\d+)-#-I.pif',
  '^P_NBSC_RES_AVAIL-#-(\d{8}.*\d{2}:\d{2}-#-BSC-#-\d+-#-\d+)-#-I.pif'
  ]
  ```

- RULE_DESC: A text string that will be logged to the audit file. This can be useful for tracing execution of the same engine rule.

  ```
  'RULE_DESC' => 'Join P_NBSC_TRAFFIC, P_NBSC_RES_AVAIL files',
  ```

- PRODUCE_PIF: Whether or not to produce PIF output from the rule.

```
PRODUCE_PIF => 'True'
```

- OUTPUT_FORMAT: The output format module to use to write out the final data. For LIF data specify:

```
OUTPUT_FORMAT => 'LIF_Writer'
```

- HEADER_COUNTERS_ORDER: The order to output the header counters in. The entry can either contain, a list of counter names; a hash of block names with a list of counter names for each block; or the filename to an ASCII file containing a list of counter names in a single column.

  For LIF_Writer, XML_Writer and CSV_Writer, only list the counters that are required to be sorted and appear at the start of the output block. All other counters in the output data will be sorted lexically. If the entry is not set, no sorting is performed.

  For Prospect_Writer, it is compulsory to provide this entry with a list of all the counters that are required to be sorted and appear in the output block. All other counters will be ignored. If the entry is not set, no counters will be output.

```
HEADER_COUNTERS_ORDER => [ qw(BSC DATE STARTTIME) ],
```

- DATA_COUNTERS_ORDER: Similar to HEADER_COUNTERS_ORDER, the order to output the data counters in.
- OUTPUT_RECORD_KEY: A list of data counters to form the record key as the first column of the output block.
- OUTPUT_RECORD_KEY_DELIMITER: A string to separate the counter names for the output record key. If not set, the default '-#-' is used.

Refer Standard Post Parser rules for information on the complete set of rules available and their use.

4.5 Statistics Configuration

The Gateway Framework supports 4 different types of statistics:

1. File Statistics – Statistics on the raw performance data files processed and the output LIF files produced.
2. Block Statistics – Statistics on the names, number of instances of each block name in the output, and the number of counters in each block.
3. Counter Statistics – Statistics on the names of counters present in both the raw and LIF data.
4. Transfer Statistics – Statistics on the raw performance files being transferred in, and the LIF files transferred off the server. See Transfer Configuration for more information.

Statistics serve 2 main purposes with the Gateway Framework:

1. Verify the integrity and completeness of data being presented to the Gateway for parsing and the output data for loading to the PM system.
2. Self reporting of the type, amount (number of files) and size (of files) being parsed by the Gateway.

All statistics are produced in a configurable output format so that they can be loaded directly into the PM system.

Note that the File and Block statistics form the basis of the event notification message being sent to the notification engine for dispatching.

There is an overhead in running statistics on the raw and output performance data. To turn off statistics completely, remove or comment out the entries from the statistics configuration file, StatisticsConfig.pm.

The entries relating to the different areas for statistics collection are described below.

4.5.1 File Statistics

File statistics are collected for each iteration of the Gateway. These statistics are then summarised at a configurable interval, the summary period. For example the Gateway may be executing every 15 minutes, but the File Statistics are summarised once a day.

File Statistics collect the following statistics on the raw files parsed by the Gateway:

- summary start date and time
- summary end time
- measurement type
- number of raw files parsed (successfully, failed, 0 size)
- total size of raw files (in KB)
- the number of files in each measurement period. This is a sliding scale, with the number of files for each 30 minute period for the first 2 hours, then hourly periods.
- The calculations are based on the time difference between the current time of parsing and the end time of the performance measurement file period. These values can therefore be used to detect backlog buildup.
- total parsing time (in seconds)

For the output file data the following information is captured:

- The number of output files created.
- The size of these output files.

These statistics are stored in intermediate PIF files during the Gateway execution and then, depending on the configured summary period, joined to create the final output file.

The configuration of File_Statistics is described below:

- RULE_TYPE: The rule type, must be set to “File_Statistics”

```
RULE_TYPE => 'File_Statistics'
```

- RULE_DESC: A brief description of the rule.

```
RULE_DESC => 'Get file statistics for raw and LIF data'
```

- STATISTICS_FILE_DESCRIPTION: The regular expressions listing the input raw files that will be matched by this rule. Statistics will be gathered for the set of files and grouped by type.

```
STATISTICS_FILE_DESCRIPTION => ['HLR','VLR'],
```

- STATISTICS_HEADER_ENTRIES: A list of counter names and values to insert into the output statistics file.

```
STATISTICS_HEADER_ENTRIES     => {
                                        'GatewayID' => "3GPP_XML",
                                        'Network'   => "NSS",
                                        'Version'   => "1.2",
                                   },
```

- STATISTICS_HEADER_MAPPINGS: A list of counter names to be mapped into the statistics output header. The hash keys are the header counter names to use in the output statistics header data. The hash values are the actual header counter names from the engine PIF output for one raw file processing whose values are mapped to the output statistics header data. Note that it the engine module is responsible for populating these and it is critical that this mapping works in order to collect File Statistics.

  The entries StartDate, StartTime, EndTime and Type are all mandatory and must be mapped.

```
STATISTICS_HEADER_MAPPINGS    => {
                 'StartDate' => 'StartDate',
                 'StartTime' => 'StartTime',
                 'EndTime'   => 'EndTime',
                 'Type'      => "Network_FileType"
                                },
```

- HEADER_INFO_FOR_STATS_FILE: The list of header values to use when creating the name of the statistics file. These can be from either the

  STATISTICS_HEADER_MAPPINGS or STATISTICS_HEADER_ENTRIES list

```
HEADER_INFO_FOR_STATS_FILE => [
                     qw(GatewayID Type ParsingDate ParsingTime)]
```

- STATISTICS_OUTPUT_DIRECTORY: The output directory to write the File Statistics files to for collection. The directory must exist and be read/writable.

```
STATISTICS_OUTPUT_DIRECTORY   => '../file_statistics/',
```

- LIF_TYPE_MAPPINGS: This provides a map between the input raw files and the resultant output files, the LIF. Each entry consists of the Type (derived in the STATISTICS_HEADER_MAPPINGS) being mapped to a series of REs which match the LIF files which are produced by those raw file types (HLR/VLR etc).

```
LIF_TYPE_MAPPINGS => {
    NSS => ['HLR','MSC'],
    BSS => 'BSC',
},
```

- STATISTICS_SUMMARY_PERIOD: The period (in minutes) to summarise file statistics. This is to allow file statistics to be accumulated independently of the scheduled interval of the Gateway.

```
STATISTICS_SUMMARY_PERIOD      => 1440, # One day.
```

  The gateway checks to see if the summary period has passed. If true, then the existing file statistics PIF files generated during the summary period are summarised and outputted using Post Parser JOIN and ACCUMULATE rules.

  The file statistics current in the Gateway statistics buffers for the current execution will not be included in the summary. These will be outputted in PIF format and summarised in the next summary.

- OUTPUT_FILENAME_START: An optional string to prepend to the output filename.

```
OUTPUT_FILENAME_START          => 'File_Statistics',
```

- DATA_KEY_NAME: A dummy key to insert into the PIF files which can then be used by the JOIN and ACCUMULATE rules when summarising the results over the STATISTICS_SUMMARY_PERIOD.

```
DATA_KEY_NAME                  => 'FS_Key',
```

A full example of the rule configuration is contained in StatisticsConfig.pm, supplied with the Vendor Gateway.

Note that the file statistics included in the event notification message are based on the results for the just completed execution of the Gateway, and do not rely on the summarised data.

4.5.2 Block Statistics

The block statistics are collected on the output formatted files. The block statistics are output directly to the configured output format file. Note that the functionality has been updated to only output one block statistics file per execution of the Gateway.

Each block statistics output file contains:

- the start date and time
- the end time
- the measurement type

And a block in the block statistics output file for each block in the output data containing:

- the block name
- the number of blocks of this name
- the number of counters in this block

```
{
  {
    StartTime 10:30
    EndTime 20050810104500Z
    GatewayID 3GPP_XML
    Network Ericsson_UTRAN
    Version 3
    Type statsfile
    SUB_TYPE ManagedElement_RncFunction_UtranCell
    StartDate 10Aug2005
    MeasurementType UTRAN
    Ericsson_UTRAN_BLOCK_Stats {
      Block_name ManagedElement_TransportNetwork_Aal2Sp
      Number_of_blocks 12
      Number_of_counter_per_Block_records 6
    }
    Ericsson_UTRAN_BLOCK_Stats {
      Block_name ManagedElement_TransportNetwork_SccpSp_SccpScrc
```

```
        Number_of_blocks 6
        Number_of_counter_per_Block_records 16
      }
      Ericsson_UTRAN_BLOCK_Stats {
        Block_name ManagedElement_Equipment_Subrack_Slot_PlugInUnit
        Number_of_blocks 414
        Number_of_counter_per_Block_records 8
      }
  …
```

Sample Block statistics output.

The block statistics configuration entries are:

- RULE_TYPE: The type of statistics rule, must be Block_Statistics

```
RULE_TYPE => 'Block_Statistics'
```

- RULE_DESC: Optional description of the block statistics rule.

```
RULE_DESC => 'Derive block statistics for XML data'
```

- STATISTICS_FILE_DESCRIPTION: The list of REs, one of which the LIF files must match, for statistics collection from that LIF. The entry below matches all output LIF files.

```
STATISTICS_FILE_DESCRIPTION => ['.*\.lif'],
```

- STATISTIC_FILE_ENTRIES: The list of entries that will be extracted from the LIF filename, which can be subsequently used in the block statistics filename or header.

```
STATISTICS_FILE_ENTRIES =>  {
    SUB_TYPE => '^(\w{3})',
},
```

- STATISTIC_HEADER_ENTRIES: The list of counter name/value attributes that will be printed to the LIF header.

```
STATISTICS_HEADER_ENTRIES     => {
     GatewayID => "3GPP_XML",
     Network   => "NSS",
     Version   => "1.2",
     MeasurementType => "HLR",
},
```

- STATISTICS_HEADER_MAPPINGS: The names of header entries from the output file to map to the statistics header. These are hash entries where the key represents the name of the counter that will be outputted in the block statistics output file and the value represents the name of the counter in the output file from which we are collecting statistics.

```
STATISTICS_HEADER_MAPPINGS => {
       StartDate => 'DATE',
       Time => 'TIME',
    },
```

- STATISTICS_UNKNOWN_SUB_TYPE: A list of entries that have been extracted from the LIF filename, using STATISTICS_FILE_ENTRIES. These entries will then be used as part of the block statistics filename.

```
STATISTICS_UNKNOWN_SUB_TYPE => 'NULL',
```

- STATISTICS_FILE_ENTRIES_FOR_HEADER: A list of entries that have been extracted from the LIF filename, using STATISTICS_FILE_ENTRIES. These entries will then be used as part of the block statistics filename.

```
STATISTICS_FILE_ENTRIES_FOR_HEADER => [qw(SUB_TYPE)],
```

- HEADER_INFO_FOR_STATS_FILE: The values from the STATISTICS_HEADER_MAPPINGS and STATISTICS_HEADER_ENTRIES to use to create the filename. These can be used to ensure uniqueness of the block statistics filenames.

```
HEADER_INFO_FOR_STATS_FILE    => [
                     qw(GatewayID MeasurementType DATE TIME)],
```

- STATISTICS_BLOCKS_DESCRIPTION: The list of REs, which the block name must match, for statistics to be gathered. The sample below matches all blocks.

```
STATISTICS_BLOCKS_DESCRIPTION => ['.*'],
```

- STATISTICS_OUTPUT_DIRECTORY: The name of the directory to write the block statistics files out to. This directory must exist and be writable.

```
STATISTICS_OUTPUT_DIRECTORY => '../block_statistics'
```

- BLOCK_NAME: The name to use for each of the statistics blocks written to the LIF.

```
BLOCK_NAME => 'GATEWAY_BLOCK'
```

- OUTPUT_FILENAME_START: An optional text string to prepend to the block statistics filename.

```
OUTPUT_FILENAME_START => 'block_stats'
```

A full configuration can also be seen in the StatisticsConfig.pm file.

A summary of the Block Statistics data is used in the notification event command sent to the notification engine if Block Statistics gathering has been configured.

4.5.3 Counter Statistics

Counter statistics are collected both from the raw input files and the LIF output. The counter statistics are outputted to a single LIF file for each Gateway execution.

The counter statistics are output, one counter per block, with the following information:

- the name of the counter.
- the group (block name) it belongs to.
- whether it was in the configured list for monitoring.
- whether it was present in the raw file.
- whether it was present in the LIF file.

The counter statistics configuration entries are:

- RULE_TYPE: The type of rule, must be Counter_Statistics

```
RULE_TYPE => 'Counter_Statistics'
```

- RULE_DESC: Optional description of the block statistics rule.

```
RULE_DESC => 'Derive counter statistics for XML data'
```

- STATISTICS_HEADER_ENTRIES: The list of header names and values to insert into the LIF header block.

```
STATISTICS_HEADER_ENTRIES     =>
     {
         GatewayID => "Nortel",
         Network   => "GPRS CN",
         Version   => 1.2,
         SenderType => "HLR",
         SenderID   => "HLR-MSCW",
     },
```

- STATISTICS_HEADER_MAPPINGS: This hash contains the names of the header counters to be mapped from header counters in the LIF block. Each entry is in the form

```
<output_name> => <lif block name>
```

  where <output_name> is the name of the counter in the Counter Statistics file and <lif block name> is the name in the output LIF block.

```
STATISTICS_HEADER_MAPPINGS    =>
     {
         'START_DATE'  => 'StartDate',
         'START_TIME'  => 'StartTime',
         'END_TIME'    => 'EndTime',
         'SENDER_ID'   => 'MeasType',
     },
```

- HEADER_INFO_FOR_STATS_FILE: An array containing the list of entries from STATISTICS_HEADER_ENTRIES and STATISTICS_HEADER_MAPPINGS that are to be used in creating the counter statistics LIF filename.

```
HEADER_INFO_FOR_STATS_FILE    =>
                      [qw(GatewayID START_DATE START_TIME SenderType)],
```

- STATISTICS_OUTPUT_DIRECTORY: The path to the directory for the counter statistic files.

```
STATISTICS_OUTPUT_DIRECTORY   => '../counter_stats'
```

- COUNTER_LIST: The list of counters that are specifically required to be checked for in the raw and LIF file. Any counter that is contained in this list will have IN_CONFIG set to true in the output LIF. All other counters found which are not in this list will have IN_CONFIG set to FALSE.

```
COUNTER_LIST => [qw (VS.C7LKSYNU VS.LKFAIL
              VS.C7COV VS.C7ONSET1) ],
```

- DISABLE_STATS: Used to disable Counter_Statistics in the Gateway if required. If set to TRUE, no counter stats will be run. This may be done for performance reasons if there is a problem with backlogs or load.

```
DISABLE_STATS => 0,
```

- OUTPUT_FILENAME_START: A string to prepend to the output LIF name, if required.

```
OUTPUT_FILENAME_START => "C_STATS",
```

The full configuration for Counter_Statistics can be seen in StatisticsConfig.pm.

4.6 Notification

Historically it has not always been easy to monitor the activity of a Gateway, especially with multiple vendor Gateway installations. One of the reasons for this was that all activities were reported to log and audit files, which are not always very accessible.

The Gateway Framework has been updated to include configurable notification functionality that will send north-bound event and alarm notifications. These notifications should logically go to the application consuming the output data from the Gateway. The current release of the Gateway has support for the following application notification API's:

- Metrica/PAS for Metrica/NPR – npralarm (supporting both event and alarm messages)
- MPM Notification Command Line Interface – mpmalarm (supporting both event and alarm messages)

The Notification Engine dispatches 2 types of activities from the Gateway Framework:

- Alarm activities – critical exit status and errors during processing of data.

- Event activities – statistical information with regards to blocks and files processed.

Alarm Notifications

Alarm notifications are sent for the following system errors:

- Engine module failure.
- Disk space failures.
- Failure to fork processes.
- File transfer failures.

Note that configuration failures will not generate alarm notification failures, as these are typically setup teething problems.

```
remsh server -l metrica '. /LOCAL/metrica/npr/setup.sh;npralarm  -target
"TmnUdp" -objectClass "Gateway" -ObjectName "slade:Ericsson UTRAN Gateway" -
name "Alarm" -text "Error parsing A20050810.1130-10530301_statsfile.xml XML
parsing failed line 1" -severity "major"' 2>/dev/null >/dev/null

remsh servername -l metrica '/Interfaces/vendor_3.1/vstart/mpmalarm  -u "Mon
Sep 26 17:00:54 2005" -n "Alarm" -o "Gateway" -x "Ftp could not open
connection to h1" -j "slade:Ericsson UTRAN Gateway" -s "major" -t "TmnUdp"'
2>/dev/null >/dev/null
```

Sample NPR and MPM alarm messages generated by notification functionality.

Event Notifications

Events notification will be dispatched, if configured, for the following events:

- Engine file parsing completed.
- Statistics of Blocks processed if Block Statistics collection is configured.
- Statistics of Files processed if File Statistics collection is configured. Note that the File Statistics summary rules (JOIN and ACCUMULATE) and functionality are not required as File Statistics event notifications are sent for each execution of the Gateway.

The File and Block statistical data is a reduced subset or summary of the normal statistical data produced. This is due to message size limitations of the receiving system.

File statistical data entries used in notification event messages are as follows:

- `Start date`
- `Start time`
- `File type`
- `O/P file size total`
- `Raw file size total`

- No files bad 0,
- No files OK

A File Statistics notification event is dispatched for every File Statistics file type collected.

Block Statistics message sends a total number of blocks and total number of block counters.

```
remsh servername -l metrica '. /LOCAL/metrica/npr/setup.sh;npralarm  -target
"TmnUdp" -objectClass "Gateway" -ObjectName "slade:Ericsson UTRAN Gateway" -
name "Event" -text "Block statistics: Total blocks 62, total no. counters  678
" -severity "indeterminate"' 2>/dev/null >/dev/null
```

Block statistics event notification command.

```
remsh servername -l metrica '. /LOCAL/metrica/npr/setup.sh;npralarm  -target
"TmnUdp" -objectClass "Gateway" -ObjectName "slade:Ericsson UTRAN Gateway" -
name "Event" -text "File stats: Type statsfile, 10Aug2005 11:30, O/P file size
total 30, Raw file size total 44, No files bad 0, No files OK 1" -severity
"indeterminate"' 2>/dev/null >/dev/null
```

File statistics event notification command.

Sample NPR event command messages generated by notification functionality.

4.6.1 Notification Configuration

The specific configuration entries, for each Alarm Activities and Event Activities rule depends on the Notification Type. All Notification type uses the same standard entries, which are configured for each rule.

Only one event and one alarm notification rule entries are allowed in the configuration. The type of rule hash entry is governed by NOTIFICATION_TYPE. This reflects the design where there is only one gathering source each for all event and alarm notifications respectively.

The following are the standard notification entries to be included in the NotificationConfig.pm module

- RULE_TYPE: The rule type to dispatch message. This must be the same as the name of the Perl module that implements the notification system. Note that this modularised approach also facilitates the inclusion of new notification modules as required.

  RULE_TYPE => 'NPR_Notification'

- RULE_DESC: A text string that will be logged to the audit file. This can be useful for tracing execution of the notification rule.

  'RULE_DESC' => 'NPR Alarm notification',

- NOTIFICATION_TYPE: The notification activity to dispatch. This is either 'Alarm', or 'Event'. Note that only one of each is allowed in the configuration.

  NOTIFICATION_TYPE => 'Alarm'

- NOTIFICATION_TARGET: A text string of the target defined in the notification system.

```
NOTIFICATION_TARGET => 'TmnUdp'
```

- NOTIFICATION_COMMAND: The command line script to dispatch message.

```
NOTIFICATION_COMMAND => 'npralarm'
```

- REMOTE_SHELL_COMMAND: The command line to execute a remote shell on the remote server. If a remote command is not required, as is the case if the Gateway is sharing a host system with the service assurance application, then the REMOTE_SHELL_COMMAND value will reflect this.

```
REMOTE_SHELL_COMMAND => 'rsh server_name -l metrica'
```

- SYSTEM_ENV_SETUP: The environment setup scripts or commands for the notification system.

```
SYSTEM_ENV_SETUP => '. /LOCAL/metrica/npr/setup.sh'
```

A sample configuration of the Notification Engine for both Alarm and Event activities rules is shown below.

```
{
  RULE_TYPE => ' NPR_Notification',
  RULE_DESC => 'NPR Alarm Notification',
  NOTIFICATION_TYPE => 'Alarm',
  NOTIFICATION_COMMAND => 'npralarm',
  NOTIFICATION_TARGET => 'AlarmTmnUdp',
  REMOTE_SHELL_COMMAND => 'rsh servername -l metrica',
  SYSTEM_ENV_SETUP => '. /LOCAL/metrica/npr/setup.sh',
},
{
  RULE_TYPE => 'NPR_Notification',
  RULE_DESC => 'NPR Event Notification',
  NOTIFICATION_TYPE => 'Event',
  NOTIFICATION_COMMAND => 'npralarm',
  NOTIFICATION_TARGET => 'EventTmnUdp',
  REMOTE_SHELL_COMMAND => 'rsh servername -l metrica',
  SYSTEM_ENV_SETUP => '. /LOCAL/metrica/npr/setup.sh',
},
```

4.7 Parallel Processing

The Gateway can be configured for parallel processing. This allows a single Gateway to spread the load of certain engine and Post Parser rules across multiple processors.

Before considering configuring Engine or Post Parser rules over multiple processors, the following points should be considered:

1. Has the Gateway been analysed to ensure it is configured for optimum performance? It should be ensured that the Gateway is as efficient as possible before considering splitting it over multiple processors.
2. Are there multiple processors available to run the Gateway on? The rules should not be configured to run with any more processes than there are processors on the machine.
3. Is there spare capacity that can be utilised? Ensure that the server capacity has not been exhausted already.
4. Is there an advantage to be gained? It is recommended that an engine or post parser rule is taking at least 10 seconds to process its set of files before it is considered for multi processor configuration. There is an overhead with spawning unix processes and managing them subsequently.

There are a number of internal constraints on parallel processing:

1. The PIF_Handler must be used, so that PIF files are being read/written to disk, and can be easily shared between processes. The in memory PIF_Cache cannot be used.
2. File_Statistics must not be configured.

Parallel processing is configured individually for each engine and post parser rule. For example, if there were 100 files to be processed by a particular post parser rule, these could be split over 4 processes, with each process handling 25 files. These processes run in parallel, and are monitored by the Gateway. They must complete before the next rule is initiated.

4.7.1 Configuration

Parallel Processing is configured individually for each engine and post parser rule. There are three configuration variables that control its use.

- NUMBER_OF_PROCESSES: The number of processes to split the processing of the rule over. Should not be more than the number of actual processors on the server.

  NUMBER_OF_PROCESSES => 4

- MIN_FILES_PER_PROCESS: The minimum number of files to assign per process. This value ensures that processes are only created if there are sufficient files available. For example for a rule processing very large files which each take 20 seconds to parse, it would be acceptable to assign one file per process. For other Post Parser rules, each file may take such as short period that at least 20 files per process would need to be assigned.

  MIN_FILES_PER_PROCESS => 20

- NUMBER_OF_PROCESSES_DIST (Optional): This entry can be used to configure different process distributions over certain hourly periods. For example it may be required that only 2 processes run at certain hours during backup periods, to reduce the load. The example below specifies only one process from midnight to 2 A.M. and two processes from 3 A.M. to 4 A.M.

```
NUMBER_OF_PROCESSES_DIST => {
      '00-02' => 1,
      '03-04' => 2,
    }
```

A sample configuration of the Post Parser PIF_2_OUTPUT rule, splitting files over 4 processes is shown below.

```
{
    RULE_TYPE => 'PIF_2_OUTPUT',
    RULE_DESC => 'Convert all BSC PIFs to LIF',
    INPUT_FILE_DESCRIPTION => '.*BSC.*\.pif',
    OUTPUT_FORMAT => 'LIF_Writer',
    NUMBER_OF_PROCESSES => 4,
    MIN_FILES_PER_PROCESS => 10,
}
```

All engine rules support parallel processing.

4.7.2 Log and Audit output

Parallel processes cannot write to the same log or audit file. Each parallel process forked will create its own log and audit file. This log and audit file will be the name of the base file suffixed by .<X> where <X> is the child number. The main Gateway process will continue to write to the main log and audit file.

For example if the Gateway is using the files log and audit, and 2 processes are being created for parallel processing, the following files will be created.

```
log.0, log.1, audit.0, audit.1
```

4.7.3 Post Parser rules supporting parallel processing

Certain rules cannot support parallel processing due to their functionality requiring input of more than one file type. The Post Parser rules that support parallel processing are:

- ACCUMULATE
- AGGREGATE_LINE
- CVAL_MANIP
- DATALINE_WHERE
- FILE_SPLIT
- FILE_SPLIT_BY_COUNTERS
- MERGE_RECORDS
- PERLIZE
- PIF_2_OUTPUT
- SPLIT_RECORDS

5 Standard Post Parser rules

This chapter describes the standard Post Parser rules supplied with the Gateway Framework.

These rules are configured in UserConfig.pm. If they don't meet the requirements for the manipulation of output data they can be augmented by vendor specific Post Parser rules.

Each rule is accompanied by a sample of the input and output PIF if applicable, to demonstrate its use.

Where configuration is not straightforward or is in a complex structure, an example is included in the description.

5.1 ACCUMULATE

The accumulate supports:

- accumulation of counter values over a number of data rows given a counter key to match data rows.
- deriving maximum/minimum values for counters over a number of data rows given a counter key to match data rows

5.1.1 Sample rule application

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
```

In this example the counters, COUNT1, COUNT2 and COUNT3 are being accumulated using CELL value as the row key, with the maximum and minimum of the LOAD counter also being derived.

The output after processing by ACCUMULATE:

```
##START|ACCUM
CELL|LOAD_MIN|COUNT1|COUNT2|LOAD_MAX|ACCUM_NUM_IN_SUM|COUNT3
DF0001|10|4|8|100|4|12
DF0002|0|4|8|60|4|12
##END|ACCUM
```

Note in the output:

The three counters COUNT1, 2 and 3 have been accumulated across the CELL. The minimum and maximum value have been placed in MIN_LOAD and MAX_LOAD respectively. The number of rows that matched the key to create the accumulated values is contained in ACCUM_NUM_IN_SUM.

5.1.2 Configuration

This section describes the rule specific configuration entries for ACCUMULATE. It does not include the standard entries supported by every Post Parser rule, detailed in the section: Post Parser 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.
- APPEND_STR: A string to appended to the accumulated counters
- OLD_COUNTER_NAMES: A list of old counters to rename. NEW_COUNTER_NAMES must also be configured.
- NEW_COUNTER_NAMES: The list of the new counter names to replace the OLD_COUNTER_NAMES with.

- COUNTER_NULL_VALUE: The value to insert for the accumulated counters when there is an error calculation their value, for example when they are non-numeric values.
- MAXIMUM_COUNTERS: the list of counters to derive the maximum values for.

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

- MAXIMUM_APPEND_STR: the string to append to the MAXIMUM_COUNTERs to create the name of the new counter.

```
MAXIMUM_APPEND_STR => "_MAX",
```

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

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

- MINIMUM_APPEND_STR – the string to append to the MINIMUM_COUNTERs to create the name of the new counter containing the minimum value.

```
MINIMUM_APPEND_STR => "_MIN",
```

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

```
{
     RULE_TYPE => 'ACCUMULATE',
     RULE_DESC => 'Accumulate Sample',
     INPUT_FILE_DESCRIPTION => ['.*'],
     COUNTERS_TO_SORT_ON => [qw(CELL)],
     OUTPUT_BLOCK_NAME => 'ACCUM',
     REDUNDANT_DATA_COUNTERS => [],
     PRODUCE_PIF => 'True',
     OUTPUT_FORMAT => 'LIF_Writer',
     NON_ADDITIVE_COUNTERS => [],
     MAXIMUM_COUNTERS => [qw(LOAD)],
     MINIMUM_COUNTERS => [qw(LOAD)],
     MAXIMUM_APPEND_STR => "_MAX",
     MINIMUM_APPEND_STR => "_MIN",
}
```

5.2 AGGREGATE_LINE

This rule, similar to ACCUMULATE, accumulates a set of values. But rather than accumulate across a set of rows, the values are accumulated from a set of counters on a particular row.

The set of counters for accumulation are matched via a series of regular expressions. All counters matching the regular expression are accumulated into a single value.

5.2.1 Sample Application

Given the following input block:

```
##START|DATA_BLOCK
C_1_1|C_1_2|C_1_3|C_1_4|C_2_1|C_1_5|C_2_2|C_2_3
20|20|10|30|20|30|10|20
10|0|40|20|0|0|0|10
20|20|30|30|10|0|0|10
10|30|40|0|20|0|10|20
30|0|10|0|20|30|10|20
##END|DATA_BLOCK
```

The set of 5 counters C_1_1 to C_1_5 and the set of counters C_2_1 to C_2_3 are going to be totalled for each row of data into 2 new counters, C_1_Total and C_2_Total.

```
##START|DATA_BLOCK
C_1_Total|C_1_1|C_1_2|C_1_3|C_1_4|C_2_1|C_1_5|C_2_2|C_2_3|C_2_Total
110|20|20|10|30|20|30|10|20|50
70|10|0|40|20|0|0|0|10|10
100|20|20|30|30|10|0|0|10|20
80|10|30|40|0|20|0|10|20|50
70|30|0|10|0|20|30|10|20|50
##END|DATA_BLOCK
```

5.2.2 Configuration

This section describes the rule specific configuration entries for AGGREGATE_LINE. It does not include the standard entries supported by every Post Parser rule, detailed in the section: Post Parser Configuration.

The mandatory configuration entries are:

- DEFAULT_NULL_VALUE: The value to insert for the new counter, if no counters in the data match it for accumulation.
- COUNTER_GROUPS: A hash containing a list of new counter names, each of which maps to an existing set of counters to be accumulated using a RE.

```
COUNTER_GROUPS => {
      C_1_Total => 'C_1_\d+',
      C_2_Total => 'C_2_\d+',
```

```
    },
```

The optional configuration entries are:

- COUNTER_NAME_TO_EXTRACT: As well as defining new counters to be created in COUNTER_GROUPS, this entry can be used to define a sub-string to extract from the counter name. This will prefix the new COUNTER_GROUP counters created.
- OUTPUT_BLOCK_NAME: The name to use for the output block.
- REDUNDANT_DATA_COUNTERS: A list of data counters to remove from the output.
- REDUNDANT_HEADER_COUNTERS: A list of header counters to remove from the output.

The configuration used for the sample application is below:

```
{
      RULE_TYPE => 'AGGREGATE_LINE',
      RULE_DESC => 'Sample A/L configuration',
      INPUT_FILE_DESCRIPTION => '.*',
      REDUNDANT_DATA_COUNTERS => [],
      DEFAULT_NULL_VALUE => 'NULL',
      COUNTER_GROUPS => {
            C_1_Total => 'C_1_\d+',
            C_2_Total => 'C_2_\d+',
      },
}
```

5.3 BATCHFILES

The BATCHFILES rule batches, or joins, a number of files together to create one large output file, usually a LIF. The batching of smaller PIF objects is usually performed to reduce the number of files being presented to the PM system for loading.

5.3.1 Sample Application

Given a list of PIF files:

```
BSC-#-02Mar2004-#-1200-#-103-#-I.pif
BSC-#-02Mar2004-#-1200-#-132-#-I.pif
BSC-#-02Mar2004-#-1200-#-131-#-I.pif
NSS-#-02Mar2004-#-1200-#-172-#-I.pif
NSS-#-02Mar2004-#-1200-#-131-#-I.pif
```

```
NSS-#-02Mar2004-#-1200-#-173-#-I.pif
```

In this example the BSC and NSS files will be batched together to create a single BSC and NSS file for loading:

```
BATCHED-#-BSC-#-02Mar2004-#-1200-#-1-#-BF.lif
```

```
BATCHED-#-NSS-#-02Mar2004-#-1200-#-1-#-BF.lif
```

The files are prefixed with the "BATCHED" string, followed by the matched pattern from the INPUT_FILE_DESCRIPTION, in this case the type and date time, for example "BSC-#-02Mar2004-#-1200".

5.3.2 Configuration

This section describes the rule specific configuration entries for BATCHFILES. It does not include the standard entries supported by every Post Parser rule, detailed in the section: Post Parser Configuration.

The optional configuration entries are:

- OUTPUT_BLOCK_NAME: The block name to use in the output data
- REDUNDANT_HEADER_COUNTERS: A list of header counters to remove in the output data
- REDUNDANT_DATA_COUNTERS: A list of data counters to remove in the output data
- OUTPUT_FILENAME_START: A string to prepend the filename with
- HOURS_TO_WAIT_FOR_PARTNER_FILES: The number of hours to wait after the last file before the files present are processed without their partner files. If set to -1, it will not wait for partner files.
- QUOTE_INPUT_PIFS: If set to true the names of the PIFs used to create the output performance LIF will be included as a comment at the top of the file.

The configuration used for the sample application:

```
{
      RULE_TYPE => 'BATCHFILES',
      RULE_DESC => 'Batch BSC and NSS PIFs',
      INPUT_FILE_DESCRIPTION => [
            '(BSC.*)-#-\d+-#-I\.pif',
            '(NSS.*)-#-\d+-#-I\.pif'
      ],
      REDUNDANT_DATA_COUNTERS => [],
      PRODUCE_LIF => 'True',
      HOURS_TO_WAIT_FOR_PARTNER_FILES => -1,
```

```
        OUTPUT_FILENAME_START => 'BATCHED',
        REDUNDANT_HEADER_COUNTERS => [],
}
```

5.4 CVAL_MANIP

The CVAL_MANIP Post Parser rule provides the functionality to manipulate counter values. Example of it use are:

- to wrap a counter value in quotes
- remove a substring from a counter value
- replace a counter value
- rearrange a counter value

5.4.1 Sample Application

Given the PIF input:

```
##START|BLOCK_DATA
C_3|C_4|C_5|C_1|C_2
3|4|5|1  10|2
3|4|5|2  14|2
3|4|5|KEY-6|2
3|4|5|KEY-2|2
3|4|5|KEY-8|2
##END|BLOCK_DATA
```

The counter value manipulation will be performed on counter C_1. The following operations will be performed on this counter:

- any values prefixed by “KEY-“ will have the portion of the value removed
- any values with one or more spaces will be quoted and the value replaced with one space.

The application of the CVAL_MANIP rule produces the following output:

```
##START|BLOCK_DATA
C_3|C_5|C_1|C_2
3|5|"1 10"|2
3|5|"2 14"|2
3|5|6|2
3|5|2|2
3|5|8|2
```

```
##END|BLOCK_DATA
```

5.4.2 Configuration

This section describes the rule specific configuration entries for CVAL_MANIP. It does not include the standard entries supported by every Post Parser rule, detailed in the section: Post Parser Configuration.

The mandatory configuration entries are:

- CNAME_MANIP: The name of the counter to be manipulated.

```
CNAME_MANIP => 'C_1',
```

- MATCH: A list of regular expressions (in order of preference) that the counter values are expected to match in order for the values to be manipulated. Each regular expression should contain at least one pattern to be extracted if the value is to be manipulated.

```
MATCH => ['^(KEY)\-(\d+)','^(\d+)\s+(\d+)'],
```

- PATTERN: A list of patterns that define how the counter value will be manipulated. There is one-to-one correspondence between the position of the regular expression in list MATCH and the patterns in PATTERN. Any occurrence of $1, $2, ... in PATTERN will be substituted by the values of the tagged expressions matched by the corresponding regular expression in MATCH. Hence the list MATCH and PATTERN must have the same number of elements.

```
PATTERN => ['$2', '"$1 $2"']
```

The optional configuration entries are:

- OUTPUT_BLOCK_NAME: The block name to use in the output data.
- OUTPUT_DIR: The name of an alternate directory that the loader files can be written for this rule. If not configured then LIFs will be written to the configured parser output directory.
- REDUNDANT_DATA_COUNTERS: A list of counters to remove from the output data.
- NON_MATCH_RECORD: Whether or not to output a PIF record whose counter value fails to match any of the patterns. If set to true, it will discard the row.

The configuration used for the sample application of the rule:

```
{
      RULE_TYPE => 'CVAL_MANIP',
      RULE_DESC => 'Counter manipulation example',
      INPUT_FILE_DESCRIPTION => ['^.*'],
      REDUNDANT_DATA_COUNTERS => ["C_4"],
      PRODUCE_PIF => 'True',
      PRODUCE_LIF => 0,
```

```
        CNAME_MANIP => 'C_1',
        MATCH => ['^(KEY)\-(\d+)','^(\d+)\s+(\d+)'],
        PATTERN => ['$2', '"$1 $2"'],
}
```

5.5 DATALINE_WHERE

The DATALINE_WHERE post parser rule is used to either remove or keep PIF data rows based on expected counter values.

5.5.1 Sample Application

Given the following input data:

```
##START|DATA_BLOCK
C_3|C_4|C_5|KEY|C_1|C_2
3|4|5|1|1|2
3|4|5|2|2|2
3|4|5|3|1|2
3|4|5|4|3|2
3|4|5|5|3|2
3|4|5|6|2|2
##END|DATA_BLOCK
```

The following operation is performed by DATALINE_WHERE to filter out PIF data rows:

- Any row with KEY value 1-5 AND with a C_1 value not equal to 1 or 3 will be kept.

This produces the output PIF:

```
##START|DATA_BLOCK
C_3|C_4|C_5|KEY|C_1|C_2
3|4|5|2|2|2
##END|DATA_BLOCK
```

Only one data row matches both criteria.

5.5.2 Configuration

This section describes the rule specific configuration entries for DATALINE_WHERE. It does not include the standard entries supported by every Post Parser rule, detailed in the section: Post Parser Configuration.

The mandatory entries are:

- COUNTER_NAMES: The entry is mandatory and is configured as an array of anonymous hashes. Each hash entry contains:
- COUNTER_NAME: the name of the counter to be checked.
- KEEP_WHERE (optional): A list of regular expressions. The counter value must match ONE of these RE's for the data line to be kept.
- REMOVE_WHERE (optional): A list of regular expressions. If the value counter matches ONE of these RE's then it will be removed.

  Either KEEP_WHERE or REMOVE_WHERE must be configured. If both are configured any data rows which pass the KEEP_WHERE list will be checked against the REMOVE_WHERE list.

```
COUNTER_NAMES =>
        [
         {
             COUNTER_NAME => 'KEY',
             KEEP_WHERE => ['^[1-5]'],
             REMOVE_WHERE => [],
         },
         {
             COUNTER_NAME => 'C_1',
             KEEP_WHERE => [],
             REMOVE_WHERE => ['^1|3$'],
         }
         ],
```

The optional configuration entries are:

- OUTPUT_BLOCK_NAME: The block name to use in the output files.
- REDUNDANT_DATA_COUNTERS: A list of counters to remove from the output data.
- ADD_NEW_COUNTER: The name of a new counter to be added to the output data.
- NEW_COUNTER_VALUE: The value of the new counter.
- FILENAME_ADDITION: A string inserted into the output filename before the '-#-DW' sequence.

The configuration below was used to produce the sample output:

```
{
      RULE_TYPE => 'DATALINE_WHERE',
      RULE_DESC => 'Dataline usage example ',
      INPUT_FILE_DESCRIPTION => ['^.*.pif'],
      REDUNDANT_DATA_COUNTERS => ['C_0' ],
```

```
        PRODUCE_PIF => 'True',
        PRODUCE_LIF => 0,
        COUNTER_NAMES =>
                [
                 {
                     COUNTER_NAME => 'KEY',
                     KEEP_WHERE => ['^[1-5]'],
                     REMOVE_WHERE => [],
                 },
                 {
                     COUNTER_NAME => 'C_1',
                     KEEP_WHERE => [],
                     REMOVE_WHERE => ['^1|3$'],
                 }
                 ],
        FILENAME_ADDITION => 'TEST'
}
```

5.6 FILE_SPLIT

The file split rule splits PIF files into smaller files based a split key made up of one or more counter values.

5.6.1 Sample Application

Given the input data:

```
##START|BLOCK
C_3|C_4|C_5|TIME|KEY|C_1|C_2
2|6|9|00:00|KEY_0|0|1
7|5|2|01:00|KEY_1|1|5
1|4|5|02:00|KEY_2|0|4
6|7|9|00:00|KEY_0|2|5
4|4|6|01:00|KEY_1|5|1
2|8|3|02:00|KEY_2|3|6
3|6|3|00:00|KEY_0|0|6
4|0|7|01:00|KEY_1|4|3
##END|BLOCK
```

This PIF data is split on the TIME and KEY counters. Each data row with a different file split key from these counter values will be output to a separate PIF.

The split key will also be used in the PIF filename to ensure the split filenames are unique.

In this case there are 3 different keys producing 3 PIFs:

File “fs_in-#-00:00-#-0-#-FS.pif” containing:

```
##START|FS
C_3|C_4|C_5|KEY|TIME|C_1
2|6|9|KEY_0|00:00|0
6|7|9|KEY_0|00:00|2
3|6|3|KEY_0|00:00|0
##END|FS
```

File “fs_in-#-01:00-#-1-#-FS.pif” containing:

```
##START|FS
C_3|C_4|C_5|KEY|TIME|C_1
7|5|2|KEY_1|01:00|1
4|4|6|KEY_1|01:00|5
4|0|7|KEY_1|01:00|4
##END|FS
```

File “fs_in-#-02:00-#-2-#-FS.pif” containing:

```
##START|FS
C_3|C_4|C_5|KEY|TIME|C_1
1|4|5|KEY_2|02:00|0
2|8|3|KEY_2|02:00|3
##END|FS
```

5.6.2 Configuration

This section describes the rule specific configuration entries for FILE_SPLIT. It does not include the standard entries supported by every Post Parser rule, detailed in the section: Post Parser Configuration.

The mandatory entries are:

- COUNTERS_USED_TO_SPLIT_FILE: A hash containing the information on how to split the file. The key is the counter names to split on and the value is a regular expression describing the portion of the counter value to extract as a key.

```
COUNTERS_USED_TO_SPLIT_FILE => {
          TIME => '(.*)',
          KEY => '^KEY_(\d+)',
```

```
},
```

- SPLIT_COUNTERS_ORDER: The order of the counter keys used to split the file in the new filename.

```
SPLIT_COUNTERS_ORDER => [ qw(TIME KEY) ],
```

The optional configuration entries are:

- OUTPUT_BLOCK_NAME: The block name to use in the output files.
- REDUNDANT_HEADER_COUNTERS: The redundant header counters to remove from the output data.
- REDUNDANT_DATA_COUNTERS: A list of counters to delete from the resultant output.

The configuration used to produce the sample application:

```
{
      RULE_TYPE => 'FILE_SPLIT',
      RULE_DESC => "splitting large files into smaller files",
      INPUT_FILE_DESCRIPTION => ['.*'],
      COUNTERS_USED_TO_SPLIT_FILE => {
            TIME => '(.*)',
            KEY => '^KEY_(\d+)',
      },
      SPLIT_COUNTERS_ORDER => [ qw(TIME KEY) ],
      PRODUCE_PIF => "True",
      PRODUCE_LIF => 0,
      OUTPUT_BLOCK_NAME => 'FS',
      REDUNDANT_DATA_COUNTERS => ['C_2'],
}
```

5.7 FILE_SPLIT_BY_COUNTERS

This rule allows you to split any PIF file into several smaller files according to the counters grouping set by the user.

A list of counter names and a tagged regular expression must be defined to create an identifier for which output file the line of data should be written to.

5.7.1 Sample Application

Given the input data:

```
##START|DATA_BLOCK
CELL|TRX|COUNT1|COUNT2|COUNT3
```

```
DF0001|1|1|2|11
DF0001|2|3|4|22
DF0002|1|9|10|11
DF0002|2|11|12|22
##END|DATA_BLOCK
```

Let us say that in this example we want to split the file on the counters COUNT1, COUNT2 and COUNT3 so that each counter is written to its own file.

The correctly configured rule would produce the following files:

file_in-#-I-#-1-#-FSC.pif:

```
##START|DATA_BLOCK
CELL|TRX|COUNT1
DF0001|1|1
DF0001|2|3
DF0002|1|9
DF0002|2|11
##END|DATA_BLOCK
```

file_in-#-I-#-2-#-FSC.pif:

```
##START|DATA_BLOCK
CELL|TRX|COUNT2
DF0001|1|2
DF0001|2|4
DF0002|1|10
DF0002|2|12
##END|DATA_BLOCK
```

file_in-#-I-#-1-#-FSC.pif:

```
##START|DATA_BLOCK
CELL|TRX|COUNT3
DF0001|1|11
DF0001|2|22
DF0002|1|11
DF0002|2|22
##END|DATA_BLOCK
```

5.7.2 Configuration

This section describes the rule specific configuration entries for FILE_SPLIT_BY_COUNTERS. It does not include the standard entries supported by every Post Parser rule, detailed in the section: Post Parser Configuration.

The mandatory entries are:

- SPLIT_CNAMES: A mandatory field that consists of a hash. The key of the hash will be a unique string that will identify the new split file data block suffix. The values of the hash will consist of a list of regular expressions of the counter names.

The optional configuration entries are:

- ONLY_INSERT: An optional list of other counter names to be reported in every record.
- WRITE_DATA_LINE: This option, when set to 'True', controls the action of this rule to output the data line even when there are no matching counters or the counters are null. When this option is not defined, by default the records without matching counters will not be output.
- OUTPUT_BLOCK_NAME: The block name to use in the output files
- REDUNDANT_HEADER_COUNTERS: The redundant header counters to remove from the output data.
- REDUNDANT_DATA_COUNTERS: A list of counters to delete from the resultant output.

The configuration used to produce the sample application:

```
{
  RULE_TYPE => 'FILE_SPLIT_BY_COUNTERS',
  RULE_DESC => 'Split file by COUNT',
  INPUT_FILE_DESCRIPTION => 'file_in-#-I\.pif',
  SPLIT_CNAMES => {
     '1' => [ 'COUNT1' ],
     '2' => [ 'COUNT2' ],
     '3' => [ 'COUNT3' ],
  },
  ONLY_INSERT => [ qw(CELL TRX) ],
  PRODUCE_PIF => 'True',
  PRODUCE_LIF => 0,
},
```

5.8 INFOINSERT

The info insert rule inserts counter data from a secondary information file, into a primary data file. It is typically used to insert hierarchy data from a configuration file into a main file, based on a counter key.

These counter keys may be made up of one or more counters, with both header and data counters configurable for both the primary and secondary files.

5.8.1 Sample Application

INFOINSERT requires at least 2 input files for processing. The primary PIF data:

```
##START|BLOCK
C_3|C_4|C_5|OBJ_ID|C_1|C_2
3|4|5|1|1|2
3|4|5|1|1|2
3|4|5|3|1|2
3|4|5|4|1|2
3|4|5|5|1|2
##END|BLOCK
```

and a secondary information file, to insert data from:

```
##START|BLOCK
CELL|INFO_KEY|TRX
10-0|1|10-10-49
10-0|2|10-10-50
10-2|3|10-10-60
10-3|4|10-10-69
10-3|5|10-10-79
10-3|6|10-10-50
10-0|7|10-10-22
10-2|8|10-10-80
10-3|9|10-10-77
10-7|10|10-10-27
##END|BLOCK
```

In this example the key matched between the files is the OBJ_ID from the primary file and INFO_KEY from the secondary file. The counters CELL and TRX are being inserted from the secondary file.

This produces the following output:

```
##START|INS_BLOCK
C_3|C_4|C_5|CELL|INFO_KEY|OBJ_ID|C_1|TRX|C_2
3|4|5|10-0|1|1|1|10-10-49|2
3|4|5|10-0|1|1|1|10-10-49|2
3|4|5|10-2|3|3|1|10-10-60|2
3|4|5|10-3|4|4|1|10-10-69|2
3|4|5|10-3|5|5|1|10-10-79|2
##END|INS_BLOCK
```

5.8.2 Configuration

This section describes the rule specific configuration entries for INFOINSERT. It does not include the standard entries supported by every Post Parser rule, detailed in the section: Post Parser Configuration.

The mandatory entries are:

- HEADER_NAMES_USED_TO_ID_DATA_RECORD: A list of header names that form the first part of a key that is used to identify which record of secondary information should be insert in this data record.

  ```
  HEADER_NAMES_USED_TO_ID_DATA_RECORD => [],
  ```

- NAMES_USED_TO_ID_DATA_RECORD: a list of counter names that are used to construct the second part of an identifier that is used to choose which record of secondary information should be included with each data record.

  ```
  NAMES_USED_TO_ID_DATA_RECORD        => ['OBJ_ID'],
  ```

- INFO_FILE_DESCRIPTION: a regular expression or a list of regular expressions describing the names of the secondary files that contain the information that is to be substituted into the data lines of the files that are described in the option INPUT_FILE_DESCRIPTION.

  ```
  INFO_FILE_DESCRIPTION               => ['info.pif'],
  ```

- HEADER_NAMES_USED_TO_ID_INFORMATION: a list of header counter names that form the first part of the key that is used to create the unique key to identify the secondary data for insertion.

  ```
  HEADER_NAMES_USED_TO_ID_INFORMATION => [],
  ```

- NAMES_USED_TO_ID_INFORMATION a list of counter names used to construct a unique identifier for the records of data in the secondary information file.

  ```
  NAMES_USED_TO_ID_INFORMATION        => ['INFO_KEY'],
  ```

The optional entries are:

- ONLY_INSERT: This list can be used to configure the list of counter names that are required for insertion, if the full set of data from the information file is not required.

```
ONLY_INSERT        => ['CELL', 'TRX'],
```

- WRITE_DATA_LINE: This option controls the action of this rule when there is no information to substitute for a line of data, i.e. the key from the main file is not found in the secondary file. If set to “True” and there is no data to substitute, the data row will be output anyway, with NULL values inserted for the secondary keys. If set to false the data row will not be outputted.
- OUTPUT_BLOCK_NAME: The name that should be used for the section name in the loader file.
- OUTPUT_FILENAME_START: A prefix that the output file name will start with.
- REDUNDANT_DATA_COUNTERS: A list of counters that should be deleted from the resultant output
- INFO_FILES_STORAGE_DIR: An optional scalar entry containing a directory name where information files can be stored. Information files not stored are deleted, as will be the case if INFO_FILES_STORAGE_DIR is not set in the INFOINSERT configuration or is set to zero (in non-debug mode).
- REMOVE_INFO_FILES: By default the information files are kept for the run. In certain situations, where the information files are being created for each Gateway iteration, this is not necessary. If this option is set to true the information files will be deleted.

The configuration used to produce the sample application:

```
{
      RULE_TYPE => 'INFOINSERT',
      RULE_DESC => 'INFOINSERT sample usage',
      INPUT_FILE_DESCRIPTION              => ['main_file.pif'],
      HEADER_NAMES_USED_TO_ID_DATA_RECORD => [],
      NAMES_USED_TO_ID_DATA_RECORD        => ['OBJ_ID'],
      INFO_FILE_DESCRIPTION               => ['info.pif'],
      HEADER_NAMES_USED_TO_ID_INFORMATION => [],
      NAMES_USED_TO_ID_INFORMATION        => ['INFO_KEY'],
      OUTPUT_BLOCK_NAME                   => 'INS_BLOCK',
      OUTPUT_FILENAME_START               => 'II',
      REDUNDANT_DATA_COUNTERS => [],
      ONLY_INSERT        => ['CELL', 'TRX'],
      WRITE_DATA_LINE    => 'True',
```

```
    PRODUCE_PIF         => 'True',
    OUTPUT_FORMAT       => 'LIF_Writer',
}
```

5.9 JOIN

The JOIN rule is used to join rows from multiple files into a single file based on the following:

pattern matching in the file name.

- counter key matching within each file.

The JOIN rule produces one larger file, with a single data row containing the data rows from the individual files in the output.

5.9.1 Sample Application

The example requires a number of input files to show the full usage. Two input files are configured:

File “CELL-HO-#-20Mar2004-#-01:00-#-I.pif”:

```
##START|CELL_HO
CHO_3|CELL_ID|BTS_ID|CHO_1|CHO_2
3|10-10-1|10-1|1|2
3|10-10-2|10-2|1|2
3|10-10-3|10-0|1|2
3|10-10-4|10-1|1|2
3|10-10-5|10-2|1|2
3|10-10-6|10-0|1|2
##END|CELL_HO
```

and file “CELL-TRAFFIC-#-20Mar2004-#-01:00-#-I.pif”

```
##START|CELL_TRAFFIC
CELL_ID|CTRF_1|BTS_ID|CTRF_2|CTRF_3
10-10-1|32|10-1|91|118
10-10-2|5|10-2|123|290
10-10-3|35|10-0|193|7
10-10-4|90|10-1|158|82
10-10-5|38|10-2|47|50
10-10-6|25|10-0|78|103
##END|CELL_TRAFFIC
```

These 2 files are joined based on the date time pattern in the filename, and on the counter keys BTS_ID and CELL_ID.

This creates one output file “CELLDATA-#-1-#-J.pif” with the joined contents:

```
##START|CELLDATA
CHO_3|CELL_ID|CTRF_1|BTS_ID|CTRF_2|CHO_1|CTRF_3|CHO_2
3|10-10-6|25|10-0|78|1|103|2
3|10-10-3|35|10-0|193|1|7|2
3|10-10-4|90|10-1|158|1|82|2
3|10-10-1|32|10-1|91|1|118|2
3|10-10-5|38|10-2|47|1|50|2
3|10-10-2|5|10-2|123|1|290|2
##END|CELLDATA
```

The joined file contains the data rows from each PIF joined to create a single row containing both the handover and traffic counters for each CELL.

5.9.2 Configuration

This section describes the rule specific configuration entries for JOIN. It does not include the standard entries supported by every Post Parser rule, detailed in the section: Post Parser Configuration.

The mandatory entries are:

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

  ```
  COUNTERS_TO_JOIN_ON => [qw(CELL_ID BTS_ID) ],
  ```

- 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 remote 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. This 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.

The sample rule configuration:

```
{
      RULE_TYPE => 'JOIN',
      RULE_DESC => 'Example join of files ',
      INPUT_FILE_DESCRIPTION => ['^CELL-HO-#-(.*)-#-I.pif',
                                 '^CELL-TRAFFIC-#-(.*)-#-I.pif'],
      OUTPUT_BLOCK_NAME => 'CELLDATA',
      REDUNDANT_DATA_COUNTERS => [],
      PRODUCE_PIF => 'True',
      PRODUCE_LIF => 0,
      COUNTERS_TO_JOIN_ON => [qw(CELL_ID BTS_ID) ],
      HEADER_COUNTERS_TO_USE_IN_OUTPUT_FILENAME => [],
      HOURS_TO_WAIT_FOR_PARTNER_FILES => -1,
      OUTPUT_FILENAME_START => 'CELLDATA',
      REDUNDANT_HEADER_COUNTERS => [],
}
```

5.10 MERGE_RECORDS

The MERGE_RECORDS Post Parser rule merges PIF data records within one PIF file based on a counter key. As each row contains the same counter names, the rule can be configured either to insert all values of a counter in the output row, or just a single value.

5.10.1 Sample Application

Given the input PIF data:

```
##START|DATA_BLOCK
OBJ_KEY|C_3|PEAK_TYPE|PEAK|C_1|C10_Total|C_2
10-10-1|3|TRAF|t200|1|1100|2
10-10-2|3|TRAF|t300|1|1200|2
10-10-3|3|TRAF|t400|1|1300|2
10-10-4|3|TRAF|t500|1|1400|2
```

```
10-10-0|3|TRAF|t600|1|1000|2
10-10-1|3|CPU|p700|1|1100|2
10-10-2|3|CPU|p800|1|1200|2
10-10-3|3|CPU|p900|1|1300|2
10-10-4|3|CPU|p1000|1|1400|2
10-10-0|3|CPU|p1100|1|1000|2
##END|DATA_BLOCK
```

The PIF data rows are going to be joined on the counter OBJ_KEY. All counters being merged onto the same row have the same value except for PEAK, which needs to be preserved from each row. This produces the following output:

```
##START|NEW_BLOCK
OBJ_KEY|C_3|PEAK_TYPE|NUM_MERGED|TCPU_PEAK|TTRAF_PEAK|C_1|C10_Total|C_2
10-10-0|3|TRAF|2|p1100|t600|1|1000|2
10-10-1|3|TRAF|2|p700|t200|1|1100|2
10-10-2|3|TRAF|2|p800|t300|1|1200|2
10-10-3|3|TRAF|2|p900|t400|1|1300|2
10-10-4|3|TRAF|2|p1000|t500|1|1400|2
##END|NEW_BLOCK
```

- The counters C_1, C_2, C_3 and C10_Total are merged into a single value.
- A new counter NUM_MERGED is added which counts the number of rows merged to create the new line.
- The PEAK value from each row is preserved. The rule creates 2 new counters, TCPU_PEAK and TTRAF_PEAK, which have the counter value from each of the merged rows. The counter PEAK_TYPE has been as a grouping key to identify the different counter values from each merged row.

5.10.2 Configuration

This section describes the rule specific configuration entries for MERGE_RECORDS. It does not include the standard entries supported by every Post Parser rule, detailed in the section: Post Parser Configuration.

The mandatory entries are:

- COUNTERS_TO_SORT_ON: A list of counters for the merge key. All rows with the same key will be merged into one output row.

```
COUNTERS_TO_SORT_ON => ['OBJ_KEY'],
```

The optional configuration entries are:

- OUTPUT_BLOCK_NAME: The block name used in the output files.
- REDUNDANT_DATA_COUNTERS: A list of counters to remove from the output data.
- GROUP_KEY: A counter name whose values within a collective set of data to be merged can be regarded as a key, the values will be used to prefix the counter names of records to be merged. If this option is not configured then a dummy prefix is used (T0, T1 ..).

  ```
  GROUP_KEY => 'PEAK_TYPE',
  ```

- MANIP_ONLY: List of counter names whose values from all rows are to be preserved in the output data. If option is not configured then all counter names other than those listed in option COUNTERS_TO_SORT_ON and REDUNDANT_DATA_COUNTERS (if configured) will be manipulated.

  ```
  MANIP_ONLY => ['PEAK']
  ```

The configuration used for the sample application of the rule:

```
{
      RULE_TYPE => 'MERGE_RECORDS',
      RULE_DESC => 'Merge Records test',
      INPUT_FILE_DESCRIPTION => ['^.*.pif'],
      OUTPUT_BLOCK_NAME => 'NEW_BLOCK',
      REDUNDANT_DATA_COUNTERS => [],
      PRODUCE_PIF => 'True',
      PRODUCE_LIF => 0,
      GROUP_KEY => 'PEAK_TYPE',
      COUNTERS_TO_SORT_ON => ['OBJ_KEY'],
      MANIP_ONLY => ['PEAK'],
}
```

5.11 PERLIZE

The PERLIZE Post Parser rule, unlike other rules, has complete flexibility as to how it is used. The functions for manipulating the header or counter data are configured in the Post Parser configuration file. The rule passes out each row of header and/or counter data to separate functions in the Post Parser configuration.

Here the data, both header and counter rows, can be manipulated in many ways including:

- deriving new counters from existing values, including using mathematical operations.
- adding new counters.

- renaming counters.
- removing counters.
- filtering files via the header fields, and counter rows individually.

5.11.1 Sample Application

Given the input data:

```
##START|HEADER
HC_1|HC_2|HC_3|DATETIME
0|1|15|20Mar2004_12:00
##END|HEADER
##START|DATA_BLOCK
C_3|C_1|C_2
19|4|6
14|3|19
22|0|3
25|5|16
0|3|10
##END|DATA_BLOCK
```

The PERLIZE rule is configured to call out to the following 2 sub-routines:

1. one to manipulate the header data, and create 2 new counters, DATE and TIME, from the current DATETIME counter.
2. one to manipulate the counter data, and create a new counter, C_ACCUM, which is the sum of all counter values on each row.

This produces the following output:

```
##START|HEADER
DATE|HC_1|HC_2|HC_3|TIME
20Mar2004|0|1|15|12:00
##END|HEADER
##START|DATA_BLOCK
C_ACCUM|C_3|C_1|C_2
29|19|4|6
36|14|3|19
25|22|0|3
46|25|5|16
```

```
13|0|3|10
##END|DATA_BLOCK
```

5.11.2 Configuration

This section describes the rule specific configuration entries for PERLIZE. It does not include the standard entries supported by every Post Parser rule, detailed in the section:Post Parser Configuration.

The mandatory configuration entries are:

- FILENAME_SUFFIX: A suffix string to append to the output file. This should reflect the operation performed by PERLIZE.

The optional configuration entries are:

- OUTPUT_BLOCK_NAME: The block name to use in the output data.
- OUTPUT_DIR: An alternative directory to write out the performance LIF data to.
- REDUNDANT_HEADER_COUNTERS: A list of header counters to delete from the output data.
- REDUNDANT_DATA_COUNTERS: A list of data counters to delete from the output data.
- HEADER_COUNTERS_OP: A subroutine to process the header block. This subroutine will be passed a reference to a hash containing the header names and values. If the subroutine returns non 0 then the file is discarded. The example below is creating separate date/time counters from one combined header counter, DATETIME:

```
HEADER_COUNTERS_OP => sub
      {
           my $h_ref = shift;
           # split date/time into 2 separate counters
           my @data = split("_",$h_ref->{DATETIME});
           $h_ref->{"DATE"} = $data[0];
           $h_ref->{"TIME"} = $data[1];
           # keep PIF
           return 0;
      },
```

- DATA_COUNTERS_OP: Same as above except each PIF data row will be passed out as a hash reference. If the subroutine returns non 0, then the row will be discarded. In the example below the total of all current counters is being output in a new counter C_ACCUM.

```
DATA_COUNTERS_OP => sub
      {
           my $c_ref = shift;
           # create a new counter
           my $accum = 0;
           foreach (values %$c_ref){
               $accum += $_;
           }
```

```
            $c_ref->{C_ACCUM} = $accum;
            # keep row
            return 0;
        },
```

This configuration was used for the sample application of the rule:

```
{
      RULE_TYPE => 'PERLIZE',
      RULE_DESC => 'Show PERLIZE usage',
      INPUT_FILE_DESCRIPTION => '.*\.pif',
      HEADER_COUNTERS_OP => sub
      {
           my $h_ref = shift;
           # split date/time into 2 separate counters
           my @data = split("_",$h_ref->{DATETIME});
           $h_ref->{"DATE"} = $data[0];
           $h_ref->{"TIME"} = $data[1];
           # all okay
           return 0;
      },
      DATA_COUNTERS_OP => sub
      {
           my $c_ref = shift;
           # create a new counter
           my $accum = 0;
           foreach (values %$c_ref){
               $accum += $_;
           }
           $c_ref->{C_ACCUM} = $accum;
           # all okay
           return 0;
      },
      FILENAME_SUFFIX => 'PZ',
      REDUNDANT_DATA_COUNTERS => [],
        REDUNDANT_HEADER_COUNTERS => ['DATETIME'],
}
```

5.12 PIF_2_OUTPUT

The PIF_2_OUTPUT Post Parser rule converts PIF based data to the final output format. It has 2 principal functions:

1. It allows the configuration of any output format that supports the LIF_Writer interface. The Gateway Framework contains as standard CSV_Writer and XML_Writer modules, as output formats.
2. Header and data counters can be output in the data in a sorted order. This is useful during development and installation, to track the values of specific counter names in the output data.

5.12.1 Sample Application

Given the input PIF:

```
##START|HEADER
DATE|STARTTIME|BSC|ENDTIME|H_1|H_2|H_3
20Mar2004|12:00|10-20-30|12:15|2|4|6
##END|HEADER
##START|BLOCK
ID_1|ID_2|C_3|C_4|C_1|C_2
1|3|30|70|10|30
1|5|10|40|20|30
2|3|30|50|10|30
2|5|40|50|20|10
##END|BLOCK
```

The PIF_2_OUTPUT rule is run on this data to:

- output the data using the LIF_Writer, hence the output will be LIF format.
- output the header counters in BSC, DATE, STARTTIME order. Other counters not in this list will be sorted lexically.
- Delete the header counter H_3
- Delete the data counter C_1

This produces the following LIF (partial output shown):

```
#%npr
#
{
  {
    BSC 10-20-30
    DATE 20Mar2004
```

```
STARTTIME 12:00
ENDTIME 12:15
H_1 2
H_2 4
OUT_BLOCK {
  ID_1-ID_2 1-3
  C_3 30
  C_4 70
  C_2 30
  ID_1 1
  ID_2 3
}
OUT_BLOCK {
  ID_1-ID_2 1-5
  C_3 10
  C_4 40
  C_2 30
  ID_1 2
  ID_2 4
}
```

5.12.2 Configuration

This section describes the rule specific configuration entries for PIF_2_OUTPUT. It does not include the standard entries supported by every Post Parser rule, detailed in the section: Post Parser Configuration.

There are no mandatory entries other than those that apply to all Post Parser rules.

The optional entries are:

- OUTPUT_FORMAT: The module to use to write out the performance data. This can be set to LIF_Writer, XML_Writer, CSV_Writer or Prospect_Writer.

  ```
  OUTPUT_FORMAT => 'LIF_Writer'
  ```

- REDUNDANT_HEADER_COUNTERS: A list of header counters to remove from the output data.
- REDUNDANT_DATA_COUNTERS: A list of data counters to remove from the output data.
- OUTPUT_BLOCK_NAME: The block name to use in the output data.

- OUTPUT_FILENAME_START: A string to prepend to the output filename.
- OUTPUT_DIR: The directory to output the performance data to, if it is not to be written to the output directory configured for the Gateway.
- NEW_HEADER_COUNTERS: A hash, containing a set of names and values of new counters to be output in the header.
- HEADER_COUNTER_ORDER: The order to output the header counters in. If not set no sorting is performed. Only list the counters that are required to be sorted and appear at the start of the output block. All other counters in the output data will be sorted lexically.

```
HEADER_COUNTERS_ORDER => [ qw(BSC DATE STARTTIME) ],
```

- DATA_COUNTERS_ORDER: Similar to above, the order to output the data counters in.

```
DATA_COUNTERS_ORDER =>
  {
    OUT_BLOCK => [ qw (C_3 C_4 C_2) ],
  },
```

- OUTPUT_RECORD_KEY: A list of data counters to form the record key as the first column of the output block records.

```
OUTPUT_RECORD_KEY => [ qw (ID_1 ID_2) ],
```

- OUTPUT_RECORD_KEY_DELIMITER: A string to separate the counter names for the output record key. If not set, the default '-#-' is used.

```
OUTPUT_RECORD_KEY_DELIMITER => '-',
```

The configuration used to produce the sample application of the rule:

```
{
      RULE_TYPE => 'PIF_2_OUTPUT',
      RULE_DESC => 'Test PIF 2 output rule',
      INPUT_FILE_DESCRIPTION => '.*',
      OUTPUT_BLOCK_NAME => 'OUT_BLOCK',
      OUTPUT_FILENAME_START => 'P2O',
      REDUNDANT_HEADER_COUNTERS => [qw(H_3)],
      REDUNDANT_DATA_COUNTERS => [qw(C_1)],
      OUTPUT_FORMAT => 'LIF_Writer',
      HEADER_COUNTERS_ORDER => [ qw(BSC DATE STARTTIME) ],
      DATA_COUNTERS_ORDER =>
       {
         OUT_BLOCK => [ qw (C_3 C_4 C_2) ],
       },
```

```
      OUTPUT_RECORD_KEY => [ qw (ID_1 ID_2) ],
      OUTPUT_RECORD_KEY_DELIMITER => '-',
}
```

5.13 PIF_REMOVE

The PIF_REMOVE rule removes PIF objects during the Post Parser stage. This can be useful to:

- reduce the complexity of configuring subsequent rules.
- reduce the memory/disk space requirements by freeing up space before further Post Parser rules run.

5.13.1 Sample Application

This section is not applicable. All files that match the INPUT_FILE_DESCRIPTION of the rule will be deleted.

5.13.2 Configuration

There are no non-standard entries for this Post Parser rule. In the example configuration below all BSC files with the "A" suffix are being deleted.

```
{
      RULE_TYPE => 'PIF_REMOVE',
      RULE_DESC => 'Remove all accumulated BSC files',
      INPUT_FILE_DESCRIPTION => ['^BSC.*-#-A\.pif'],
}
```

5.14 SPLIT_RECORDS

The SPLIT_RECORDS rule splits a single PIF data row into multiple rows, based on different counter names. Counters that are being used to split the data will have a new value in each new data rows.

5.14.1 Sample Application

Given the input PIF data:

```
##START|BLOCK
C_3|C_1|SV_KEY|CV_KEY|C_2
31|10|SV_2|CV_0|13
14|10|SV_18|CV_11|12
23|10|SV_22|CV_1|11
##END|BLOCK
```

The rule will perform the following:

- split each row into 2 rows, using the keys SV_KEY and CV_KEY. The type used to split the row will be placed in a new counter COUNTER_ID.
- The value that was contained in SV_KEY or CV_KEY will be output in a new counter KEY.
- The counters C_2 and C_3 will be written out in each data row.

The output PIF produced by the rule:

```
##START|SPLIT_BLOCK
C_3|COUNTER_ID|KEY|C_2
31|T_SV|SV_2|13
31|T_CV|CV_0|13
14|T_SV|SV_18|12
14|T_CV|CV_11|12
23|T_SV|SV_22|11
23|T_CV|CV_1|11
##END|SPLIT_BLOCK
```

5.14.2 Configuration

This section describes the rule specific configuration entries for SPLIT_RECORDS. It does not include the standard entries supported by every Post Parser rule, detailed in the section: Post Parser Configuration.

The mandatory configuration entries are:

- SPLIT_CNAMES: The key of the hash will be a unique string that will identify the new split record. The new split record will report values for the counter names list in its array. The size of the array can be variable, but should not exceed the number of elements configured for the array NEW_CNAMES. The counter names listed in different arrays do not need to be mutually exclusive. Unless the options 'ONLY_INSERT' or 'REDUNDANT_DATA_COUNTERS' are configured, any counter names that are not listed in the arrays are automatically reported in each of the split records.

```
SPLIT_CNAMES => {
            'T_CV' => ['CV_KEY'],
            'T_SV' => ['SV_KEY'],
},
```

- NEW_CNAMES: An array of new counter names that will apply to the counters that have been split in the record.

```
NEW_CNAMES => ['KEY'],
```

The optional configuration entries are:

- OUTPUT_BLOCK_NAME: A replacement data block name used in the output files

- OUTPUT_DIR: Full path to an alternative directory where the output files will be written.
- ONLY_INSERT: A list of other counter names to be reported in every record. If not set, nor REDUNDANT_DATA_COUNTERS is set, no such counter names values will be inserted.

```
ONLY_INSERT => ['C_3', 'C_2'],
```

- REDUNDANT_DATA_COUNTERS: A list of counters that should be deleted from the resultant output. If set then all other counter names reported and not specified in SPLIT_CNAMES will be reported. If both REDUNDANT_DATA_COUNTERS and ONLY_INSERT, ONLY_INSERT takes precedence. If neither is set no counter names will be inserted.
- NEW_REC_ID_NAME: Defines the new counter name in the block, it reports as its values the keys of the hash 'SPLIT_CNAMES'.

```
NEW_REC_ID_NAME => 'COUNTER_ID',
```

The configuration for the sample application of the rule is below:

```
{
      RULE_TYPE => 'SPLIT_RECORDS',
      RULE_DESC => 'Split records test',
      INPUT_FILE_DESCRIPTION => ['^.*.pif'],
      SPLIT_CNAMES => {
                  'T_CV' => ['CV_KEY'],
                  'T_SV' => ['SV_KEY'],
      },
      NEW_CNAMES => ['KEY'],
      NEW_REC_ID_NAME => 'COUNTER_ID',
      OUTPUT_BLOCK_NAME => 'SPLIT_BLOCK',
      ONLY_INSERT => ['C_3', 'C_2'],
      PRODUCE_PIF => 'True',
}
```

5.15 UNPEGGER

The UNPEGGER Post Parser rule is used to derive unpegged values for pegged counter types.

Pegged counters are defined as those whose value constantly increases up to a set value, until it rolls over back to 0. A single value of a pegged counter contains no information on system performance. At least 2 values are required to derive the change in value over a period.

For example:

At 15:00 counter pdpContexts had a pegged value of 13456.

At 15:15 counter pdpContexts had a pegged value of 14456.

At 15:30 counter pdpContexts had a pegged value of 20000.

Therefore the unpegged value for this period at 15:15 is 1000, and at 15:30 the unpegged value is 5544.

These type of counters are usually found in IP based networks, and are either 32 or 64 bit in size.

5.15.1 Sample Application

The application of this rule is quite complicated. The example described here only gives the simplest application of the rule's usage.

Given 2 input files "GGSN-#-GGSN-R1D9-#-02Mar2004-#-14:15-#-I.pif"

```
##  Parser Intermediate File
##START|HEADER
H_4|STARTTIME|DATE|ManagedElement|H_1|H_2|DURATION|H_3
4|14:15|02Mar2004|GGSN-R1D9|1|2|15|3
##END|HEADER
##START|GGSN_BLOCK
GgsnFunction|VS.IPSec.IncDataOct|VS.IPSec.DiscDataPkt|GiIsp|VS.IPSec.IncDataPk
t|VS.IPSec.OutDataOct
0|961|2034|4|9995|5305
0|14312|1621|7|14441|2846
0|1455|7127|12|6533|4730
0|12076|4191|19|5113|14295
0|11901|9860|28|1607|3949
##END|GGSN_BLOCK
```

and file "GGSN-#-GGSN-R1D9-#-02Mar2004-#-14:30-#-I.pif":

```
##  Parser Intermediate File
##START|HEADER
H_4|STARTTIME|DATE|ManagedElement|H_1|DURATION|H_2|H_3
4|14:30|02Mar2004|GGSN-R1D9|1|15|2|3
##END|HEADER
##START|GGSN_BLOCK
```

```
GgsnFunction|VS.IPSec.IncDataOct|VS.IPSec.DiscDataPkt|GiIsp|VS.IPSec.IncDataPk
t|VS.IPSec.OutDataOct
0|18284|29817|4|29650|25838
0|15924|24286|7|28204|18781
0|24969|27115|12|28747|28223
0|17761|23425|19|28908|15957
0|21861|19129|28|17700|17415
##END|GGSN_BLOCK
```

The following processing is performed by the rule:

- Each GGSN pattern in the file name, “GGSN-#-GGSN-R1D9” is processed as a group.
- Files with the GGSN pattern are then processed within the date time key, in this case 02Mar2004-#-14:15 followed by 02Mar2004-#-14:30.
- Since the file 14:15 has no previous file no unpegged output is produced, the file is saved for the next iteration.
- The counter values in the 14:30 file are unpegged against the previous file 14:15.
- The DATE, STARTTIME and DURATION counters in the header are used to derive the time difference between the files.
- The counters GgsnFunction and GiIsp are used as data row keys to match the related data rows between each PIF file.
- The VS.IPSec* counters are being unpegged. They are all 32 bit counters.
- The output PIF file will be prefixed with “UNPEG”.

This produces the output file:

“UNPEG-#-GGSN-#-GGSN-R1D9-#-02Mar2004-#-14:30-#-I.pif”:

```
##  Parser Intermediate File
##START|HEADER
H_4|STARTTIME|DATE|ManagedElement|H_1|DURATION|H_2|H_3
4|14:30|02Mar2004|GGSN-R1D9|1|15|2|3
##END|HEADER
##START|GGSN_BLOCK
GgsnFunction|VS.IPSec.IncDataOct|VS.IPSec.DiscDataPkt|GiIsp|VS.IPSec.IncDataPk
t|VS.IPSec.OutDataOct
0|17323|29817|4|19655|25838
0|1612|24286|7|13763|18781
0|23514|27115|12|22214|28223
```

```
0|5685|23425|19|23795|15957
0|9960|19129|28|16093|17415
##END|GGSN_BLOCK
```

This is the unpegged data. For example in the first row (GgsnFunction=0, GiIsp=4), the difference in value for counter VS.IPSec.IncDataOct from 14:15 to 14:30 is

18284 (from 14:30 file) less 961 (from 14:15 file) = 17323

The rule can equally be configured to unpeg data for multiple date/time periods in a single PIF file.

The rule can also handle a backlog of files, as multiple files are sorted in date/time order and then processed in this order.

5.15.1.1 Time and Duration Formats

Unpegging of valid data must be based on valid date, time and duration calculations. The UNPEGGER supports a number of formats for the date and time in the pegged filename, as extracted using INPUT_FILE_DATETIME_KEY. These formats are:

<day><month><year>[DATE SEP]<hour>[HOUR SEP]<minute>

where:

- day – is a 2 digit value
- month – is a 2 digit value, or a 3 letter prefix e.g. Apr
- year – is a 4 digit value
- [DATE SEP] – is an optional date separator. Acceptable values for the separator are “-“, “_”, “.” and “-#-“.
- hour – the hour in 24 hourly format
- minute – is a 2 digit value.

Examples of valid datetime values are:

```
02Mar2004-#-11:00
02032004_11.00
02Mar20041100
```

5.15.2 Configuration

This section describes the rule specific configuration entries for UNPEGGER. It does not include the standard entries supported by every Post Parser rule, detailed in the section: Post Parser Configuration.

The mandatory configuration entries are:

- INPUT_FILE_DESCRIPTION: The list of pegged files to be processed by this rule. The element ID and key must be extracted in the configuration. This ensures that the correct

element is unpegged against its previous file. In the example below the element type, GGSN, along with the element ID is being used as the pattern to match.

```
INPUT_FILE_DESCRIPTION => '^(GGSN-#-\w+\-\w+).*\.pif',
```

- INPUT_FILE_DATETIME_KEY: A regular expression matching the input file name and extracting the date and time key from the filename. This is required to sort the input files in the correct order, so the oldest input file will be processed first.

```
INPUT_FILE_DATETIME_KEY =>
                '^.*-#-(\d{2}\w{3}\d{4}-#-\d{2}:\d{2})-#-I\.pif',
```

- INPUT_DATE_FORMAT: A string configuring the sequence of the date string matched in INPUT_FILE_DATETIME_KEY. The configuration can either be set to 'DMY', YMD' or 'MDY' depending on the order of the Day, Month, and Year in the matched string. If this configuration entry is not defined, the order of the date format used will be as in the order above.

```
INPUT_DATE_FORMAT    => 'YMD',
```

- PEG_FILENAME_PREFIX: The prefix used to generate the previous pegged PIF file. Once the current file has been unpegged, it must be saved as the previous file. The values in this file will be used as the previous values for the next iteration of the rule.

```
PEG_FILENAME_PREFIX => 'SAVED',
```

- UNPEG_FILE_TYPE: The type of pegged files that are being handled. This may be set to either:
- HEADER, where the datetime and duration fields are contained in the header, and apply to the whole file.
- DATA, where the datetime and duration fields are contained in the pif data rows, and apply individually to the pif data row.

```
UNPEG_FILE_TYPE => 'HEADER',
```

- KEEP_RAW_GRANULARITY: This option decides what happens when a PIF file is missing. A PIF file is determined as missing when the previous PIF endtime does not match the current PIF starttime. When this occurs a decision must be made on whether to output a LIF, with the values calculated with the available files. If KEEP_RAW_GRANULARITY is set to TRUE then a LIF file will be created, ignoring the missing PIF. If KEEP_RAW_GRANULARITY is set to 0, then no output LIF will be produced to preserve the expected raw performance data granularity.

```
KEEP_RAW_GRANULARITY   => 'TRUE',
```

- MAX_PEG_PIF_AGE: This formula is used to calculate the maximum time difference that is allowed between the previous timestamp and current timestamp before the previous is considered to have expired. This may be the date/time counters in either the header or counter data depending on the type of unpegging being done.

This formula should include the DURATION tag, as the max age should be calculated as a factor of this. Duration is calulated as the difference between the start and end time of the current PIF. For example if the starttime of the file is 13:15, the endtime is 13:30, the MAX_PEG_PIF_AGE will be 120 minutes (15*4+60). If the file has expired the previous PIF will be deleted and the current saved for the next run.

```
MAX_PEG_PIF_AGE         => 'DURATION*4+60',
```

- CALCULATE_ROLLOVER: The option decides whether or not the unpegged value is calculated when rollover occurs. Pegged counter values always increase until they reach their max size (2^32-1 for example), and are then rolled over to 0. If CALCULATE_ROLLOVER is set to true, the UNPEGGER rule will calculate the rollover value, based on the maximum configured in PEG_COUNTERS.

  If CALCULATE_ROLLOVER is set to 0, a NULL value will be inserted into the LIF for the counter value when rollover occurs.

```
CALCULATE_ROLLOVER      => 'TRUE',
```

- DEFAULT_NULL_VALUE: The value to use as the null value when required in the LIF output.

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

- DATETIME_COUNTERS: This hash supports three options used to calculate the time difference between current and previous peg files, and the measurement duration.

  The date/time counters can be in almost any format as they are parsed using a DateTime module, which is extremely flexible in it accepted format.

  INPUT_DATE_FORMAT is required for ambiguous date format in the data record, similar to the one for the file name date above.

  One of these combinations must exist in the PIF.

  1. Start date/time and end date/time.

     This hash maps the counters, STARTDATE, STARTTIME, ENDDATE and ENDTIME to the counter names for the start and end date/times in the PIF header.

```
DATETIME_COUNTERS       => {
    STARTDATE => 'StartDate',
    STARTTIME => 'StartTime',
    ENDDATE   => 'EndDate',
    ENDTIME   => 'EndTime',
    INPUT_DATE_FORMAT => 'YMD',
}
```

  2. Start date/time and duration.

This hash maps the counters STARTDATE, STARTTIME, and DURATION to the counter names for the start date, start time and duration in the PIF header.

The optional FORMAT entry determines how the DURATION value is interpreted. Valid values are 'seconds' and 'minutes'. Defaults to minutes.

```
DATETIME_COUNTERS      => {
    STARTDATE => 'StartDate',
    STARTTIME => 'StartTime',
    DURATION  => 'gp',
    FORMAT    => 'seconds',
    INPUT_DATE_FORMAT => 'YMD',
}
```

3. End date/time and duration.

   This hash maps the counters ENDDATE, ENDTIME, and DURATION to the counter names for the end date, end time and duration in the PIF header.

```
DATETIME_COUNTERS  => {
    ENDDATE   => 'EndDate',
    ENDTIME   => 'EndTime',
    DURATION  => 'gp',
    FORMAT    => 'seconds',
    INPUT_DATE_FORMAT => 'YMD',
}
```

- COUNTERS_TO_SORT_ON: The list of counters in the data rows used to map the row in the previous PIF to the current, when calculating the unpegged values:

```
COUNTERS_TO_SORT_ON => [ qw (GgsnFunction GiIsp) ],
```

- PEG_COUNTERS: A hash containing list of REs. Any counters in the input data which match the RE will be unpegged. The hash value specifies the rollover value for the counter.

  For 32 bit counters this should be set to 4294967295.

  For 64 bit counters this should be set to 18446744073709551615.

```
PEG_COUNTERS => {
             'VS\.IPSec.*' => '4294967295'
           },
```

The optional configuration entries are:

- REDUNDANT_HEADER_COUNTERS: The list of counters to remove from the header in the output data.
- REDUNDANT_DATA_COUNTERS: The list of counters to remove from the data rows of the output data.

The configuration sample for the worked example is below:

```
{
RULE_TYPE => 'UNPEGGER',
RULE_DESC => 'unpeg counter values for LIF output',
INPUT_FILE_DESCRIPTION => '^(GGSN-#-\w+\-\w+).*\.pif',
INPUT_FILE_DATETIME_KEY =>
'^.*-#-(\d{2}\w{3}\d{4}-#-\d{2}:\d{2})-#-I\.pif',
        PEG_FILENAME_PREFIX => 'SAVED',
        UNPEG_FILENAME_PREFIX => 'UNPEG',
        UNPEG_FILE_TYPE => 'HEADER',
        PRODUCE_PIF => 1,
        OUTPUT_FORMAT => 'LIF_Writer',
        KEEP_RAW_GRANULARITY => 'TRUE',
        MAX_PEG_PIF_AGE => 'DURATION*4+60',
        CALCULATE_ROLLOVER => 'TRUE',
        DEFAULT_NULL_VALUE => 'NULL',
DATETIME_COUNTERS => {
        STARTTIME => 'STARTTIME',
        STARTDATE => 'STARTDATE',
        DURATION => 'DURATION',
        },
COUNTERS_TO_SORT_ON => [ qw (GgsnFunction GiIsp) ],
REDUNDANT_HEADER_COUNTERS => [  ],
REDUNDANT_DATA_COUNTERS => [],
PEG_COUNTERS => {
'VS\.IPSec.*' => '4294967295'
        },
}
```

6 Performance Tips

The amount, size and complexity of data to be processed varies from vendor to vendor. Hence performance of different vendor Gateways can vary widely. Any productised Vendor Gateways are profiled for performance problems as part of the testing process. However, there can be issues with server installations and configurations that require performance tuning.

This section details some common changes that can improve performance of the Gateway. Before applying any changes, ensure the Gateway is outputting the correct LIF data for loading. It is also advisable to make a reference copy of the output so the resultant data from the any changes can be validated.

1. **Minimize the number of intermediate and output files:** Typically a Gateway will process less larger files with more data more efficiently than a greater number of smaller files with a smaller number of records. For example it is preferable to process a set of TRX counters in one large file per BSC, rather than a number of smaller files per CELL, even if it means that hierarchy data is duplicated. This is less of an issue if using PIF data is cached in memory. It may also have an impact on the loadmap configuration.

2. **Use BATCHFILES in preference to PIF_2_LIF:** The BATCHFILES rule can be used to create larger output PIF and LIF files, where multiple files matching the same pattern are joined into one larger file. This should be used in preference to the PIF_2_LIF rule as it produces less files, reducing I/O. It is also more efficient for subsequent loading.

3. **If outputting to multiple directories, use the Transfer Stage:** For parallel PM/NPR installations it is often required to output the performance data to 2 destinations. Rather than using multiple instances of the same rule to output to 2 directories, use the Transfer Engine or external scripts to distribute the files.

4. **Use tmpfs filesystems or cache PIF data in memory:** Performance data goes through a number of transformations before final output. Disk I/O can be a major bottleneck. Use either tmpfs filesystems or configure the PIF data to be cached in memory. This can offer significant savings.

5. **Use parallel processing:** If the server has more than one processor, configure parallel processing for rules which meet the guidelines detailed. This can also be beneficial if the disk is slow, as multiple reads/writes can be queued to the disk.

6. **Use PERLIZE for complex operations:** If there is a specific counter manipulation/calculation requirement which requires a number of transformations, use the PERLIZE rule to configure it in a single function, rather than write specific rule(s).

7. **Gently stop Gateway process:** If required, the execution chain can be stopped gently by creating an empty file named **`stop_gateway`** in the input directory. Gateway will stop the current engine stage (does not parse all remaining raw files) and proceed to post parsing stage. The remaining raw files will be parsed when the Gateway is restarted.

Index

Appendix A Notices and Trademarks

This appendix contains the following:

- Notices
- Trademarks

6.1.1.2 Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome
Minato-ku
Tokyo 106-0032
Japan.

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
5300 Cork Airport Business Park
Kinsale Road
Cork
Ireland.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

6.1.1.3 Trademarks

IBM, IBM logo, Tivoli, and Metrica are trademarks of International Business Machines Corporation in the United States, other countries or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Intel and Pentium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

Other company, product or service names may be trademarks or service marks of others.

International Business Machines Corporation
5300 Cork Airport
Business Park
Kinsale Road
Cork
Ireland

Printed in the Republic of Ireland