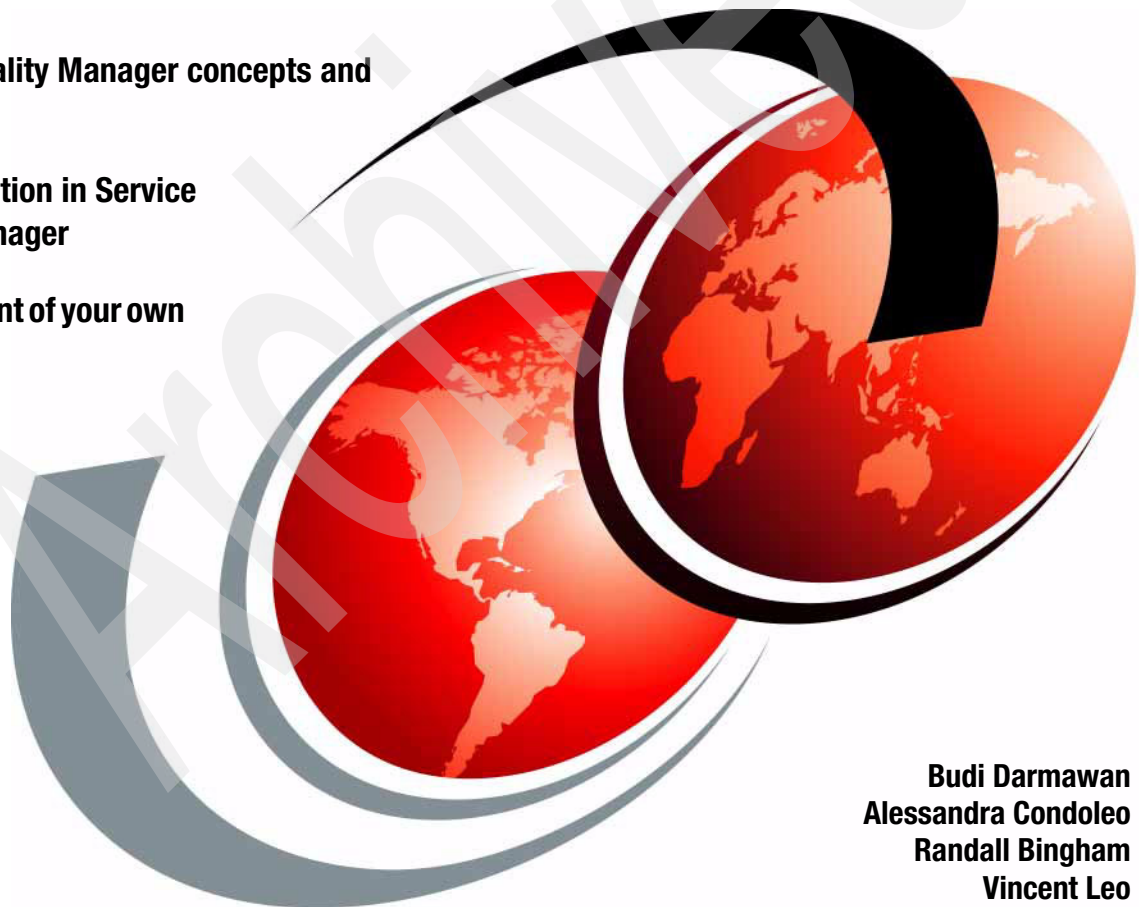


# IBM Tivoli Netcool Service Quality Manager Data Mediation Gateway Development

Service Quality Manager concepts and  
overview

Data mediation in Service  
Quality Manager

Development of your own  
interface



Budi Darmawan  
Alessandra Condoleo  
Randall Bingham  
Vincent Leo





International Technical Support Organization

**IBM Tivoli Netcool Service Quality Manager Data  
Mediation Gateway Development**

March 2009

Archived

**Note:** Before using this information and the product it supports, read the information in “Notices” on page vii.

### **First Edition (March 2009)**

This edition applies to Version 4, Release 1 of IBM Tivoli Netcool Service Quality Manager.

© Copyright International Business Machines Corporation 2009. All rights reserved.

Note to U.S. Government Users Restricted Rights -- Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

# Contents

<b>Notices</b> .....	vii
Trademarks .....	viii
<b>Preface</b> .....	ix
The team that wrote this book .....	ix
Become a published author .....	x
Comments welcome .....	xi
<b>Chapter 1. IBM Tivoli Netcool Service Quality Manager</b> .....	1
1.1 Service quality management .....	2
1.2 Product architecture .....	4
1.3 Service Quality Manager server .....	5
1.4 Data mediation subsystem .....	6
1.4.1 Extract Transform Load process .....	7
1.4.2 Gateway architecture .....	10
1.4.3 PIF file format .....	12
<b>Chapter 2. Gateway concepts and processing</b> .....	13
2.1 Gateway architecture .....	14
2.1.1 Gateway components .....	14
2.1.2 Gateway operation .....	16
2.2 Perl library extension and engine configuration .....	19
2.3 Working with transfer configuration rules .....	20
2.3.1 FTP transfer .....	20
2.3.2 SCP/RCP transfer .....	21
2.4 Working with post parser rules .....	23
2.4.1 Common parameters .....	23
2.4.2 ACCUMULATE .....	24
2.4.3 AGGREGATE_LINE .....	25
2.4.4 CVAL_MANIP .....	26
2.4.5 DATALINE_WHERE .....	28
2.4.6 FILE_SPLIT .....	29
2.4.7 FILE_SPLIT_BY_COUNTERS .....	30
2.4.8 INFOINSERT .....	31
2.4.9 JOIN .....	34
2.4.10 MERGE_RECORDS .....	35
2.4.11 PERLIZE .....	36
2.4.12 PIF_2_CSV .....	37
2.4.13 PIF_2_OUTPUT .....	38

2.4.14 SPLIT_RECORD .....	38
2.4.15 UNPEPPER .....	40
2.5 Performance tips .....	41
<b>Chapter 3. Gateway development</b> .....	<b>43</b>
3.1 Identifying the data source .....	44
3.1.1 Input specification for GPRS Technology Pack .....	44
3.1.2 GPRS Service Solution interface control guide .....	47
3.1.3 Mapping of data from gateway and adapters .....	50
3.2 Gateway configuration for Motorola GGSN .....	51
3.2.1 Configuring the Gateway directory settings .....	51
3.2.2 Configuring auditing and logging for the gateway .....	52
3.2.3 Enabling debug mode settings for the gateway .....	52
3.3 Transfer process for the Motorola GGSN gateway .....	53
3.3.1 Extracting data from source .....	53
3.3.2 Sending data to the adapter .....	55
3.4 Modifying the Generic CSV parser engine .....	56
3.4.1 Verifying the parser engine configuration .....	56
3.4.2 Modifying the configuration .....	56
3.5 Developing the post parser configuration for Motorola GGSN .....	58
3.5.1 Manipulating the data in the PERLIZE post parser rule .....	58
3.5.2 Outputting data in the CSV format using the PIF_2_CSV rule .....	63
3.6 Running gateway programs .....	64
3.6.1 Transferring file inbound .....	65
3.6.2 Processing the CSV file .....	67
3.6.3 Running the PERLIZE stage .....	69
3.6.4 Running the PIF_2_CSV stage .....	71
3.6.5 Transferring file outbound .....	72
<b>Appendix A. Sample data and programs listing for gateways</b> .....	<b>75</b>
A sample Motorola GGSN CSV file .....	76
Sample first level PIF data file .....	76
Sample output CSV file .....	78
CSV_Writer_ITSO.pm .....	78
Motorola GGSN EngineConfig.pm .....	85
Motorola GGSN UserConfig.pm .....	88
Sample UserConfig.pm .....	92
<b>Appendix B. Additional material</b> .....	<b>95</b>
Locating the Web material .....	95
Using the Web material .....	96
System requirements for downloading the Web material .....	96
How to use the Web material .....	96

**Abbreviations and acronyms** ..... 97

**Related publications** ..... 99

IBM Redbooks ..... 99

Other publications ..... 99

Online resources ..... 100

How to get Redbooks ..... 100

Help from IBM ..... 100

**Index** ..... 101



# 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 give 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.*

**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.

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.

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.

## COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

# Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. These and other IBM trademarked terms are marked on their first occurrence in this information with the appropriate symbol (® or ™), indicating US registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. A current list of IBM trademarks is available on the Web at <http://www.ibm.com/legal/copytrade.shtml>


The following terms are trademarks of the International Business Machines Corporation in the United States, other countries, or both:

AIX®

IBM®

Netcool®

Redbooks®

Redbooks (logo) ®

ServiceAssure™

Tivoli®

The following terms are trademarks of other companies:

ITIL is a registered trademark, and a registered community trademark of the Office of Government Commerce, and is registered in the U.S. Patent and Trademark Office.

Oracle, JD Edwards, PeopleSoft, Siebel, and TopLink are registered trademarks of Oracle Corporation and/or its affiliates.

Java, Solaris, Sun, and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Expression, MS, and the Windows logo are trademarks of Microsoft 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.

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

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

# Preface

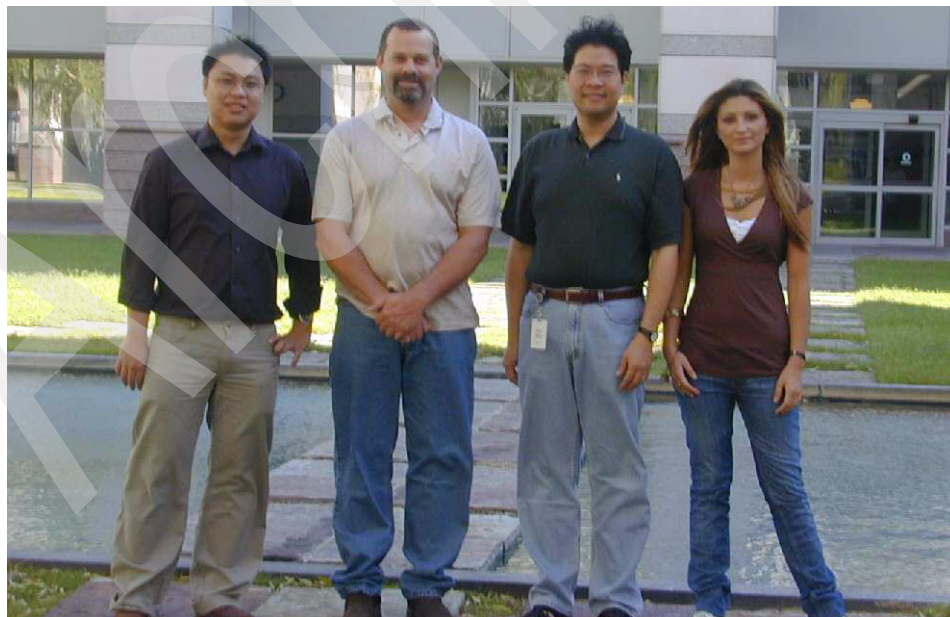
This IBM® Redbooks® publication discusses IBM Tivoli® Netcool® Service Quality Manager V4.1.1 data mediation development. It is aimed at implementers and system integrators connecting new data sources that will be managed by Service Quality Manager.

Data mediation in Service Quality Manager is performed by Perl gateways. Gateways extract data from vendor specific hardware monitors and prepare the data to be loaded into the Service Quality Manager application.

The development aspect of data mediation for gateways uses Perl extensively. The control files or rules are written as Perl subroutines. We discuss the detailed specifications and processing of these mediation rules.

## The team that wrote this book

This book was produced by a team of specialists from around the world working at the International Technical Support Organization, Austin Center.



*Figure 1 Vincent, Randy, Budi, and Alessandra*

**Budi Darmawan** is a project leader at the International Technical Support Organization, Austin Center. He writes extensively and teaches IBM classes worldwide on all areas of systems and application management. Before joining the ITSO 10 years ago, Budi worked in IBM Indonesia as a lead implementer and solution architect. His current interests are business service management, application management, and Java™ programming.

**Alessandra Condoleo** is a solution designer for GTS Italy, working in the SPL2 team designing and implementing IT Lifecycle Management and Governance Service solutions. She has been working at IBM for two years and is currently in a Master's degree program for Business Administration. She developed the ROI SOA investment calculation tool. Alessandra is a Netcool Trained practitioner and is ITIL® Foundation Certified. Before working at IBM two years ago, Alessandra worked at the University of Pisa, Italy, as a programmer. She received the "Laurea" (M.S.) degree in Biomedical Engineering (summa cum laude) from the University of Pisa.

**Randall Bingham** is a Network Performance Engineer in the USA. He has 20 years of experience in the telecommunications industry. He has worked at IBM for 4 years. His areas of expertise include Network Management, Performance Management, and Fault Management.

**Vincent Leo** is a Level 2 Support Engineer in Malaysia. He has 8 years of experience in application support and performance management for the telecommunication field, and has been with IBM for 3 years. He is CCNA, MCSE, and RHCT certified, and holds a degree in Mechanical Engineering from the University of Leicester. His other areas of expertise include storage systems and embedded Linux®.

Thanks to the following people for their contributions to this project:

Wade Wallace  
International Technical Support Organization, Austin Center

Ann Hayes, Ken Tyler, Mohd Ezman Zainudin, Sanjay Nayak, Thierry Supplisson,  
Barra Seck,  
IBM Software Group, Tivoli Service Quality Manager team

## Become a published author

Join us for a two- to six-week residency program! Help write a book dealing with specific products or solutions, while getting hands-on experience with leading-edge technologies. You will have the opportunity to team with IBM technical professionals, Business Partners, and Clients.

Your efforts will help increase product acceptance and customer satisfaction. As a bonus, you will develop a network of contacts in IBM development labs, and increase your productivity and marketability.

Find out more about the residency program, browse the residency index, and apply online at:

[ibm.com/redbooks/residencies.html](http://ibm.com/redbooks/residencies.html)

## Comments welcome

Your comments are important to us!

We want our books to be as helpful as possible. Send us your comments about this book or other IBM Redbooks in one of the following ways:

- ▶ Use the online **Contact us** review Redbooks form found at:

[ibm.com/redbooks](http://ibm.com/redbooks)

- ▶ Send your comments in an e-mail to:

[redbooks@us.ibm.com](mailto:redbooks@us.ibm.com)

- ▶ Mail your comments to:

IBM Corporation, International Technical Support Organization  
Dept. HYTD Mail Station P099  
2455 South Road  
Poughkeepsie, NY 12601-5400



# IBM Tivoli Netcool Service Quality Manager

IBM Tivoli Netcool Service Quality Manager manages the system's commitment to deliver high quality service for an IT service provider. It originated in quality management for the wireless telecommunication industry. This book focuses on data mediation used to collect data from vendor devices using a Perl gateway. In this chapter, we introduce Service Quality Manager concepts and give an overview of Service Quality Manager. The chapter contains the following sections:

- ▶ 1.1, "Service quality management" on page 2
- ▶ 1.2, "Product architecture" on page 4
- ▶ 1.3, "Service Quality Manager server" on page 5
- ▶ 1.4, "Data mediation subsystem" on page 6

## 1.1 Service quality management

As in every industry, the telecommunications industry is challenged to do more with less, many in the industry are searching more aggressively for leading edge management systems to automate day to day collection and management of network performance and necessary customer management data.

Service providers are pressured to seek growth and control costs, while their infrastructure is getting more complex with the addition of multiple technologies and vendors to deliver their services. Other market conditions, such as increased competition and demanding customer expectations, also contributes to the issue.

The key to addressing these challenges is understanding the customer experience. Insights into the real customer experience can help you keep your customers, create and launch new services, and optimize existing services.

IBM Tivoli Netcool Service Quality Manager enables you to see your service through the customer's eyes and thus helps you deliver high-quality service. With an end-to-end view, you can verify that each service (including voice, video, e-mail, Voice over IP (VoIP), i-mode, online gaming, virtual private network (VPN), IP-TV, DSL, Metro-Ethernet and more) is functioning correctly and within the expected bounds.

To help you manage services for each subscriber group in the network, Tivoli Netcool Service Quality Manager takes a technology-agnostic approach so that you can manage service quality and the customer experience through a wide variety of delivery technologies. Additionally, it features quick-to-deploy solutions for managing services such as Global System for Mobile (GSM) communications, General Packet Radio Services (GPRS), third- generation (3G) wireless technologies, Digital Subscriber Line (DSL), and WiMAX/WiFi networks.

Tivoli Netcool Service Quality Manager brings together an historical view of service quality management (SQM) and service level agreement (SLA) management into one solution, designed specifically to manage and improve service quality despite the complexities of today's infrastructure. This solution includes customer experience management capabilities that is designed to address the full spectrum of quality from the network, service, and customer perspectives.

Tivoli Netcool Service Quality Manager also includes an optional component that focuses specifically on customer experience management (CEM). CEM provides a detailed analysis of an individual subscriber 's experience. It examines subscriber transactions by service, location, subscriber group, and device type, then supplies this critical customer experience information to teams such as customer care, marketing, network operations, and executives.

By integrating these diverse efforts, Tivoli Netcool Service Quality Manager helps you operate efficiently and achieve a sharpened focus on customer service quality, in the context of your overall business management priorities.

IBM Tivoli Netcool Service Quality Manager provides a real-time, end-to-end view of a service to enable Service Providers to understand service quality from the customer's perspective. Service Quality Manager has the following major features:

- ▶ Monitors and improves the quality of each customer experience, resulting in more effective customer care and increased customer satisfaction.
- ▶ Responds to network issues based on corporate directives such as revenue, profitability, service, and customer impact.
- ▶ Provides product differentiation to your enterprise sales team by offering guaranteed service level agreements (SLAs) to attract and retain high-value enterprise customers.
- ▶ Enables the successful, rapid introduction of new services that you can offer with confidence in their service quality.
- ▶ Identifies high-priority problems quickly and accurately with powerful root-cause and impact analysis.
- ▶ Offers comprehensive data collection capabilities and extensive pre-established service models with full key quality indicator (KQI) and key performance indicator (KPI) mapping and reporting.

## 1.2 Product architecture

IBM Tivoli Netcool Service Quality Manager was formerly called ServiceAssure™ from Vallent Technology. It has the components shown in Figure 1-1.

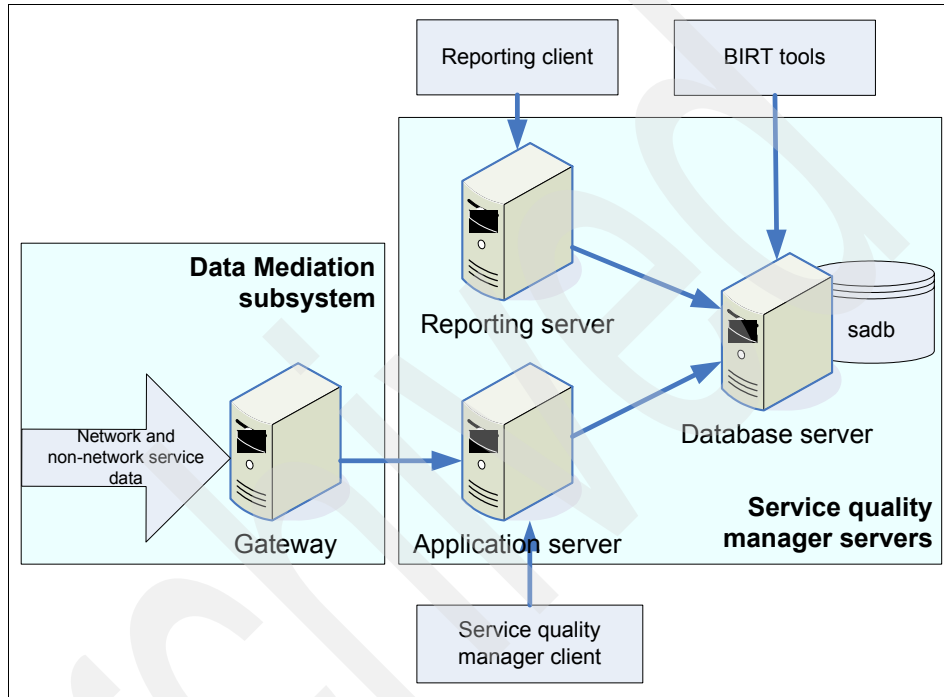


Figure 1-1 Service Quality Manager components

The Service Quality Manager components, as shown in Figure 1-1, are:

- ▶ The gateway machine, which collects data from network devices and pre-processes it into a format that Service Quality Manager understands. You can have more than one gateway machine, depending on your network's requirements.
- ▶ The application server is the main processing center of Service Quality Manager. It provides data collection and analysis of KPI and KQI. It measures the Service Level Agreement (SLA) and generates alerts for violations and warnings.
- ▶ The database server hosts the Service Quality Manager database (sadb). It runs the Oracle® Database server. In a smaller installation, the gateway, application, and database servers can reside on the same machine.

- The reporting server runs the BusinessObjects server. The BusinessObjects server provides reporting capability for Service Quality Manager. It reports historical data of certain KPIs and KQIs.

## 1.3 Service Quality Manager server

The main components of Service Quality Manager is the server components. These components can reside in a single server or be distributed over several different servers for load balancing and failover.

The server components can be viewed in more detail in Figure 1-2.

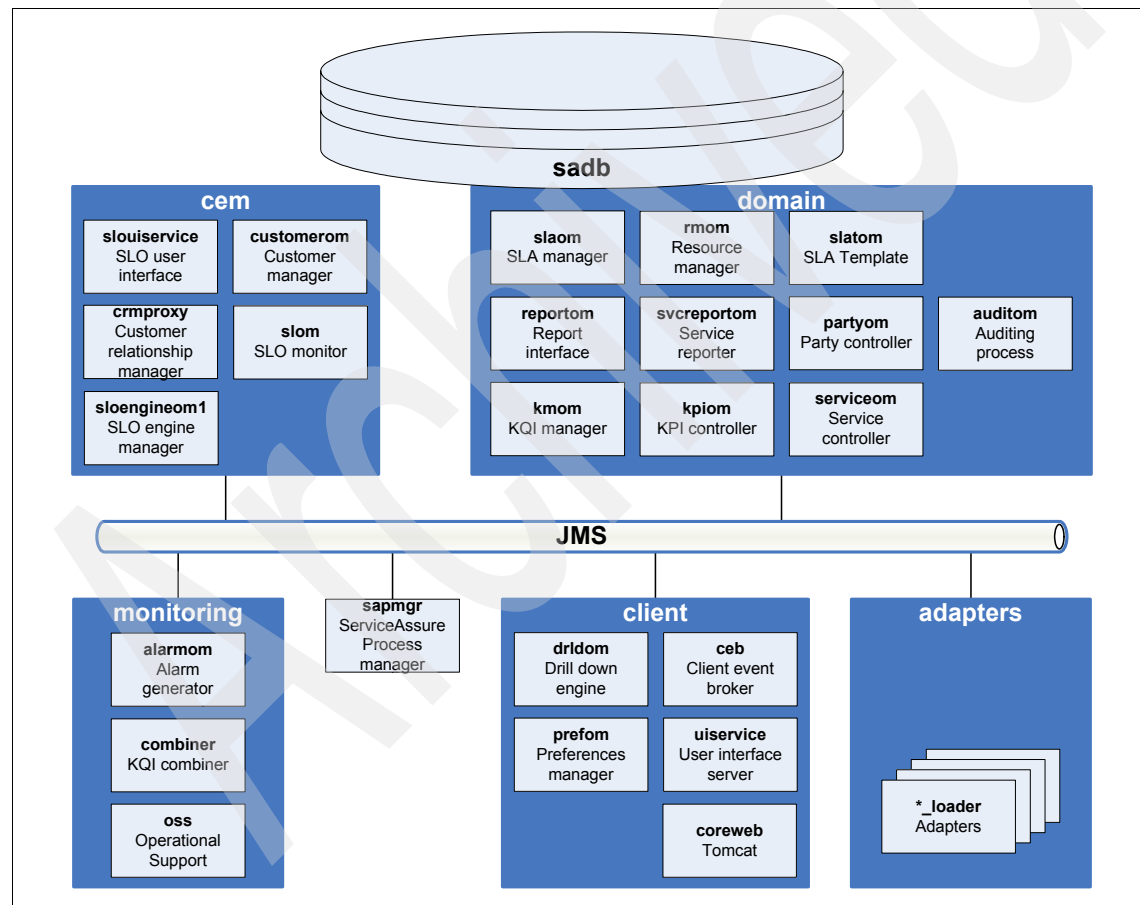


Figure 1-2 Service Quality Manager server components

The server components are grouped into the following categories:

- ▶ Domain processes, which are the primary object managers for controlling Service Quality Manager resources. The domain processes access the database and provide information about a certain object type using the messaging infrastructure.
- ▶ Customer Experience Management (CEM) processes, which has processes that extend the Service Quality Manager data model in V4.1.1 to analyze the overall customer experience. These processes are also object managers with access to databases and provide information about objects using the messaging infrastructure.
- ▶ Monitoring processes provide monitoring support for other processes and Service Quality Manager data. They centralize alert and notification management of Service Quality Manager.
- ▶ Client processes provide user interface support for accessing the Service Quality Manager application. These processes does not access the database directly, but communicate with the domain and CEM processes for object access.
- ▶ Adapter processes collect data from external data sources to be loaded into Service Quality Manager for processing. This is our main discussion point in this book.

## 1.4 Data mediation subsystem

Usually, data from a specific data source needs to be moved or extracted and go through some kind of transformation before it can be useful for any form of analysis. Most of the time, a business application provides a means to study this data, and presents it in a standard reporting format.

However, the common problem is that the data resides in various different systems, exists in a mixed variety of formats, and sometimes is available in varied instances.

Therefore, a process should exist to allow the data from a source to be moved to the end application or loaded to the database repository. Of course, transformation of the data would most likely take place between this process and trimmed for any unnecessary information. This entire process is known as *data mediation*.

The data mediation in Service Quality Manager provides a facility for getting data from a managed environment (device controller, database information, or console output) and placing it into Service Quality Manager. It performs the

necessary conversion and calculation of the source data into performance indicator information.

Data mediation is typically an extraction transformation and loading (ETL) process. It extracts data from the source, transforms it into the target format, and loads the data into a persistent storage (database). A data mediation gateway collects information from a vendor specific device manager and gets raw data. The raw data is formatted into a vendor neutral PIF format and then transformed into the final CSV format as required by the adapter. The PIF files are temporary files that are used in the intermediate processing stages within the gateway.

### 1.4.1 Extract Transform Load process

The overall process is an ETL process. This section describes the individual stages.

#### **Extract**

The first part of an ETL process is to extract the data from the source. Most data mediation projects consolidate data from several different sources. Each source may have different data formats. Extraction converts the data into a format for transformation processing.

An intrinsic part of the extraction is parsing of the extracted data, where the data is checked to see if it meets the expected pattern or structure; if not, the data can be rejected at this point.

The extract of the data is the beginning of the ETL process. This is where data is read and retrieved from source systems to a temporary transitional location, or a final location for further processing. A transfer may take place using different methods, such as a copy from a local server or through a transfer protocol (ftp, rcp, or scp) from a remote server.

There are two forms of how extraction could occur:

- **Push**

The data at the source location is extracted usually at the source server and transferred to the destination location. In this case, the process that performs the extract is running on the source server. Normally this happens in a situation when access to the source location is not permitted or to prevent additional server load at the destination.

Figure 1-3 shows the general idea of a push extract mechanism.

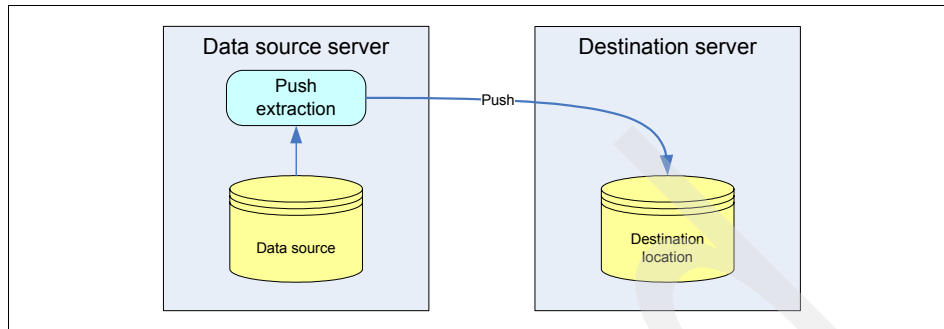


Figure 1-3 Push extraction

#### ► Pull

For a pull extraction, the data extract mechanism take place at the destination location and is downloaded from the destination location. The process performing the extract runs on the destination server instead. Generally, a pull extraction is ideal when a single point of reference for management is preferred. Figure 1-4 shows an illustration of how a pull extraction mechanism occurs.

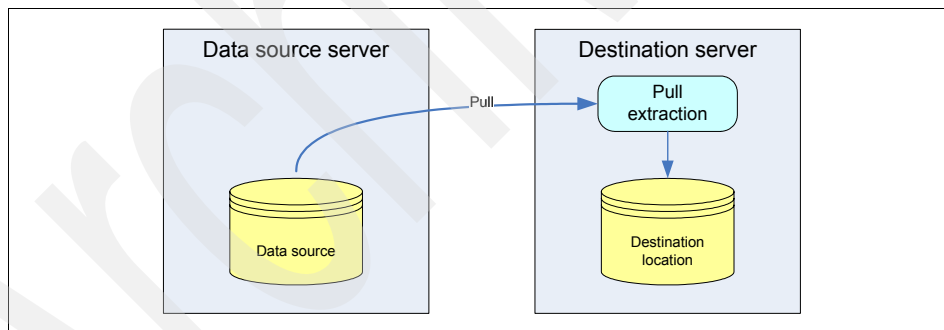


Figure 1-4 Pull extraction

## Transform

With the data extracted from the source, the next phase is to modify the source data format and manipulate it depending on the specific requirements of the target application. To achieve this process, we perform data transformation.

The transform stage applies a series of rules or functions to the extracted data to derive data that will be loaded in the end format. Some data sets may require little or no manipulation. In other cases, one or more of the following transformation types may be needed to meet the business requirements or

technical needs of the loading processes. Service Quality Manager gateways uses regular expressions for transformation processing.

The transformation types are:

- ▶ Selecting only certain columns to load (or selecting null columns not to load).
- ▶ Translating coded values (if the source system stores 1 for male and 2 for female, but the warehouse stores M for male and F for female). This is called automated data cleansing; no manual cleansing occurs during ETL.
- ▶ Encoding free-form values (mapping “Male” to “1” and “Mr” to M).
- ▶ Deriving a new calculated value ( $\text{sale\_amount} = \text{qty} * \text{unit\_price}$ ).
- ▶ Joining together data from multiple sources (lookup, merge, and so on).
- ▶ Summarizing multiple rows of data (total sales for each store and for each region).
- ▶ Transposing or pivoting (turning multiple columns into multiple rows or vice versa).
- ▶ Applying any form of simple or complex data validation; if the validation fails, a full, partial, or no rejection of the data occurs, and thus no, partial, or all the data is handed over to the next step, depending on the rule design and exception handling. Most of the above transformations might result in an exception, for example, when a code-translation parses an unknown code in the extracted data.

However, depending on the data source, there may be a series of transformation rules involved that obtains the final end result. Mapping of data from the source to the destination according to the element most likely will be required.

The end result is to obtain a uniform, standardized data format that can be loaded into a database of the end target for the next phase. In fact, the perfect scenario is when the data can be used in *any* kind of database. Apart from this purpose, an ideal transformation should be able to consolidate data from one data source system or format or from multiple data source systems or formats.

## Load

The load phase loads the data into the database, usually the data warehouse. Depending on the requirements of the organization, this process ranges widely. Some data warehouses might overwrite existing information weekly with cumulative, updated data, while other data warehouses or even other parts of the same data warehouse might add new data in a historized form, such as hourly. The timing and scope to replace or append data are strategic design choices that depend on the time available and the needs of the business.

As the load phase interacts with a database, the constraints defined in the database schema as well as in triggers activated upon data load apply (for example, uniqueness, referential integrity, mandatory fields, and so on) also contribute to the overall data quality performance of the ETL process.

With the transformation already performed, this should be a straightforward part of the ETL process. Checking data consistency also occurs during this stage to ensure loading is successful, and any possible bad formatted data is rejected. The schema about how the data is written may vary for different systems, that is, whether information will be overwritten, updated, or data added. This could be updated periodically (perhaps every 15 minutes or hourly). All of this actually depends on the design of the application providing further analysis of the data.

## 1.4.2 Gateway architecture

The purpose of the gateway is to provide a standard format for the transfer, parsing, manipulation, and presentation of data from network elements to the Service Quality Manager system.

Raw data is typically collected by the vendor supplied Operational Support System (OSS) or Operation Maintenance Center (OMC). Collection intervals can vary, but typical configurations are 15 or 30 minutes. The OMC must be configured by the service provider to output the desired raw performance data. The files are often very large and can be archived for several days within the OMC. The gateway is typically interested in near real time performance data.

The gateway 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 quality management application to generate a service quality indicator and service level monitoring.

Gateways usually receive data in a specific data format from a network appliance vendor. Each gateway processes a specific data source based on the output provided from the vendor's data format. Gateways generate performance data that is vendor neutral, but technology specific. This is in contrast to the service solution adapter that processes a technology specific data to load it to Service Quality Manager.

The purpose of the gateway framework is to provide a standard framework for the transfer, parsing, manipulation, and ultimately the presentation of performance data from network elements to the Service Quality Manager system.

The raw data is typically transferred from the network element vendor supplied Operation Management Console. The data is available at fixed periods, usually in increments of 15 or 30 minutes, or up to several hours. Retrieving several hours of data can be very time consuming.

The framework is made up of several processing stages that perform different functions to output the desired data and data format. It employs standardized tools for logging, measuring integrity, and recovery, and rules for transforming the data into the required output. The final output is then ready for the Service Quality Manager application to use to manage various telecommunication and enterprise customer needs. The processing of the gateway framework is shown in Figure 1-5.

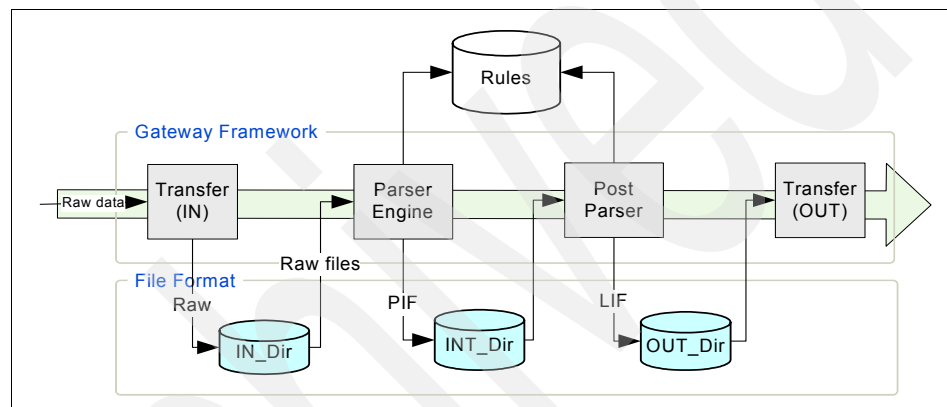


Figure 1-5 Gateway framework characteristics

The gateway framework consists of three main components:

- ▶ Transfer engine (IN and Out). This component is optional, and the file can also reside directly on the gateway machine.
- ▶ Parser engine.
- ▶ Post parser engine.

These components interact mainly with various directories for reading, writing, and storing files. The parser engine converts data from the raw data original format to a PIF file. This conversion is vendor specific, as each vendor may have different types of source file. The post parser engine works with the PIF file. It reads one or more PIF files and writes to another PIF file for further processing. The final output of the post parser engine is dictated by the consumer of the data; Service Quality Manager uses CSV format, while Performance Manager for Wireless uses LIF format.

### 1.4.3 PIF file format

As Parser Intermediate File (PIF) is used considerably in gateway processing, it is important to understand its construction. Example 1-1 shows a sample PIF file.

*Example 1-1 PIF file sample*

---

```
## Parser Intermediate File
##START|HEADER
day|startTime|BLOCKNAME|endTime|FILENAME|systemName
14Oct2008|28800|datablockname|29700|Data-sample.csv|sys1
##END|HEADER
##START|datablockname
host|pkgcount|pkgdelete|pkgfail|systemIPAddress
app1248|1613949|17|7|212.47.101.248
server090|282730|56|9|212.47.101.90
server121|8319912|88|3|212.47.101.121
##END|datablockname
```

---

The PIF file contains a header and one or more data blocks. The header contains information that is common to all rows of the data blocks. Fields in PIF are separated by the ‘|’ character. Lines starting with double hashes (##) indicate directives, while other lines contain data.

# Gateway concepts and processing

This chapter describes in-depth information about Service Quality Manager gateways. We explain the concepts and development of a gateway module in Service Quality Manager. This chapter contains the following sections:

- ▶ 2.1, “Gateway architecture” on page 14
- ▶ 2.2, “Perl library extension and engine configuration” on page 19
- ▶ 2.3, “Working with transfer configuration rules” on page 20
- ▶ 2.4, “Working with post parser rules” on page 23

## 2.1 Gateway architecture

In this section, we discuss the gateway architecture in detail. We discuss:

- ▶ 2.1.1, “Gateway components” on page 14
- ▶ 2.1.2, “Gateway operation” on page 16

### 2.1.1 Gateway components

This section discusses the gateway architecture and its main components. The gateway is made up of:

- ▶ “Gateway framework” on page 14
- ▶ “Vendor gateway” on page 15
- ▶ “Gateway configuration” on page 15

Figure 2-1 shows the gateway components.

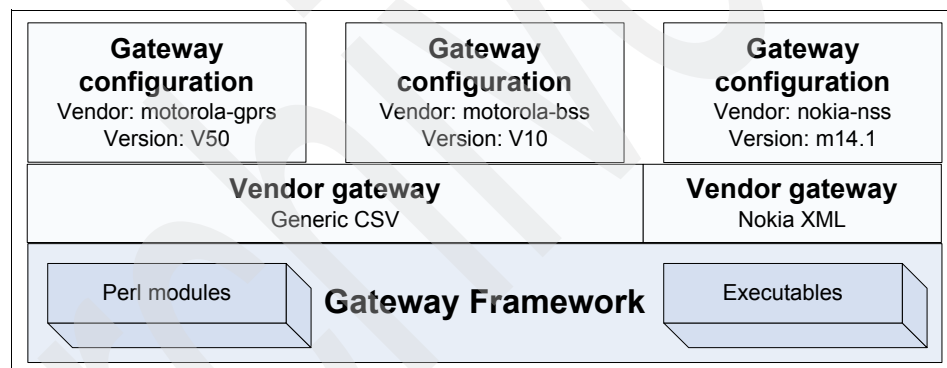


Figure 2-1 Gateway components

#### Gateway framework

The gateway framework provides the ability to perform the extraction and transformation capability in the ETL operation. The gateway framework provides:

- ▶ Executables for running the gateway, including the infrastructure for logging and file management.
- ▶ Predefined Perl modules that can be called or invoked. These modules are usually invoked as a set of rules that can be configured from a file.

The gateway framework contains the following directories:

- ▶ vstart: Startup programs for the gateway
- ▶ parsersrc: Storing customized Perl modules (.pm files)

- ▶ `perl_extensions`: Perl modules repository for the post parser rules

The processing of the gateway framework itself allows:

- ▶ Transfer of data for extraction purposes. This part of the module actually runs in two phases.
  - Inbound transfer (IN) executes before parsing of the data
  - Outbound transfer (OUT) executes after parsing of the data
- ▶ Parsing of the data based on a vendor specific data format for transformation operation.

The gateway framework files resides under the `gateway-framework` sub-directory of `$GATEWAY_ROOT`.

## Vendor gateway

The vendor gateway package consists of a number of configuration Perl modules that perform the manipulation of the vendor specific data. Vendor gateways are typically designed for processing data from a specific vendor format. It reads vendor data and converts it into a common format for further processing. Usually only the required Engine module needs to be installed for a specific vendor gateway to function.

Vendor gateway files resides under the `modules` sub-directory of `$GATEWAY_ROOT`. The current provided vendor gateways from Gateway Framework V3.4.2.1 are for:

- ▶ 3gpp ASN1 V3.4.0.2
- ▶ 3gpp XML V3.4.3.1
- ▶ Alcatel BSS V3.4.0.3
- ▶ Alcatel NSS V3.4.0
- ▶ Ericsson GSM V3.4.0
- ▶ Generic ASCII V3.4.0
- ▶ Generic CSV V3.4.1.1
- ▶ Nokia ASCII V3.4.0.1
- ▶ Nokia XML V3.4.0.2
- ▶ Nortel BSS V3.4.0.1
- ▶ Siemens BSS V3.4.0.1
- ▶ Siemens NSS V3.4.1.1
- ▶ Siemens V3.4.0

## Gateway configuration

The gateway configuration files allow the vendor gateway to be more suitable for vendor specific data. The gateway configuration files are available as technology packs in IBM Tivoli Netcool Performance Manager for Wireless, a sister product

to Service Quality Manager. The gateway configuration files are installed in the config sub-directory of \$GATEWAY\_ROOT.

More details about the IBM Tivoli Netcool Performance Manager for Wireless product can be found at the following URL:

<http://www-01.ibm.com/software/tivoli/products/netcool-performance-mgr-wireless/>

Technology Packs are discussed at the following URL:

<http://www-01.ibm.com/software/tivoli/products/netcool-performance-mgr-wireless-techpacks/>

## 2.1.2 Gateway operation

The gateway is invoked using the script `gateway_start.sh` from the `vstart` directory in Gateway Framework. The vendor gateway configuration is provided as the relevant vendor, sub-system, and version arguments to this script. The vendor, sub-system, and version arguments are used to qualify the directory in the {GATEWAY\_ROOT}/config directory.

The overall operation of a gateway executable is shown in Figure 2-2. It provides details about the processing of the gateway.

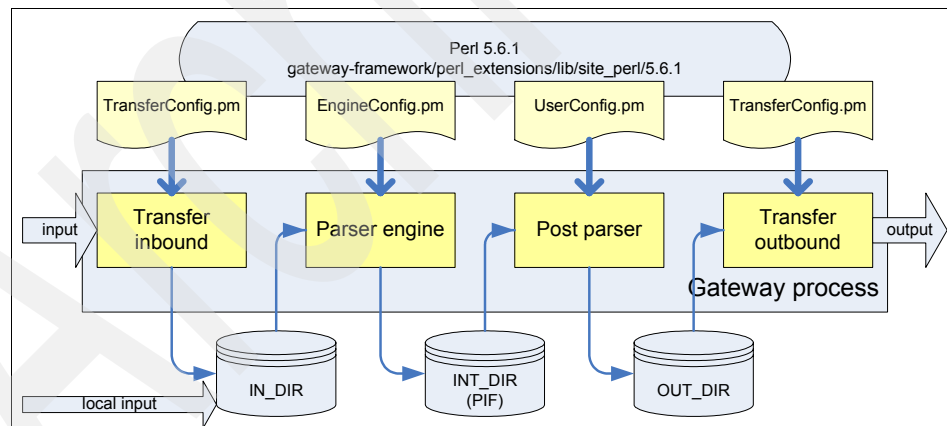


Figure 2-2 Gateway processing characteristics

The gateway requires a specific version of the Perl modules. The version that we use requires Perl 5.6.1. You can find instructions for using Perl in your platform at the following URL:

<http://www-01.ibm.com/support/docview.wss?uid=swg27011991>

The next section goes through each stage and module of the data mediation process for the gateway process. Note that we use the terminology IN\_DIR, INT\_DIR, and OUT\_DIR as variables that are specified from the gateway properties file.

### **Inbound transfer**

Depending on whether the data extraction is push or pull, as explained in “Extract” on page 7, this optional stage enables the configuration of the transfer of data from the source. It supports the SFTP, SCP, FTP, RCP, and local CP (copy) file transfer mechanisms. The transfer rule allows the configuration of multiple instances, so extraction of files from different source locations are possible. However, these rules are executed in sequence, not in parallel.

Configuration of this stage is optional when data is pushed to the gateway or provided by another external tool. The configuration file for this stage is in `TransferConfig.pm`.

### **Parser engine**

In this stage, the engine parses the raw data source file, which could be either in a vendor specific data format, or a third-party data format, to produce the data in Parser Intermediate Format (PIF). It picks up the raw data source file from IN\_DIR, and temporarily stores it as a PIF file in INT\_DIR.

The parser engine executes based on a specific vendor specific engine rule. Each vendor gateway provides its own configuration module. For example, to parse Nokia NSS format data, the NOKIA\_XML engine rule is used. Another rule uses the CSV\_GATEWAY that we use for parsing Motorola GPRS CSV format data. The configuration file for this stage is in `EngineConfig.pm`.

### **Post parser**

The post parser stage further processes the data through multiple standard and vendor specific post parser rules, transforming data into the final desired output format for successful loading that conforms to the system requirements.

There are several transformation functions possible in this stage. These rules are implemented in the gateway framework, so it is available for all vendors. The following are some of the common rules:

- ▶ Joining different PIF files using the JOIN rule.
- ▶ Accumulate.
- ▶ CVAL\_MANIP.
- ▶ INFOINSERT.
- ▶ Removing entire data records with the DATALINE\_WHERE rule.

- ▶ Customized manipulation of data using the PERLIZE rule.
- ▶ Converting a PIF file to a desired final output file format with the PIF\_2\_OUTPUT rule.

For a complete listing and detailed explanation of these transformation functions, see the standard post parser rules in section 5 of the *Gateway Framework User Guide*, which is provided as a PDF in the Service Quality Manager Gateway installation media. We discuss how to configure some of these rules when developing sample development in 3.2, “Gateway configuration for Motorola GGSN” on page 51.

In this stage, the post parser rule can produce two types of output:

- ▶ PIF files, which are used as input to the next post parser rule in the processing chain. They are temporarily written to INT\_DIR.
- ▶ Output files, which are output to another directory as the final output file format for loading into the system repository. These are output to OUT\_DIR and could be in one of the following formats:
  - LIF files
  - CSV files
  - XML files

The configuration file for this stage is in UserConfig.pm.

## Outbound transfer

With the final output files generated, this stage moves them to a separate location for loading in a remote server or other servers, if necessary. Again, the transfer engine is responsible for this process, and this is an optional stage.

**Note:** Another option is to directly configure OUT\_DIR as the final location, and so we can skip this part. However, this is solely up to your discretion, as it is sometimes preferable to keep the directories separate for application administration and management reasons.

The configuration file for this stage is in TransferConfig.pm.

## Other features

The gateway framework also includes other features that are available for all vendor gateways. These additional features include:

- ▶ Logging and auditing
- ▶ Crash recovering
- ▶ Backlog processing
- ▶ Parallel processing

- ▶ Memory caching of intermediate data
- ▶ Saving of parsed, intermediate, and load data
- ▶ Compression of input and stored data

The gateway configuration defines the rules for parsing various data from specific vendor subsystems. The main configuration files in the directory are `TransferConfig.pm` (optional), `EngineConfig.pm`, and `UserConfig.pm`. The vendor gateways (Technology Packs) already provide a base configuration platform. However, you may modify the configuration to better suit your environment.

## 2.2 Perl library extension and engine configuration

Service Quality Manager gateways uses Perl as the primary programming language. Perl is easily extended using Perl modules. Perl modules are text files with the extension of `.pm`. The Perl modules are constructed in such a way that they provides library functions that can be invoked from the vendor gateway configuration.

Gateway framework defines that Perl modules must have the following sub-routines:

- ▶ `New`: Initialize the module.
- ▶ `load_config`: Loading configuration files.
- ▶ `process_a_file`: Process the data.

The Perl modules for `EngineConfig.pm` are typically provided with the vendor gateway installation. A sample engine rule definition is for CSV raw data; the module is called `GENERIC_CSV_GATEWAY`.

Typically, the gateway module will provide the parser engine module for loading raw data. A generic parser engine is provided, such as for CSV file handling. The CSV file handler is called the `GENERIC_CSV_GATEWAY` rule. We discuss more about this generic CSV gateway in 3.4, “Modifying the Generic CSV parser engine” on page 56.

Example 2-1 shows a sample rule for a generic CSV gateway.

*Example 2-1 Generic CSV gateway*

---

```
'RULE_TYPE'           => 'GENERIC_CSV_GATEWAY',
'RULE_DESC'           => 'Motorola GGSN v50',
'INPUT_FILE_DESCRIPTION' => [ '^PassiveCollector.*$' ],
'INPUT_DIR_DEPTH'     => 0,
'ORDER_OF_FILES'      => 'OLDEST_FIRST',
'FIELD_SEPARATOR'     => ',',
'HEADER_FIELDS'       => [ 3 ],
'DATA_FIELDS'         => [ 0, 1, 2, 4, 5, 38 ],
'INVALID_VALUE_MATCH' => [],
'WHITESPACE_MODIFIER' => '-',
'FIELDS_TO_KEY_PIF_FILENAME' => [],
'DEFAULT_NULL_VALUE'  => '',
'UPPERCASE_COUNTERS'  => 0,
'PIF_FILENAME_EXTRACT' => 'PassiveCollector-C_IBM.pm-gw_(\w+)\.',
'FILENAME_HEADER_FIELDS' => {
    BLOCKNAME =>
'PassiveCollector-C_IBM.pm-gw_([A-Za-z]*)_[0-9]{2}_[0-9]{2}_[0-9]{4}_[0-9]{2}_[0-9]{2}_[0-9]{2}\\.csv',
```

---

If you need a specialized gateway for your own use, you can create your own Perl module.

## 2.3 Working with transfer configuration rules

Transfer configuration rules templates are located in the gateway directory `gateway-framework/perl_extensions/lib/site_perl/5.6.1/TransferEngine`. The defined protocols are `ftp` or `scrcp`.

### 2.3.1 FTP transfer

The FTP based transfer uses the following mandatory arguments:

<b>HOST</b>	Target ftp server host name or IP address
<b>DIRECTION</b>	IN or OUT
<b>REMOTE_DIR</b>	Remote path of the file
<b>LOCAL_DIR</b>	Local path of the file

<b>TIMESTAMP_FILE</b>	File name to store the list of files in that have been processed
<b>DEPTH</b>	Number of levels from the remote directory to descend
<b>INPUT_FILE_DESCRIPTION</b>	List of hash values that are used to specify the file name characteristic to be transferred
<b>USER</b>	User ID of the target FTP server
<b>PASS</b>	Password of the target FTP server
Additionally, the transfer uses the following optional arguments:	
<b>STATISTICS_FILENAME</b>	File name to save the statistic output in
<b>OUTPUT_FORMAT</b>	Unused
<b>ENABLE_PING</b>	Whether or not to check with the <b>ping</b> command first
<b>PING_PROTOCOL</b>	Whether to use tcp or udp for running the <b>ping</b> command.
<b>PING_RETRY_ATTEMPTS</b>	Number or retries to perform for the <b>ping</b> command
<b>RETRY_INTERVAL</b>	Retry interval for the ftp connection attempt
<b>TIMEOUT</b>	Timeout to wait for the ftp connection in seconds
<b>ENABLE_LOCAL_COMPRESSION</b>	Whether or not to compress local files
<b>NUMBER_OF_FILES_TO_PROCESS</b>	Maximum number of files to process in each iteration
<b>TMP_FILENAME_EXTENSION</b>	File extension to be use for temporary file names
<b>DELETE_ORIGINAL</b>	Whether to delete the original files after ftp (1/true or 0/false)
<b>OVERWRITE_FILES</b>	Whether to overwrite a local file (1/true or 0/false)
<b>TRANSFER_MODE</b>	Set binary or ASCII transfer

### 2.3.2 SCP/RCP transfer

The FTP based transfer uses the following mandatory arguments:

<b>HOST</b>	Target ftp server host name or IP address
<b>DIRECTION</b>	IN or OUT

<b>PROTOCOL_PATH</b>	The executable of <b>ssh</b> or <b>scp</b> for an scp protocol or <b>rsh</b> or <b>rcp</b> for an rcp or cp protocol.
<b>REMOTE_DIR</b>	Remote path of the file
<b>LOCAL_DIR</b>	Local path of the file
<b>TIMESTAMP_FILE</b>	File name to store list of files that have been processed
<b>INPUT_FILE_DESCRIPTION</b>	List of hash values that is used to specify the file name characteristic to be transferred

Additionally, it uses the following optional arguments:

<b>BULK_TRANSFER</b>	Bulk copying using the <b>cpio</b> utility
<b>DEPTH</b>	Number of levels from the remote directory to descend
<b>STATISTICS_FILENAME</b>	File name to save the statistic output
<b>OUTPUT_FORMAT</b>	Unused
<b>ENABLE_PING</b>	Whether or not to check with the <b>ping</b> command first
<b>PING_PROTOCOL</b>	Whether to use tcp or udp to run the <b>ping</b> command
<b>PING_RETRY_ATTEMPTS</b>	Number of retries to perform for the <b>ping</b> command
<b>RETRY_ATTEMPTS</b>	Number or retries to connect to the remote host
<b>RETRY_INTERVAL</b>	Retry interval for ftp connection attempts
<b>TIMEOUT</b>	Timeout for waiting for the ftp connection in seconds
<b>ENABLE_LOCAL_COMPRESSION</b>	Whether or not to compress local files
<b>ENABLE_REMOTE_COMPRESSION</b>	Whether or not to compress local files
<b>NUMBER_OF_FILES_TO_PROCESS</b>	Maximum number of files to process in each iteration
<b>TMP_FILENAME_EXTENSION</b>	File extension to be use for temporary file names
<b>DELETE_ORIGINAL</b>	Whether to delete the original files after ftp (1/true or 0/false)
<b>OVERWRITE_FILES</b>	Whether to overwrite a local file (1/true or 0/false)

## 2.4 Working with post parser rules

The post parser rules deal with a known format, the PIF file, and so the rules are more generic. The Service Quality Manager gateway framework provides a wealth of templates for processing data. The `UserConfig.pm` listing for the sample post parser rules is provided in “Sample UserConfig.pm” on page 92.

The following sections discuss some Perl rules that can be used in the `UserConfig.pm`:

- ▶ “ACCUMULATE” on page 24
- ▶ “AGGREGATE\_LINE” on page 25
- ▶ “CVAL\_MANIP” on page 26
- ▶ “DATALINE\_WHERE” on page 28
- ▶ “FILE\_SPLIT” on page 29
- ▶ “FILE\_SPLIT\_BY\_COUNTERS” on page 30
- ▶ “INFOINSERT” on page 31
- ▶ “JOIN” on page 34
- ▶ “MERGE\_RECORDS” on page 35
- ▶ “PERLIZE” on page 36
- ▶ “PIF\_2\_CSV” on page 37
- ▶ “SPLIT\_RECORD” on page 38
- ▶ “UNPEGGER” on page 40

### 2.4.1 Common parameters

There are some common, optional parameters for these rules. They are:

#### **REDUNDANT\_HEADER\_COUNTERS**

This is a list of counters to remove from the output header.

#### **REDUNDANT\_DATA\_COUNTERS**

This is a list of counters to remove from the output data.

#### **HEADER\_COUNTERS\_ORDER**

A set of header fields used in a certain order.

#### **DATA\_COUNTERS\_ORDER**

A set of data fields used in a certain order.

## 2.4.2 ACCUMULATE

The ACCUMULATE rule aggregates counter values based on the grouping of an identifier field. The default operation is to add all non-key columns. You can choose columns to be aggregated as maximum, minimum, or average. If you use these functions, you must also specify the string appendage to the counter name. The process for ACCUMULATE is shown in Figure 2-3.

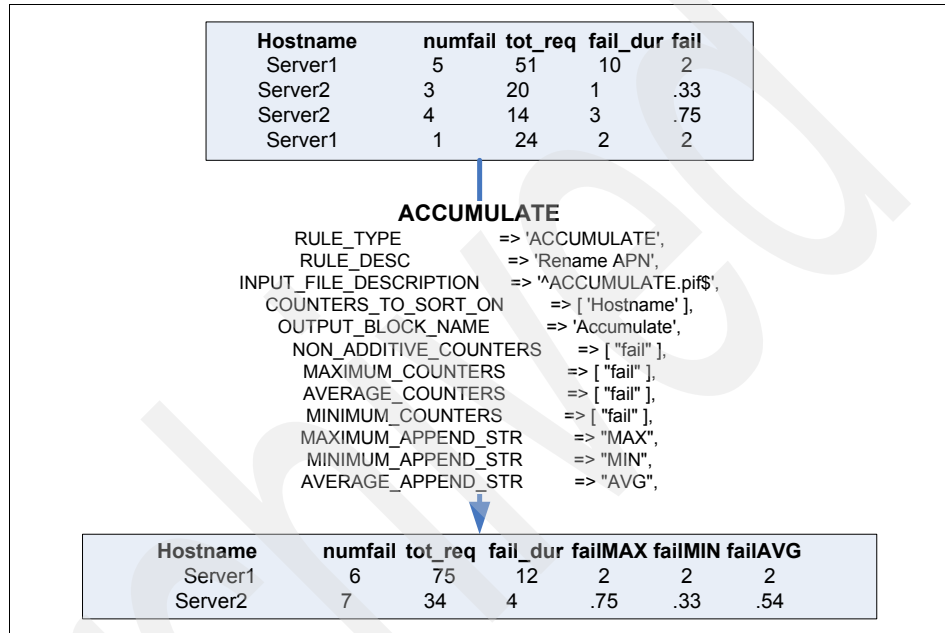


Figure 2-3 ACCUMULATE rule processing

The required parameters are:

### OUTPUT\_BLOCK\_NAME

Block name for the output of the PIF file

### COUNTERS\_TO\_SORT\_ON

Sorting key for the table aggregation

The optional parameters are:

### COUNTER\_NULL\_VALUE

Null value substitution if counter calculation has an error.

### NON\_ADDITIVE\_COUNTERS

Counters that cannot be accumulated. They are passed as is. An example are the time stamp fields.

**MAXIMUM\_COUNTERS**

Counters to calculate the maximum value.

**MINIMUM\_COUNTERS**

Counters to calculate the minimum value.

**AVERAGE\_COUNTERS**

Counters to calculate the average value.

**MAXIMUM\_APPEND\_STR**

String to append to the field name for maximum values.

**MINIMUM\_APPEND\_STR**

String to append to the field name for minimum values.

**AVERAGE\_APPEND\_STR**

String to append to the field name for average values.

**2.4.3 AGGREGATE\_LINE**

The AGGREGATE\_LINE rule matches counters from a single line into a single counter value. The process is shown in Figure 2-4. It adds all counters in the field that matches the regular expression (pscc\_ \d+) into a new counter name, which is the HASH key for the COUNTER\_GROUPS parameter (pscc). Typically, you would then remove the aggregated counters using the REDUNDANT\_DATA\_COUNTERS parameter.

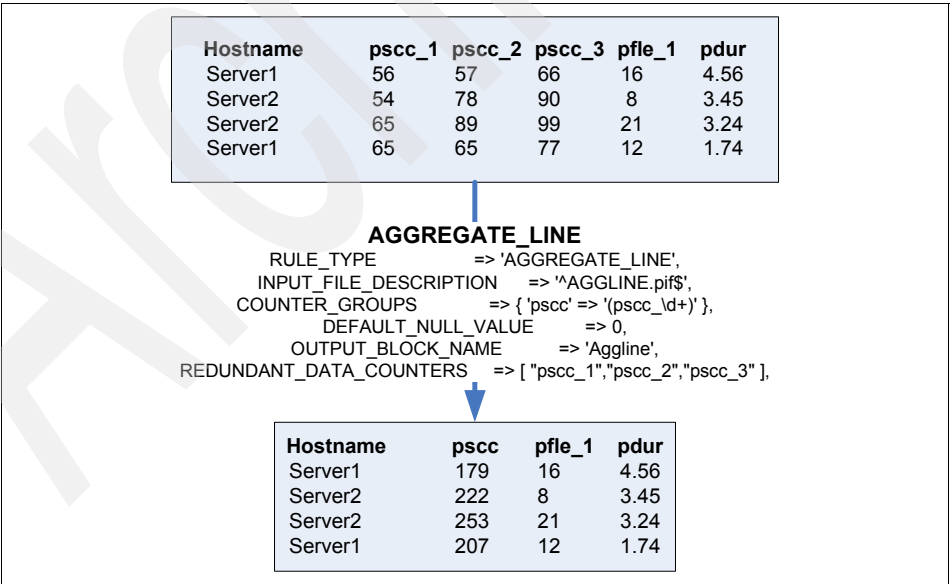


Figure 2-4 AGGREGATE\_LINE rule

The required parameters are:

**COUNTER\_GROUPS**

Hash to new counters with aggregates from values that matches a regular expression.

**DEFAULT\_NULL\_VALUE**

Null value substitution if the counter calculation has an error.

The optional parameters are:

**OUTPUT\_BLOCK\_NAME**

The name of the output data block

**COUNTER\_NAME\_TO\_EXTRACT**

Counter names that would be extracted

## 2.4.4 CVAL\_MANIP

The counter value manipulation (CVAL\_MANIP) rule is used for string manipulation of the contents of the PIF file. This helps prepare the PIF file for loading, rearranging fields, and combining them to get the format you want. Some usage examples are:

- ▶ Wrapping counter values in quotes
- ▶ Remove a string from a value
- ▶ Replacing a counter value
- ▶ Rearranging a counter value

The pattern matching is performed for the counter specified in CNAME\_MANIP. Regular expressions are stored in the MATCH parameters. Matching stops at the first match, so the most specific must be specified first in the list. The example in Figure 2-5 on page 27 uses a catch-all expression (\w+) last. The substitution is using the regular expression matching from the entry in the PATTERN argument, such as \$1 is the first matched string and so on. The number of elements in MATCH and PATTERN must be the same.

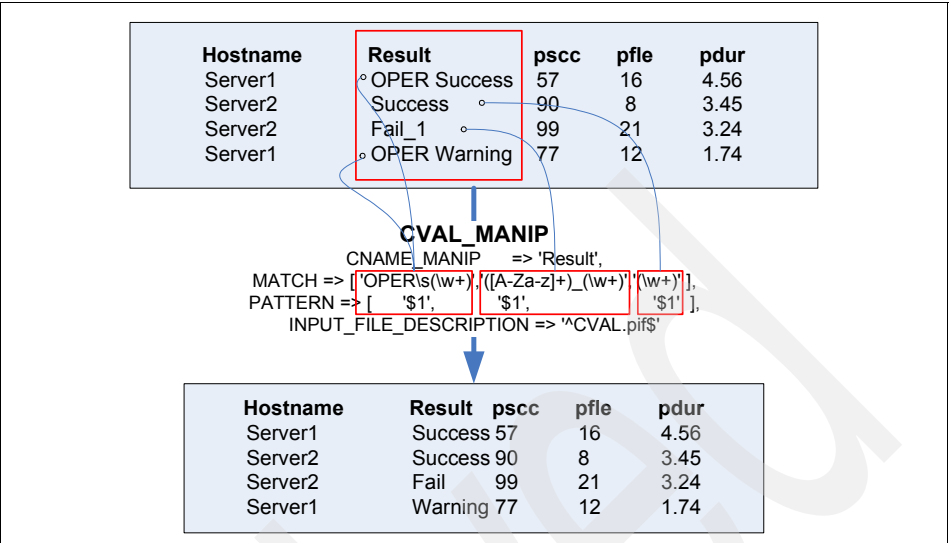


Figure 2-5 Counter value manipulation rule

The required parameters are:

- CNAME\_MANIP** This is the name of the counter to be manipulated.
- MATCH** This is 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.
- PATTERN** This is a list of patterns that define how the counter value will be manipulated. There is a 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.

The optional parameters are:

- OUTPUT\_BLOCK\_NAME** This is the block name to use in the output data.

**OUTPUT\_DIR** This is the name of an alternate directory to which the loader files can be written for this rule. If not configured, then LIFs will be written to the configured parser output directory.

**NON\_MATCH\_RECORD** This value determines 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.

## 2.4.5 DATALINE\_WHERE

The DATALINE\_WHERE rule includes or excludes a data row depending on the counter pattern value. The COUNTER\_NAMES rule is a hash of values of COUNTER\_NAME, KEEP\_WHERE, and REMOVE\_WHERE as the conditions to keep or remove the row. The process is shown in Figure 2-6. You can also add a new counter.

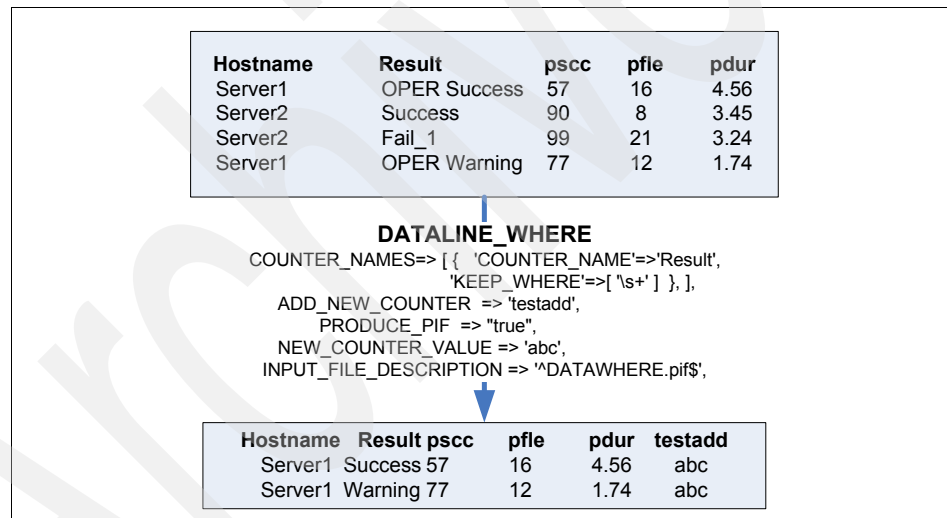


Figure 2-6 The DATALINE\_WHERE rule

The required parameters are:

**COUNTER\_NAMES** Counter names to check. This is an array of a hash table. Each hash entry must contain COUNTER\_NAME and either the KEEP\_WHERE or REMOVE\_WHERE keys.

The optional parameters are:

**ADD\_NEW\_COUNTER**

New counter name to add

**NEW\_COUNTERS\_VALUE**

Value of the new counter

**OUTPUT\_BLOCK\_NAME**

Output data block name

**FILENAME\_ADDITION**

String to add to the file name

**OUTPUT\_DIR**

Output file path

## 2.4.6 FILE\_SPLIT

This FILE\_SPLIT rule splits PIFs based on a counter's field content. Different rows may end up in different files. First, you specify the key counters to perform the sorting. Then you specify the regular expression that you want to use to split the file. Each unique value (from matching the key counters to the regular expression) generates a new file. You can also rename a set of counter names. The process is shown in Figure 2-7.

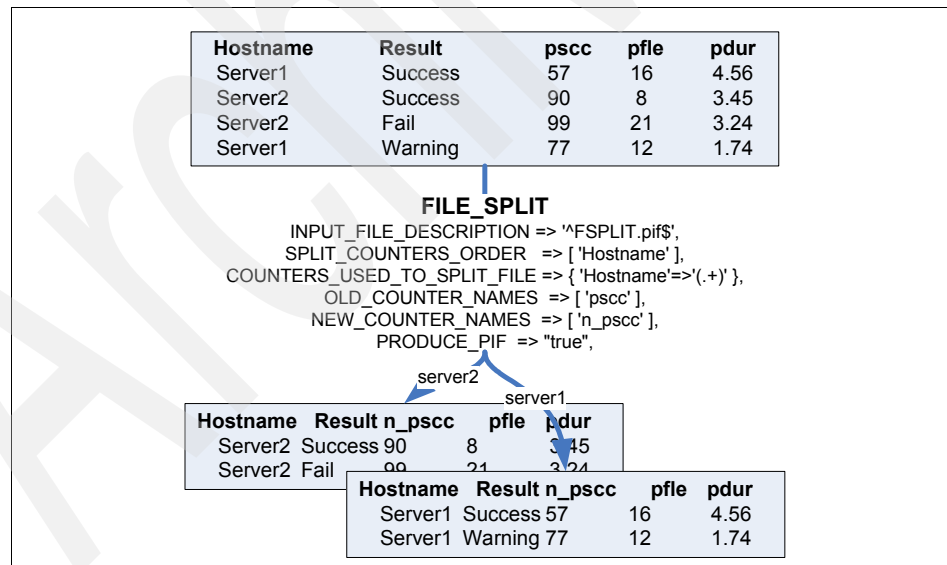


Figure 2-7 The FILE\_SPLIT rule

The required parameters are:

**SPLIT\_COUNTERS\_ORDER**

The order of counters for applying a regular expression that will be used to split the file.

**COUNTERS\_USED\_TO\_SPLIT\_FILE**

The array of a regular expression used to obtain the split file identifier. The number of entries in COUNTERS\_USED\_TO\_SPLIT\_FILE must match SPLIT\_COUNTERS\_ORDER.

The optional parameters are:

**NEW\_COUNTER\_NAMES**

List of new counter names. This parameter must be used with OLD\_COUNTER\_NAMES and has the same number of entries.

**OLD\_COUNTER\_NAMES**

List of old counter names. This parameter must be used with NEW\_COUNTER\_NAMES and has the same number of entries.

## 2.4.7 FILE\_SPLIT\_BY\_COUNTERS

The FILE\_SPLIT\_BY\_COUNTERS rule splits a PIF by putting counters on different files. You specify the counter names that you want to put into each files. The process is shown in Figure 2-8 on page 31.

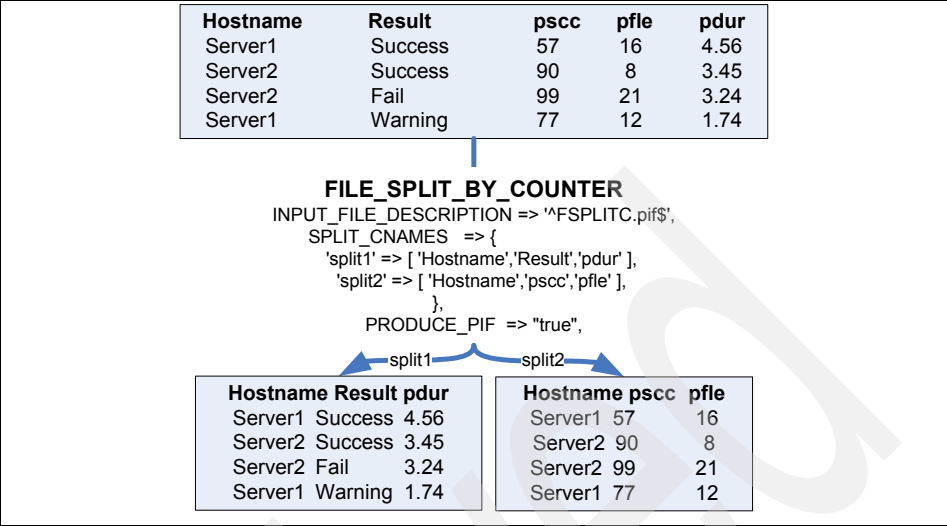


Figure 2-8 FILE\_SPLIT\_BY\_COUNTERS rule

The required parameters are:

**SPLIT\_CNAMES** Counter names used to split the file

The optional parameters are:

**ONLY\_INSERT** Array of counter names to be inserted into the output

**WRITE\_DATA\_LINE** Whether or not to write the data line

## 2.4.8 INFOINSERT

The INFOINSERT rule is used when we wish to include information from one PIF file in another PIF file. The main difference between this rule and the JOIN rule is that we do not wish to simply aggregate PIF files: here, we are actually processing them to get a much more complete and complex PIF file. This typically happens when we need to add hierarchical information to a PIF file that has only counters, based on some counter pattern presented in it.

The INFOINSERT 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. The INFOINSERT rule requires at least two input files for processing: The file from where data will be extracted (lookup.pif), and the file where data will be inserted (infoins.pif). The process is shown in Figure 2-9.

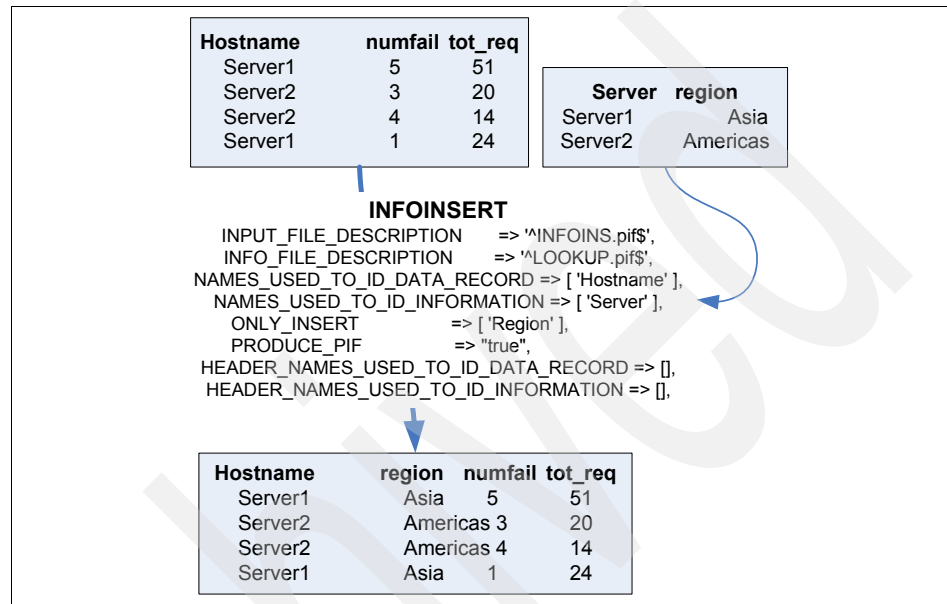


Figure 2-9 INFOINSERT rule

The required parameters are:

#### INFO\_FILE\_DESCRIPTION

This is 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.

#### HEADER\_NAMES\_USED\_TO\_ID\_DATA\_RECORD

This is 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 into this data record.

#### NAMES\_USED\_TO\_ID\_DATA\_RECORD

This is 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.

## **HEADER\_NAMES\_USED\_TO\_ID\_INFORMATION**

This is 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.

## **NAMES\_USED\_TO\_ID\_INFORMATION**

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

The optional parameters 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.

### **WRITE\_DATA\_LINE**

This option controls the action of this rule when there is no information to substitute for a line of data, that is, 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 output.

### **OUTPUT\_BLOCK\_NAME**

This is the name that should be used for the section name in the loader file.

### **OUTPUT\_FILENAME\_START**

This is a prefix that the output file name will start with.

### **OUTPUT\_DIR**

This is the output directory. The default is stored in the properties file.

### **INFO\_FILES\_STORAGE\_DIR**

This is an optional scalar entry containing a directory name where information files can be stored. Information files not stored are deleted, as is 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.

## 2.4.9 JOIN

The JOIN rule merges two PIF files into a single file based on matching file names and matching counters. The process is shown in Figure 2-10. We advise using the OUTPUT\_FILENAME\_START as well so the file name does not start with a hyphen (-).

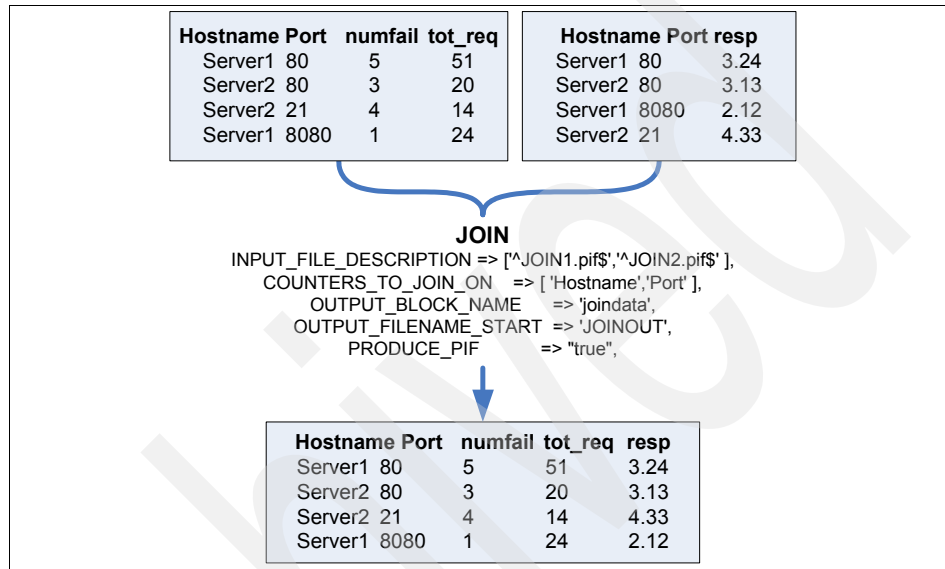


Figure 2-10 JOIN operation

The required parameters are:

### COUNTERS\_TO\_JOIN\_ON

Source counters for joining the files

### OUTPUT\_BLOCK\_NAME

Output data block name

The optional parameters are:

### OUTPUT\_FILENAME\_START

This is a prefix to use on the output file name.

### HEADER\_COUNTERS\_TO\_USE\_IN\_OUTPUT\_FILENAME

This is 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 a PIF file waits 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.

**TIME\_JOINFILE\_PRODUCED**

If set to TRUE, the rule will add a time and date to the output joined file. The time and date is in the following 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. This can be useful when trying to debug the joining of a large number of files or a complex rule.

**NEW\_HEADER\_COUNTERS**

Additional header fields to be created.

**OUTPUT\_DIR**

Destination path.

**2.4.10 MERGE\_RECORDS**

The MERGE\_RECORDS rule merges multiple records into a single records by sorting the key counters. The process is similar to ACCUMULATE, but the key difference is that merging a record does not process the ACCUMULATE rule, but instead keeps all the individual counters based on the grouping key. The process is shown in Figure 2-11. The port key is used for grouping the data, and all the different port numbers generate different counter names, resulting in some null values.

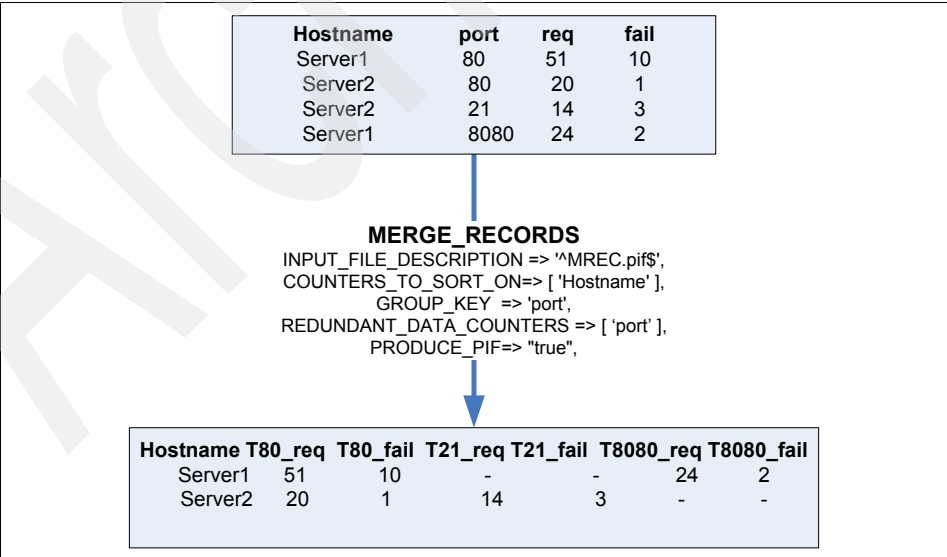


Figure 2-11 MERGE\_RECORDS rule

The required parameters are:

**COUNTERS\_TO\_SORT\_ON**

Counters to sort for a merging record

The optional parameters are:

**GROUP\_KEY**

The counter name that is used to define grouped records.

**MANIP\_ONLY**

Only return counters are manipulated.

## 2.4.11 PERLIZE

PERLIZE is, by far, the most flexible of all rules, for one very simple reason: It allows you to input your own Perl program into the gateway framework! A support engineer with Perl programming knowledge can correct, even on site, a problem that exists within a gateway, bypassing all the processing by using the PERLIZE rule.

Of course, this advantage is also its greatest disadvantage: using PERLIZE to excess may lead to documentation problems (for example, only the professional that programmed those lines on site will understand what happened if he or she forgets to document his work).

Also, using PERLIZE too much may lead to redundancy: you may be tempted to write a few code lines to perform a task that can be easily performed by another rule, for example.

Therefore, PERLIZE should be used only in situations where other rules are not recommended or when the vendor gateway rules cannot help. Read the instructions both for the framework, vendor gateway, and gateway-adaptor rules *before* venturing too deep into Perl programming.

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 through the header fields and counter rows individually

The required parameters are:

**FILENAME\_SUFFIX** This is a suffix string that you append to the output file.  
This should reflect the operation performed by PERLIZE

The optional parameters are:

**OUTPUT\_BLOCK\_NAME**

This is the block name to use in the output data.

**OUTPUT\_DIR**

This is an alternative directory to which you write the performance LIF data.

**HEADER\_COUNTERS\_OP**

This is a subroutine to process the header block. This subroutine is passed a reference to a hash containing the header names and values. If the subroutine returns a non-zero value, then the file is discarded.

**DATA\_COUNTERS\_OP**

This is a subroutine to process each PIF data row that is passed out as a hash reference. If the subroutine returns a non-zero value, then the row will be discarded.

## 2.4.12 PIF\_2\_CSV

This rule generates an output file in the CSV format. This is important in Service Quality Manager, as it uses CSV files as source data, and there is a problem in generating the CSV file from the gateway framework. There are no required parameters. The optional parameters are:

**OUTPUT\_FORMAT** The format of the output file, which is CSV for Service Quality Manager.

**NEW\_HEADER\_COUNTERS**

Additional header counters.

**OUTPUT\_BLOCK\_NAME**

Data block to be written in the output file.

**OUTPUT\_FILENAME\_START**

Output file name prefix (not used).

**FIELD\_DELIMITER** Output field delimiters. The default is a comma (not used).

**HEADER\_FIELDS\_FOR\_OUTPUT\_FILENAME**

An array of header fields to be used as the output file name. This parameter overrides the default file name that uses the input file name as a base to construct the output file name.

**OUTPUT\_FILENAME\_DELIMITER**

Character that delimits the header fields.

**COUNTERS\_ORDER**

The order of the output fields, both from the header and data fields.

### 2.4.13 PIF\_2\_OUTPUT

The PIF\_2\_OUTPUT rule generates an output file that is not in the PIF format. There are no required parameters. The optional parameters are:

**OUTPUT\_FORMAT** The format of the output file

**NEW\_HEADER\_COUNTERS**

Additional header counters

**OUTPUT\_BLOCK\_NAME**

Data block to be written in the output file

**OUTPUT\_FILENAME\_START**

Output file name prefix

**OUTPUT\_DIR**

Output file name directory path

### 2.4.14 SPLIT\_RECORD

The SPLIT\_RECORD rule split a record based on certain counter names. It generates new counter names of the split key. The hash key of the SPLIT\_CNAMES becomes a new key counter, which retain the uniqueness of the data row. The new key counter name is specified under the NEW\_CNAMES parameter. The process is shown in Figure 2-12 on page 39.

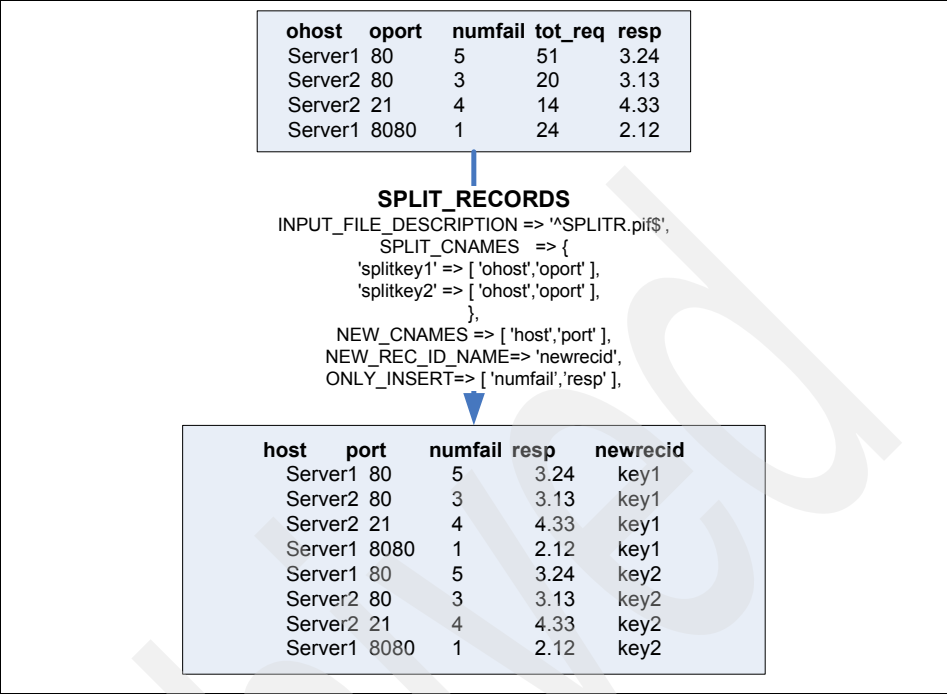


Figure 2-12 The SPLIT\_RECORD rule

The required parameters are:

- SPLIT\_CNAMES** Counter names for the split
- NEW\_CNAMES** New counter names after split

The optional parameters are:

- ONLY\_INSERT** Specifies the counter names to be inserted.
- OUTPUT\_BLOCK\_NAME** The output block name.
- OUTPUT\_FILENAME\_START** Prefix for the output file name.
- OUTPUT\_DIR** Output directory path.
- NEW\_REC\_ID\_NAME** The new counter field name for the key counter.

## 2.4.15 UNPEGGER

The UNPEGGER rule calculates the difference in values between two particular rolling counters (counters that always increase in value). The rule can handle a single time period per PIF, contained in the header, or can handle multiple periods within a PIF. For header based datetime counters, the functions are prefixed h\_. For data based datetime counters, the functions are prefixed d\_.

The required parameters are:

**INPUT\_FILE\_DATETIME\_KEY**

The time stamp to be used for performing UNPEGGER.

**PEG\_FILENAME\_PREFIX**

Prefix for the peg file name

**UNPEG\_FILENAME\_PREFIX**

Prefix for the unpeg file name

**KEEP\_RAW\_GRANULARITY**

Whether or not to keep raw granularity

**MAX\_PEG\_PIF\_AGE**

Maximum pegging age

**CALCULATE\_ROLLOVER**

Whether or not to calculate rollover

**DEFAULT\_NULL\_VALUE**

The default value to fill in when there is a calculation error

**PEG\_COUNTERS**      Counters for the peg process

**COUNTERS\_TO\_SORT\_ON**

Sorting key

**UNPEG\_FILE\_TYPE**

File type for the unpeg file

**DATETIME\_COUNTERS**

Counter for date and time

The optional parameters are:

**INPUT\_DATE\_FORMAT**

Format of the date time files

**ROLLOVER\_WINDOW**

Window for the roll over data

## 2.5 Performance tips

This section discusses some miscellaneous performance tips for the gateway processing. Some of the discussion is taken from the *Gateway Framework User's Guide*, found at:

[http://publib.boulder.ibm.com/infocenter/tivihelp/v8r1/topic/com.ibm.netcool\\_pm.doc/GatewayFramework\\_3.1\\_UserGuide.pdf](http://publib.boulder.ibm.com/infocenter/tivihelp/v8r1/topic/com.ibm.netcool_pm.doc/GatewayFramework_3.1_UserGuide.pdf)

The following are some common changes that can improve performance of the gateway. Before applying any changes, ensure that the gateway is outputting the correct data format for loading.

- ▶ 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. This is less of an issue if you are use PIF data that is cached in memory. It may also have an impact on the loadmap configuration.
- ▶ If outputting to multiple directories, use the Transfer Stage: Rather than using multiple instances of the same rule to output to two different paths, use the Transfer Engine or external scripts to distribute the files.
- ▶ Never use a low performance disk or a network mounted disk (NFS disk) for the intermediate processing files: The network and processing impact may significantly reduce the throughput of your processing.
- ▶ Use tmpfs file systems 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 file systems or configure the PIF data to be cached in memory. This can offer significant savings.
- ▶ Use parallel processing: If the server has more than one processor, configure parallel processing for rules that meet the detailed guidelines. This can also be beneficial if the disk is slow, as multiple reads/writes can be queued to the disk. The related parameters in EngineConfig.pm are:  

```
NUMBER_OF_PROCESSES => 4  
MIN_FILES_PER_PROCESS => 2
```
- ▶ Use PERLIZE for complex operations: If there is a specific counter manipulation or 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).

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

# Gateway development

This chapter describes a sample gateway development of a Motorola GGSN data gateway structure and modification of that structure. This chapter contains the following sections:

- ▶ 3.1, “Identifying the data source” on page 44
- ▶ 3.2, “Gateway configuration for Motorola GGSN” on page 51
- ▶ 3.3, “Transfer process for the Motorola GGSN gateway” on page 53
- ▶ 3.4, “Modifying the Generic CSV parser engine” on page 56
- ▶ 3.5, “Developing the post parser configuration for Motorola GGSN” on page 58
- ▶ 3.6, “Running gateway programs” on page 64

## 3.1 Identifying the data source

The standard Motorola GGSN vendor subsystem v50 release gateway parses raw performance data in the CSV file format. Refer to “A sample Motorola GGSN CSV file” on page 76 for a sample of this raw performance data file.

The installed gateway configuration produces LIF files that are meant to be used with Performance Manager for Wireless loaders. Hence, further changes to the gateway configuration is necessary in order for it to produce the CSV file format that is required by the Service Quality Manager adapters. The Service Quality Manager adapters require the input file to be a CSV file with a name conforming to the 3GPP standard for time stamp.

We discuss data source information in the following sections:

- ▶ “Input specification for GPRS Technology Pack” on page 44
- ▶ “GPRS Service Solution interface control guide” on page 47
- ▶ “Mapping of data from gateway and adapters” on page 50

### 3.1.1 Input specification for GPRS Technology Pack

To develop a gateway module, we must understand the input file format. You can get the format from the customer or the technology vendor. As an example, we are using the Motorola GGSN gateway configuration from the GPRS Technology Pack. Apart from the gateway configuration, every Technology Pack also includes a functional specification document. We extract the input file specification from the functional specification, as shown in Table 3-1.

Table 3-1 Raw data field names

Number	CSV field name	Usage
0	systemName	GGSN_ID
1	systemIPAddress	Part of APN_ID
2	index	Part of APN_ID
3	day	Header date
4	startTime	Header start time
5	endTime	Header end time
6	cgprsAccPtActivePdps	Not used
7	cgprsAccPtDestAddrViolTpdus	Not used

Number	CSV field name	Usage
8	cgprsAccPtDhcpAddrReleases	Not used
9	cgprsAccPtDhcpAddrRequests	Not used
10	cgprsAccPtDownstreamPacketCount	Not used
11	cgprsAccPtGgsnDeactivatedPdps	Not used
12	cgprsAccPtlpv6DownstreamPackets	Not used
13	cgprsAccPtlpv6DownstreamTrafficVolume	Not used
14	cgprsAccPtlpv6GgsnDeactivatedPdps	Not used
15	cgprsAccPtlpv6GgsnSuccDeactivatedPdps	Not used
16	cgprsAccPtlpv6MsActivatedDynPdps	Not used
17	cgprsAccPtlpv6MsActivatedPdps	Not used
18	cgprsAccPtlpv6MsDeactivatedPdps	Not used
19	cgprsAccPtlpv6MsSuccActivatedDynPdps	Not used
20	cgprsAccPtlpv6MsSuccActivatedPdps	Not used
21	cgprsAccPtlpv6MsSuccDeactivatedPdps	Not used
22	cgprsAccPtlpv6NetworkInitDeactPdps	Not used
23	cgprsAccPtlpv6NetworkInitDeactSuccPdps	Not used
24	cgprsAccPtlpv6UpstreamPackets	Not used
25	cgprsAccPtlpv6UpstreamTrafficVolume	Not used
26	cgprsAccPtMsActivatedDynPdps	The usage of these fields are discussed in Table 3-2 on page 46.
27	cgprsAccPtMsActivatedPdps	
28	cgprsAccPtMsDeactivatedPdps	
29	cgprsAccPtNetworkInitPdps	
30	cgprsAccPtRedirInterMobilTraffic	Not used
31	cgprsAccPtRevDownstrTrafficVol	Not used
32	cgprsAccPtRevUpstreamTrafficVol	Not used
33	cgprsAccPtSourceAddrViolTpds	Not used
34	cgprsAccPtSuccDhcpAddrRequests	Not used

Number	CSV field name	Usage
35	cgprsAccPtSuccGgsDeactivatedPdps	Not used
36	cgprsAccPtSuccMsActivatedDynPdps	The usage of these fields are discussed in Table 3-2 on page 46.
37	cgprsAccPtSuccMsActivatedPdps	
38	cgprsAccPtSuccMsDeactivatedPdps	
39	cgprsAccPtSuccNetworkInitPdps	
40	cgprsAccPtUpstreamPacketCount	Not used

Using the fields in Table 3-1 on page 44, the necessary calculation for producing KPIs are listed in Table 3-2. The field names in Table 3-2 are truncated, as we omit the prefix cgprsAccPt. Also, the data in Table 3-2 is put in the PIF data block called cgprsAccPtStatisticsEntry. Table 3-2 lists the information that must be produced from the raw file that will be processed by the Technology Pack gateway. We will use this for our gateway mapping process to define KPI attributes.

Table 3-2 Fields data related to GPRS Core GGSN for Motorola

Field name	Expression® or mapping	Description
GGSN_ID	systemName	A unique identifier for the GGSN
APN_ID	systemIPAddress&".":&index	A unique identifier for the APN
SuccessfulPDPActivation	SuccMsActivatedPdps+ SuccMsActivatedDynPdps+ SuccNetworkInitPdps	The total successful PDP activations
SuccMsActivatedPdps	SuccMsActivatedPdps	The total number of successfully completed PDP context activation procedures initiated by MS® on this APN
SuccMsActivatedDynPdps	SuccMsActivatedDynPdps	The total number of successfully completed dynamic PDP context activation procedures initiated by MS on this APN
SuccNetworkInitPdps	SuccNetworkInitPdps	The total number of successfully completed network initiated PDP context activation procedures
AttemptsPDPActivation	MsActivatedPdps+ MsActivatedDynPdps+ NetworkInitPdps	The total attempted PDP activations

Field name	Expression® or mapping	Description
MsActivatedPdps	MsActivatedPdps	The total number of PDP context activation procedures initiated by an MS on this APN
MsActivatedDynPdps	MsActivatedDynPdps	The total number of dynamic PDP context activation procedures initiated by an MS on this APN
NetworkInitPdps	NetworkInitPdps	The total number of network initiated PDP context activation procedures
SuccMsDeactivatedPdps	SuccMsDeactivatedPdps	The total number of successfully completed PDP context deactivation initiated by the MS
MsDeactivatedPdps	MsDeactivatedPdps	The total number of PDP context deactivation procedures initiated by an MS on this APN

### 3.1.2 GPRS Service Solution interface control guide

The version of Service Quality Manager GPRS Service Solution installed is V1.3. This GPRS Service Solution includes several data sources for:

- ▶ Radio Access Network (RAN) PM
- ▶ Core Gateway GPRS Support Node (GGSN) PM
- ▶ Core Serving GPRS Support Node (SGSN) PM
- ▶ End-to-End (E2E) Active Test

For each data source, there is an interface control guide in a PDF file. The interface control guide explains the input file requirements so that data can be loaded successfully. We use the GPRS Core GGSN PM Service Solution Interface Control Guide as a basis to develop and configure the gateway we have.

In conjunction with the functional specification of the technology pack, we can correlate the performance data information for the GPRS Core GGSN PM.

#### **GPRS Core GGSN PM CSV file naming convention**

The adapter in the GPRS Core GGSN PM Service Solution requires the input CSV file to be named as follows:

YYYYMMDD.hhmm-YYYYMMDD.hhmm\_UniqueID\_GGSN.csv

where YYYYMMDD.hhmm elements correspond to the file interval start time and end time, respectively.

UniqueID is an optional element that can be used to, for example, uniquely identify the GPRS Core Network. In a situation where the deployed solution has multiple mediation points, then this element is recommended.

We use a custom field in the PERLIZE rule and a modified CSV output writer to achieve this name. This customization is explained in 3.5, “Developing the post parser configuration for Motorola GGSN” on page 58. Example 3-1 show some of the file names using the desired naming convention.

*Example 3-1 GPRS Core GGSN CSV file naming convention*

---

```
A20081008.1500-20081008.1515_1_GGSN.csv
A20081008.1500-20081008.1515_2_GGSN.csv
A20081008.1515-20081008.1530_1_GGSN.csv
A20081008.1515-20081008.1530_2_GGSN.csv
```

---

### **GPRS Core GGSN PM CSV file format**

The content of the CSV file for the core GGSN PM is listed in Table 3-3. The fields should be described in the CSV header lines.

*Table 3-3 GPRS Core GGSN PM CSV file format*

Field name	Field description	Constraints
GGSNName	Textual name of GGSN	Not null, text string (up to 64 characters)
APN	The access point name	Not null, text string (up to 256 characters)
pdpCreate_successes	Number of successful PDP Contexts creation attempts	Nullable, but >=0 if present
pdpCreate_failures	Number of unsuccessful PDP Contexts creation attempts	Nullable, but >=0 if present
mslnit_pdpDelete_successes	Number of successful mobile initiated PDP context deletion attempts	Nullable, but >=0 if present
mslnit_pdpDelete_failures	Number of unsuccessful mobile initiated PDP context deletion attempts	Nullable, but >=0 if present
nwlnit_pdpDelete_successes	Number of unsuccessful network initiated PDP context deletion attempts	Nullable, but >=0 if present
nwlnit_pdpDelete_failures	Number of unsuccessful network initiated PDP context deletion attempts	Nullable, but >=0 if present

In order to generate this data, we would have to manipulate the raw performance data in “A sample Motorola GGSN CSV file” on page 76. Some of the basic changes involve:

- ▶ Filtering the data and fields that are not used
- ▶ Combining day and start/end time to get the complete time stamp, and reformatting them
- ▶ Calculating the required data
- ▶ Renaming the data field
- ▶ Sorting the data and field according to the above format

Some of these changes are available in the common post parser rules in the gateway. Manipulating the time stamp and calculating the data would need to be further developed using the PERLIZE rule.

The service solution expects the GPRS Core GGSN CSV input file to contain one row for every unique identifier of the combination:

- ▶ GGSNName
- ▶ APN

As seen in our sample raw performance data in “A sample Motorola GGSN CSV file” on page 76, the file complies with this requirement, so this is not an issue here. However, APN is a combination of more than one field, so data manipulation to combine the fields should be configured too.

### **Service Quality Manager delivery and collection mechanism**

Most of the default settings for this mechanism are used with some exceptions.

- ▶ Transfer mechanism

The gateway is able provide the data push of the GPRS Core GGSN CSV input file to where the Service Quality Manager adapter is running and we make the necessary configuration to enable this action.

- ▶ Data directory

This default location for the data directory that will be used can be found in `$WMCROOT/var/adapter/gprs_core_ggsn_pm_loader`, where `WMCROOT` is the environment variable for the root directory of Service Quality Manager and can be found in the profile for user `saserver`. Usually, it is `/app1/sa`, unless a different path is used during install.

- ▶ File interval

The file interval we are using is 15 minutes rather than the default 60 minutes.

► Transfer latency

If the CSV files that come in periodically to the data directory are not on time, there is a maximum latency period of 60 minutes. This is the default, but is configurable.

► Files per interval

The GPRS Core GGSN Service Solution expects at least one input file per interval. In our case, we have two files per interval because we have two sources of GGSN raw performance data.

### 3.1.3 Mapping of data from gateway and adapters

The information from the service solution interface control guide and the input format (gateway's functional specification) should be mapped. As we now have an idea of how the data should be presented, we could proceed with developing the gateway configuration files in 3.2, "Gateway configuration for Motorola GGSN" on page 51.

Table 3-4 shows how the mapping of the information between Table 3-2 on page 46 for the data input and Table 3-3 on page 48 from the interface control guide documentation. Information that is not available will be left as null.

*Table 3-4 Mapping of information*

Interface field name	Functional specification field name
GGSNName	GGSN_ID
APN	APN_ID
pdpCreate_successes	SuccessfulPDPActivation
pdpCreate_failures	AttemptsPDPActivation-SuccessfulPDPActivation
mslnit_pdpDelete_successes	SuccMsDeactivatedPdps
mslnit_pdpDelete_failures	MsDeactivatedPdps-SuccMsDeactivatedPdps
nwlnit_pdpDelete_successes	Not available
nwlnit_pdpDelete_failure	Not available

## 3.2 Gateway configuration for Motorola GGSN

Now that we know how to present the data based on the information gathered in the last section, we can go ahead and configure the gateway with the most basic settings, and finally develop it to generate our desired end result. Along the way, we may highlight some debugging options and customization of the product for features that may not yet be available.

Although the gateway is properly installed and set up with the Motorola GGSN vendor sub-system v50 release gateway configuration, this only provides a standard configuration platform. There are still some additional configuration steps to get it running properly before we discuss further developing it to generate our desired end result output. The configuration files for this gateway can be found in \$GATEWAY\_ROOT/config/motorola-ggsn/v50 directory.

### 3.2.1 Configuring the Gateway directory settings

First, configure the directories where the gateway will be processing the files, usually because either the path provided in the standard configuration is not correct or it has not been created yet.

This configuration should be made in the properties file. IN\_DIR, INT\_DIR, and OUT\_DIR are the compulsory directory settings to ensure that the gateway is able to run. Chapter 2, “Gateway concepts and processing” on page 13 explains how these settings are used. Example 3-2 shows the directories being used in the properties file.

*Example 3-2 The gateway directory settings in the properties file*

---

```
# Gateway processing directory settings
IN_DIR=../spool/parse
INT_DIR=../spool/inter
OUT_DIR=../spool/loader

# storage directory settings
INPUT_STORAGE_DIR=../spool/archive
INTERMEDIATE_STORAGE_DIR=0
OUTPUT_STORAGE_DIR=1
```

---

We are also using the INPUT\_STORAGE\_DIR directory setting, which is optional, to store input raw performance data files after the gateway has processed them. The other optional directory settings are INTERMEDIATE\_STORAGE\_DIR and OUTPUT\_STORAGE\_DIR. The intermediate storage is usually set to 0, which disables it.

The directory path that is used in the settings can either be a full path or an absolute path according to the location of the properties file.

### 3.2.2 Configuring auditing and logging for the gateway

Similarly for the auditing and logging of the gateway, the default configured path may not yet be available, and sometimes it is not the preferred location. Although the gateway will still run without this configuration, we cannot tell if the gateway ran successfully or has any error messages, and this is not an acceptable situation.

Normally, the standard configuration names are audit and log. If there are other vendor subsystems, however, this will often cause confusion, so we use more meaningful names. The names for auditing and logging are AUDIT\_FILE and LOG\_FILE, respectively. Example 3-3 shows how to use these names in the properties file.

*Example 3-3 The gateway auditing and logging names in the properties file*

---

```
# Log and audit files settings.
# LOG_LEVEL can be set from 0 to 5, described as follow:
# 0 - No logs
# 1 - Critical error logs, Gateway unable to run or stop running
# 2 - Major error logs, Gateway configuration issues
# 3 - Minor error logs, raw data issues
# 4 - Warning
# 5 - Info
LOG_FILE=../../logs/gways-motorola-ggsn-v50.log
LOG_LEVEL=1
AUDIT_FILE=../../logs/gways-motorola-ggsn-v50.audit
```

---

By defining the LOG\_LEVEL option, more information is available in LOG\_FILE as the number increases from 0 to 5. The default LOG\_LEVEL of 5 will be useful during the installation and configuration phase of the gateway. However, when we have finished developing the gateway and ensure that it is running well, we should lower the LOG\_LEVEL to avoid accumulating a large amount of logs that may take up disk space.

### 3.2.3 Enabling debug mode settings for the gateway

The gateway also has debug mode settings that are useful during development or troubleshooting a problem. The settings do not actually output more debug messages in the logs, but, with them enabled, the raw performance data files and

PIF files will not be removed from their directories after the gateway has completed running.

With these settings, you can trace the execution of the gateway at every step and check whether the files produced in each step are correct. You can set this function using the DEBUG option in the properties file. Example 3-4 shows how to turn on debug mode for the gateway in the properties file.

*Example 3-4 The Gateway debug mode settings in properties file*

---

```
# Debug level  
# 0 - no debugging  
# debug - debugging on  
DEBUG=debug
```

---

As when the logging level is set too high, the debug mode should be disabled when you are certain that the gateway is running properly. Leaving the debug mode enabled will fill up disk space and sometimes cause incorrect output, because if some PIF files are using the same file name, you may not be able to overwrite them. Setting this option to 0 will turn off this feature.

## 3.3 Transfer process for the Motorola GGSN gateway

The transfer process consists of two parts: the extraction or inbound transfer (see 3.3.1, “Extracting data from source” on page 53) and the delivery or outbound transfer (see 3.3.2, “Sending data to the adapter” on page 55).

### 3.3.1 Extracting data from source

The Motorola GGSN vendor subsystem V50 release’s raw performance data files consist of CSV data generated through SNMP data acquisition. Usually, this data is available in a separate system, and must be extracted and sent to the gateway in order to process these files.

To do this task, we use the transfer (in) process in the Gateway Framework architecture. The configuration is in the `TransferConfig.pm` file. Example 3-5 shows the transfer configuration for extracting the CSV file from the source by using FTP to the gateway in the `TransferConfig.pm` file.

*Example 3-5 TransferConfig.pm for inbound transfer*

---

```
{
  RULE_DESC      => 'Transfer files in',
  PROTOCOL       => "ftp",
  HOST           => "9.3.5.11",
  DIRECTION      => "IN",
  USER          => "saserver",
  PASS           => "Saserver01",
  LOCAL_DIR      => "../spool/parse",
  REMOTE_DIR     => "/code/motorola-ggsn-source",
  TIMESTAMP_FILE => "../timestamp_in",
  DEPTH          => 0,
  INPUT_FILE_DESCRIPTION => {
    CSV => '^.*\\.csv$',
  },
  RETRY_ATTEMPTS => 4,
  RETRY_INTERVAL => 5,
  TIMEOUT        => 30, # Seconds
  BULK_TRANSFER  => "True",
},
```

---

Most of the transfer configuration is quite straightforward, but take note of `DIRECTION` option, set to `IN`, which means send the file to the gateway. The `TIMESTAMP_FILE` option lets the gateway know the history of the last run of the transfer and the files that were previously picked up, so only the transfer of new files will take place. This prevents duplicate file transferal.

Here the `INPUT_FILE_DESCRIPTION` option is also introduced, which we will see more in the other configuration part. It is used to identify only the file name required for the regular expression pattern matching. It is often use with the `DEPTH` option, so it is able to search a number of levels down into the defined subdirectory. The number 0 denotes that there is only the current level of the directory.

### 3.3.2 Sending data to the adapter

After the gateway has finished processing the raw performance data files, the generated output files are sent to the gateway so they can be loaded. In our example, the destination is the adapter in the GPRS Core GGSN PM Service Solution.

Here we use the transfer (out) process in the Gateway Framework architecture, where a configuration similar to what was done for the transfer (in) process is done in the `TransferConfig.pm` file. Example 3-6 shows the transfer configuration, where the output CSV file generated by the gateway is pushed to the adapter in the `TransferConfig.pm` file.

*Example 3-6 TransferConfig.pm for the outbound transfer*

---

```
{
    RULE_DESC      => 'Transfer files out',
    PROTOCOL       => "ftp",
    HOST           => "9.3.5.11",
    DIRECTION      => "OUT",
    USER          => "saserver",
    PASS          => "Saserver01",
    LOCAL_DIR      => "../spool/loader",
    REMOTE_DIR     =>
        "/app1/sa/var/adapter/gprs_core_ggsn_pm_loader/upload",
    TIMESTAMP_FILE => "../timestamp_out",
    DEPTH          => 0,
    INPUT_FILE_DESCRIPTION => {
        CSV => '^.*\\.csv$',
    },
    RETRY_ATTEMPTS => 4,
    RETRY_INTERVAL => 5,
    TIMEOUT        => 30, # Seconds
    DELETE_ORIGINAL => "True",
},
```

---

Here the `DIRECTION` option is set to `OUT`, so the file is sent out from the gateway instead. The `TIMESTAMP_FILE` is also used, but remember to set it using a different file name than the one we used when transferring the file in, or you may have a conflict.

## 3.4 Modifying the Generic CSV parser engine

This section discusses modifying the standard generic CSV engine in two sections:

- ▶ “Verifying the parser engine configuration” on page 56
- ▶ “Modifying the configuration” on page 56

### 3.4.1 Verifying the parser engine configuration

The Motorola GGSN vendor subsystem v50 release’s gateway configuration provides a default configuration for the Generic CSV engine we previously installed. It can process the Motorola GGSN CSV format raw performance data to produce temporary PIFs.

The first level PIF file is actually the initial PIF file created by the vendor gateway engine in INT\_DIR. Normally, these files end with the `-.I.pif` suffix generated by the parser engine. The parser engine is governed by the `EngineConfig.pm` file.

The primary verification is to ensure that the `INPUT_FILE_DESCRIPTION` matches the file of raw performance data. Check for other configurations that have relevant files or file names, such as `INPUT_DIR_DEPTH`, `ORDER_OF_FILES`, and `FILENAME_HEADER_FIELDS`. Usually, these are common misconfiguration errors.

It seems that the default engine configuration already matches the raw performance data file name we are using, as shown in Example 3-7. If it does not match, then it is necessary to modify the options related to the file and its naming convention.

*Example 3-7 Sample Motorola GGSN raw performance data file name*

---

```
PassiveCollector-IBM.pmgw_cgprsAccPtStatisticsEntry_10_10_2008_14_55_23.csv
PassiveCollector-IBM.pmgw_cgprsAccPtStatisticsEntry_10_10_2008_14_59_09.csv
PassiveCollector-IBM.pmgw_cgprsAccPtStatisticsEntry_10_10_2008_15_10_23.csv
PassiveCollector-IBM.pmgw_cgprsAccPtStatisticsEntry_10_10_2008_15_14_09.csv
```

---

### 3.4.2 Modifying the configuration

Normally, changing the vendor gateway parser engine configuration is preferable. The parser engine configuration should already match each vendor subsystem and release raw data. Modifying it without proper knowledge can cause the gateway to be unable to produce PIF files.

It is better to make the configuration changes at the post parser level to manipulate data in the first level PIF files after they are generated. However, sometimes making changes to the vendor gateway engine configuration cannot be avoided. Most of the time, this is due to limitations with the common data manipulation features in the post parser configuration, and they are only available through the specific vendor gateway configuration.

In our case, there are some configuration option changes to be made:

► **HEADER\_FIELDS and DATA\_FIELDS**

Gateway configurations from the Technology Pack are designed to produce LIF output for the Performance Manager for Wireless product. However, we require CSV file output for Service Quality Manager. Header blocks and data blocks only exist in PIF and LIF files, but not in CSV files. When exporting to CSV file format, the gateway combines both header content and data block content in a single row, with the header data appearing first.

The specific format of the CSV output file names in Example 3-1 on page 48 requires some information to be put into the header, such as start and end time stamps. The systemIPAddress from the header field must be moved to the data fields. The systemIPAddress combines with index data forms APN\_ID field. The mapping for HEADER\_FIELDS and DATA\_FIELDS is shown in Example 3-8. The field's number corresponds to the number in Table 3-1 on page 44.

*Example 3-8 The EngineConfig.pm for header and data fields*

---

'HEADER_FIELDS'	=> [ 0,3,4,5 ],
'DATA_FIELDS'	=> [ 1,2,6..100 ],

---

► **Time rounding feature in HEADER\_DATA\_RECORD\_PROCESSING**

For Service Quality Manager, time stamp data must fall in the correct time boundary of the granularity period. For example, a 15 minute data granularity period should only have a time stamp at 0, 15, 30, and 45 minutes. This can be achieved by modifying the HEADER\_DATA\_RECORD\_PROCESSING subroutine in the parser engine configuration.

The original parser has time stamps in the data block, but we move it to the header. So, instead of using the \$d\_ref variables, we use the \$h\_ref variables. This change is shown in Example 3-9 for the EngineConfig.pm file.

*Example 3-9 Generic CSV engine time rounding settings in EngineConfig.pm*

---

```
$h_ref->{startTime} = timeRounding($h_ref->{startTime});  
$h_ref->{endTime} = timeRounding($h_ref->{endTime});  
  
if($h_ref->{startTime} == $h_ref->{endTime}){  
    $h_ref->{endTime} = $h_ref->{endTime} + 900;  
}
```

---

In every vendor gateway, there is specific documentation that has more details about the available configuration options. Some of the configuration options we have discussed for Generic CSV can be found in the *Generic CSV User Guide* packaged with the installation.

## 3.5 Developing the post parser configuration for Motorola GGSN

The post parser configuration is where the majority of the development takes place. We have to modify this configuration to generate a CSV output file instead of a LIF file. We want to produce a CSV file for the adapter in the GPRS Core GGSN PM service solution. This configuration will be done in the UserConfig.pm file.

Our solution is discussed in these sections:

- ▶ “Manipulating the data in the PERLIZE post parser rule” on page 58
- ▶ “Outputting data in the CSV format using the PIF\_2\_CSV rule” on page 63

### 3.5.1 Manipulating the data in the PERLIZE post parser rule

According to 3.1, “Identifying the data source” on page 44, we have an idea of how to generate the required data from the raw data, either through direct one-to-one, or many-to-one mappings. This mapping relationship, which is created by renaming existing data and calculations, does not seem to be directly available with the standard post parser rule.

We can create these mappings by using the PERLIZE rule, which provides advanced customized data manipulation in header and data blocks. By enabling debug mode, we can see the first level PIF (see “Sample first level PIF data file” on page 76 for more information).

From our previous gateway parser engine configurations, we understand some data resides in the header and data block section. The purpose of having header data is because it is common information that can be used for all other data block sections, so they would not need to be repeated within each data block section. Eventually, the combination of a header and each data block section will form one record, or row, in the CSV file format. The PERLIZE rule allows manipulation of the data in header and block sections.

We decide how to manipulate the data from the PIF file as follows:

- ▶ Input to the PERLIZE rule is defined with the INPUT\_FILE\_DESCRIPTION option, as shown in see Example 3-10. We also show how the PERLIZE rule is called by simply defining the RULE\_TYPE option. The RULE\_DESC option is mainly a description, so this rule can be easily searched for when performing auditing or logging, because sometimes the same rule type may be used more than once.

*Example 3-10 Matching the PIF file name for the PERLIZE rule in UserConfig.pm*

---

RULE_TYPE	=> 'PERLIZE',
RULE_DESC	=> 'Manipulating Motorola GGSN Data',
INPUT_FILE_DESCRIPTION	=> '^*-#-I.pif\$',

---

▶ Header manipulation

In the header, there are several changes that we want to perform:

- We would like to change the time format for startTime and endTime to the format YYYYMMDD.hhmm. Currently, startTime and endTime information only includes the time, but does not include date information, which has to be created by combining the time with the day. To combine them, a conversion to UNIX® date time is done and then converted to the desired format. The Date::Manip Perl libraries should be included to allow this date and time manipulation.
- The startTime and endTime fields use prefixes that conform to the naming convention of the output file discussed in “GPRS Core GGSN PM CSV file naming convention” on page 47.
- We want to rename the field from systemName to GGSNName. Copy the same data from the old field name to a new field name. Then the old data field, which is not necessary, is removed with the REDUNDANT\_HEADER\_COUNTERS option.

- Manipulate the end\_Time field to add the necessary hyphen and added the INDEX field to define the suffix index for the 3GPP file name format.

All these operations above are configured through the HEADER\_COUNTERS\_OP sub routine shown in Example 3-11. A basic knowledge of Perl scripting is a prerequisite to understanding this operation and is not covered in this book.

*Example 3-11 Manipulating data in the header section with the PERLIZE rule in UserConfig.pm*

---

```

HEADER_COUNTERS_OP => sub {
    use Date::Manip;
    my $h_ref=shift;

    # Manipulate date format
    $h_ref->{"day"}=UnixDate($h_ref->{"day"},"%s");
    $h_ref->{"startTime"}=$h_ref->{"day"} + $h_ref->{"startTime"};
    $h_ref->{"startTime"}=localtime($h_ref->{"startTime"});
    $h_ref->{"startTime"}=UnixDate($h_ref->{"startTime"},"%Y%m%d.%H%M");
    $h_ref->{"startTime"}="A" . $h_ref->{"startTime"};
    $h_ref->{"endTime"}=$h_ref->{"day"} + $h_ref->{"endTime"};
    $h_ref->{"endTime"}=localtime($h_ref->{"endTime"});
    $h_ref->{"endTime"}=UnixDate($h_ref->{"endTime"},"%Y%m%d.%H%M");
    $h_ref->{"endTime"}="-" . $h_ref->{"endTime"};
    $h_ref->{"INDEX"}="_1" # index field

    # Renaming counter to form GGSNNAME
    $h_ref->{"GGSNNAME"}=$h_ref->{"systemName"};

    return 0;
},

```

---

#### ► Data block

Based on the mappings listed in Table 3-2 on page 46, the calculation of the data fields are performed as follows:

- Merge the data fields systemIPAddress and index to form the APN data field. When APN is created, systemIPAddress and index can be deleted by using the REDUNDANT\_DATA\_COUNTERS option, because they are no longer required.
- Calculations are performed for pdpCreate\_successes, pdpCreate\_failures, mslnit\_pdpDelete\_successes, and mslnit\_pdpDelete\_failures.
- Null values are provided for nwlinit\_pdpDelete\_successes and nwlinit\_pdpDelete\_failures.

All these operations are configured by using the DATA\_COUNTERS\_OP option instead, as shown in Example 3-12.

*Example 3-12 Manipulating data in the block section with the PERLIZE rule in UserConfig.pm*

---

```
DATA_COUNTERS_OP => sub {
    my $c_ref=shift;

    # Merge counters to form APN
    $c_ref->{"APN"}=$c_ref->{"systemIPAddress"}.":". $c_ref->{"index"};

    # Calculate pdpCreate_successes
    $c_ref->{"PDP_CREATE_SUCCESSES"}=
        $c_ref->{"cgprsAccPtSuccMsActivatedPdps"}+
        $c_ref->{"cgprsAccPtSuccMsActivatedDynPdps"}+
        $c_ref->{"cgprsAccPtSuccNetworkInitPdps"};

    # Calculate pdpDelete_failures
    $c_ref->{"PDP_CREATE_FAILURES"}=
        $c_ref->{"cgprsAccPtMsActivatedPdps"}+
        $c_ref->{"cgprsAccPtMsActivatedDynPdps"}+
        $c_ref->{"cgprsAccPtNetworkInitPdps"}-
        $c_ref->{"cgprsAccPtSuccMsActivatedPdps"}-
        $c_ref->{"cgprsAccPtSuccMsActivatedDynPdps"}-
        $c_ref->{"cgprsAccPtSuccNetworkInitPdps"};

    # Renaming counter to form msInit_pdpDelete_successes
    $c_ref->{"MSINIT_PDPDELETE_SUCCESSES"}=
        $c_ref->{"cgprsAccPtSuccMsDeactivatedPdps"};

    # Calculate msInit_pdpDelete_failures
    $c_ref->{"MSINIT_PDPDELETE_FAILURES"}=
        $c_ref->{"cgprsAccPtMsDeactivatedPdps"}-
        $c_ref->{"cgprsAccPtSuccMsDeactivatedPdps"};

    # Creating counters nwInit_pdpDelete_*
    $c_ref->{"NWINIT_PDPDELETE_SUCCESSES"}='';
    $c_ref->{"NWINIT_PDPDELETE_FAILURES"}='';

    return 0;
},
```

---

We then remove the unused fields using the REDUNDANT\_HEADER\_COUNTERS and REDUNDANT\_DATA\_COUNTERS parameters, as shown in Example 3-13.

*Example 3-13 Unused fields*

---

```
REDUNDANT_HEADER_COUNTERS  => [qw(FILENAME BLOCKNAME day systemName)],
REDUNDANT_DATA_COUNTERS   => [ 'systemIPAddress',
    'index',
    'cgprsAccPtActivePdps',
    'cgprsAccPtDestAddrViolTpds',
    'cgprsAccPtDhcpAddrReleases',
    'cgprsAccPtDhcpAddrRequests',
    'cgprsAccPtDownstreamPacketCount',
    'cgprsAccPtGgsnDeactivatedPdps',
    'cgprsAccPtIpv6DownstreamPackets',
    'cgprsAccPtIpv6DownstreamTrafficVolume',
    'cgprsAccPtIpv6GgsnDeactivatedPdps',
    'cgprsAccPtIpv6GgsnSuccDeactivatedPdps',
    'cgprsAccPtIpv6MsActivatedDynPdps',
    'cgprsAccPtIpv6MsActivatedPdps',
    'cgprsAccPtIpv6MsDeactivatedPdps',
    'cgprsAccPtIpv6MsSuccActivatedDynPdps',
    'cgprsAccPtIpv6MsSuccActivatedPdps',
    'cgprsAccPtIpv6MsSuccDeactivatedPdps',
    'cgprsAccPtIpv6NetworkInitDeactPdps',
    'cgprsAccPtIpv6NetworkInitDeactSuccPdps',
    'cgprsAccPtIpv6UpstreamPackets',
    'cgprsAccPtIpv6UpstreamTrafficVolume',
    'cgprsAccPtMsActivatedDynPdps',
    'cgprsAccPtMsActivatedPdps',
    'cgprsAccPtMsDeactivatedPdps',
    'cgprsAccPtNetworkInitPdps',
    'cgprsAccPtRedirInterMobilTraffic',
    'cgprsAccPtRevDownstrTrafficVol',
    'cgprsAccPtRevUpstreamTrafficVol',
    'cgprsAccPtSourceAddrViolTpds',
    'cgprsAccPtSuccDhcpAddrRequests',
    'cgprsAccPtSuccGgsDeactivatedPdps',
    'cgprsAccPtSuccMsActivatedDynPdps',
    'cgprsAccPtSuccMsActivatedPdps',
    'cgprsAccPtSuccMsDeactivatedPdps',
    'cgprsAccPtSuccNetworkInitPdps',
    'cgprsAccPtUpstreamPacketCount',
    ],
```

---

### 3.5.2 Outputting data in the CSV format using the PIF\_2\_CSV rule

Here we use the PIF\_2\_CSV rule to output data in the CSV format. This option is only available with Gateway Framework Version 3.4.2.1. which can be downloaded from the support page at:

<http://www-01.ibm.com/support/docview.wss?uid=swg24021846>

The configuration of the PIF\_2\_CSV rule is shown in Example 3-14.

*Example 3-14 PIF\_2\_CSV rule in UserConfig.pm*

---

```
RULE_TYPE           => 'PIF_2_CSV',
RULE_DESC           => 'Create CSV output',
INPUT_FILE_DESCRIPTION =>
['^(PassiveCollector-IBM.pmgw_cgprsAccPtStatisticsEntry.*)-#-I-#-PZ.pif
$'],
OUTPUT_BLOCK_NAME    => 'GGSN',
HEADER_FIELDS_FOR_OUTPUT_FILENAME => [qw(startTime endTime
                                           INDEX GGSNNAME)],
REDUNDANT_HEADER_COUNTERS => [qw(startTime endTime INDEX)],
OUTPUT_FORMAT         => 'CSV_Writer_ITS0',
COUNTERS_ORDER        => [qw(GGSNNAME APN
                               PDPCREATE_SUCCESSES PDPCREATE_FAILURES
                               MSINIT_PDPDELETE_SUCCESSES MSINIT_PDPDELETE_FAILURES
                               NWINIT_PDPDELETE_SUCCESSES NWINIT_PDPDELETE_FAILURES)],
```

---

In Example 3-14, the matching PIF file name ends with the PZ suffix, which means it is from the last PERLIZE rule we have configured. As explained earlier, configure RULE\_TYPE to call the PIF\_2\_CSV rule and give a meaningful description for RULE\_DESC. The OUTPUT\_BLOCK\_NAME is not used for Service Quality Manager; we just call the output block as GGSN.

We use HEADER\_FIELDS\_FOR\_OUTPUT\_FILENAME options that allow us to set the desired file name that is needed by Service Quality Manager adapters. We do not use the delimiter, as we include the delimiters in the PERLIZE custom code. The source field names are startTime, endTime, INDEX, and GGSNNAME.

The OUTPUT\_FORMAT default is CSV\_Writer. However, it introduces opening and closing comments in the CSV file that cannot be processed by Service Quality Manager. We create a new module by copying CSV\_Writer.pm into CSV\_Writer\_ITSO.pm and put them in the configuration path. The original CSV\_Writer.pm Perl modules reside in \$GATEWAY\_ROOT/gateway-framework/perl\_extensions/lib/site\_perl/5.6.1. We copy that directory to our configuration directory at \$GATEWAY\_ROOT/config/motorola-ggsn/v50/CSV\_Writer\_ITSO.pm. The modified CSV\_Writer.pm perl module is provided in “CSV\_Writer\_ITSO.pm” on page 78.

## 3.6 Running gateway programs

To start running the gateway, use the script gateway\_start.sh in the vstart directory in Gateway Framework. Provide the relevant vendor subsystem and release argument to this script (found in the Gateway Configuration directory) in order to launch the correct one. An example invocation is:

```
GATEWAY_ROOT/gateway-framework/vstart/gateway_start.sh -vendor  
motorola-ggsn -release v50
```

This launches the Motorola GGSN vendor subsystem v50 release gateway. However, passing different arguments launches other gateway configurations if they are already set up.

**Note:** Ensure that the PERL5\_BASE environment variable is set up with the path used when building Perl, or the gateway cannot run.

The gateway program must be invoked from a scheduler or a cron job. The gateway processes any file data that exists at that point. It does not go to sleep to wait for the next iteration. A separate program must iterate and execute the gateway\_start.sh script. A sample walk through of the various stages of the gateway is provided in the following sections:

- ▶ 3.6.1, “Transferring file inbound” on page 65
- ▶ 3.6.2, “Processing the CSV file” on page 67
- ▶ 3.6.3, “Running the PERLIZE stage” on page 69
- ▶ 3.6.4, “Running the PIF\_2\_CSV stage” on page 71
- ▶ 3.6.5, “Transferring file outbound” on page 72

### 3.6.1 Transferring file inbound

The transfer inbound is performed using ftp. The configuration file is provided in Example 3-5 on page 54. The audit log contains information about the files that are transferred, Example 3-15 contains the audit log (we strip out the time stamp for clarity).

*Example 3-15 The inbound file transfer audit log*

---

```
PIF module in use PIF_Handler
Start Transfer Engine: IN
Processing Transfer rule Transfer files in
TransferEngine::Ftp: Getting files
TransferEngine::Ftp File transferred
/code/motorola-ggsn-source/PassiveCollector-IBM.pmgw_cgprsAccPtStatisti
csEntry_13_01_2009_11_47_42.csv
Finished Transfer rule result 1
Finished Transfer Engine: IN
```

---

The processing log message is listed in Example 3-16. We use the most detailed level of detail (LOGLEVEL=5). We also strip out the time stamp for clarity.

*Example 3-16 The inbound file transfer message log*

---

```
5 lock_process
(..//spool/parse,/gways/gateway-framework/parsersrc/gateway_main.pm)
A PIF module in use PIF_Handler
3 Loader dir size utility not configured
A Start Transfer Engine: IN
A Processing Transfer rule Transfer files in
5 TransferEngine: PROTOCOL is ftp
4 TransferEngine: Protocol is ftp
5 Checking optional TIMEOUT which can be type SCALAR
5 Checking optional OVERWRITE_FILES which can be type SCALAR
5 Checking optional TMP_FILENAME_EXTENSION which can be type SCALAR
5 Checking optional PING_RETRY_ATTEMPTS which can be type SCALAR
5 Checking optional RETRY_INTERVAL which can be type SCALAR
5 Checking optional PING_PROTOCOL which can be type SCALAR
5 Returning 0 errors from configuration check
A TransferEngine::Ftp: Getting files
5 Time stamp file ../timestamp_in doesn't exist
5 TransferEngine::Ftp: set transfer mode to binary
5 Transfer - created connection to 9.3.5.11
5 Transfer files in /code/motorola-ggsn-source since
5 Slept for 0 seconds
5 dir_ent:
```

```

-rwx----- 1 0      0      1787 Jan 13 17:49
PassiveCollector-IBM.pmgw_cgprsAccPtStatisticsEntry_13_01_2009_11_47_42
.csv
1 TransferEngine::Ftp: Getting files and modification times
5 Processing file
PassiveCollector-IBM.pmgw_cgprsAccPtStatisticsEntry_13_01_2009_11_47_42
.csv
5 File
PassiveCollector-IBM.pmgw_cgprsAccPtStatisticsEntry_13_01_2009_11_47_42
.csv mod time 1231868953 size 1787
5 Getting the files now...
5 Matched file
PassiveCollector-IBM.pmgw_cgprsAccPtStatisticsEntry_13_01_2009_11_47_42
.csv to type CSV
4 TransferEngine::Ftp Starting copy
PassiveCollector-IBM.pmgw_cgprsAccPtStatisticsEntry_13_01_2009_11_47_42
.csv
A TransferEngine::Ftp File transfered
/mnt/hgfs/src/PassiveCollector-IBM.pmgw_cgprsAccPtStatisticsEntry_13_01
_2009_11_47_42.csv
5 Setting new modtime 1231868953 for /code/motorola-ggsn-source
5 Writing out new modification time
A Finished Transfer rule result 1
4 TransferEngine:: DIRECTION IN - copied 1 files
A Finished Transfer Engine: IN

```

---

The file transfer stores the file time stamp information in a file. It is used to avoid reprocessing the file. For the inbound transfer, the file is called `.timestamp_in`, as specified in the rule definition. Example 3-17 shows the `.timestamp_in` file with one entry.

*Example 3-17 The `.timestamp_in` file*

---

```

$mod_time = {
  '/code/motorola-ggsn-source' => {
    'PREV_FILES' => {},
    'LAST_TIME' => '1231868953',
    'COPIED_FILES' => {

      'PassiveCollector-IBM.pmgw_cgprsAccPtStatisticsEntry_13_01_2009_11_47_4
      2.csv' => '1787'
    }
  }
};

```

---

### 3.6.2 Processing the CSV file

The parser engine, controlled by the `Engine_config.pm`, is responsible for converting the input CSV file into a PIF file for further processing. The `Engine_config.pm` file that we use is discussed in 3.4, “Modifying the Generic CSV parser engine” on page 56.

The parser engine process is also recorded in audit and log message files. Example 3-18 shows the entries in the audit files without the time stamp.

*Example 3-18 The parser engine audit file*

---

```
Start Parser Engine
Processing rule 'Motorola GGSN v50', type 'GENERIC_CSV_GATEWAY'
Processing files in '../spool/parse'
Engine process_files: 1 process(es) 1 files total 1
Processing
'PassiveCollector-IBM.pmgw_cgprsAccPtStatisticsEntry_13_01_2009_11_47_4
2.csv'
    Generic CSV Interface: Created output file
    '../spool/inter/PassiveCollector-IBM.pmgw_cgprsAccPtStatisticsEntry_13_
01_2009_11_47_42.csv-#-I.pif'
Finished
'PassiveCollector-IBM.pmgw_cgprsAccPtStatisticsEntry_13_01_2009_11_47_4
2.csv'
Engine: Processed 1 files
Engine: processed 1 files in total
Finish Parser Engine
```

---

Example 3-19 shows the messages generated by processing the CSV file into a PIF file using the Generic CSV processing. The Generic CSV processing capability is provided by the vendor gateway.

*Example 3-19 The parser engine log messages*

---

```
4 GenUtils: changing PIF handling mode to PIF_Handler
A Start Parser Engine
A Processing rule 'Motorola GGSN v50', type 'GENERIC_CSV_GATEWAY'
5 Checking optional DATA_FIELDS which can be type ARRAY
5 Checking optional INVALID_VALUE_MATCH which can be type ARRAY
5 Checking optional DEFAULT_NULL_VALUE which can be type SCALAR
5 Checking optional WHITESPACE_MODIFIER which can be type SCALAR
5 Returning 0 errors from configuration check
A Processing files in '../spool/parse'
5 Sorting files in ascending order
5 Processing files:
```

---

```

../spool/parse/PassiveCollector-IBM.pmgw_cgprsAccPtStatisticsEntry_13_0
1_2009_11_47_42.csv
5 GenUtils: calc_nprocs() no configuration for parallel processing or
too few files (1) return 1 processor
A Engine process_files: 1 process(es) 1 files total 1
5 Engine: total files processed so far 0 allowed
5 Engine raw file
../spool/parse/PassiveCollector-IBM.pmgw_cgprsAccPtStatisticsEntry_13_0
1_2009_11_47_42.csv
5 Request to create recovery file for rule GENERIC_CSV_GATEWAY
5 Recovery file: Input key
GENERIC_CSV_GATEWAYPassiveCollector-IBM.pmgw_cgprsAccPtStatisticsEntry_
13_01_2009_11_47_42.csv hashed to -1948647783
4 Storing current file,
PassiveCollector-IBM.pmgw_cgprsAccPtStatisticsEntry_13_01_2009_11_47_42
.csv key -1948647783 file
4 Extracting FILENAME_HEADER_FIELDS
A Processing
'PassiveCollector-IBM.pmgw_cgprsAccPtStatisticsEntry_13_01_2009_11_47_4
2.csv'
5 GenUtils: Opening file
PassiveCollector-IBM.pmgw_cgprsAccPtStatisticsEntry_13_01_2009_11_47_42
.csv
4 Bad characters rule entry found:
5 Found Header Line =
systemName,systemIPAddress,index,day,startTime,endTime,cgprsAccPtActive
Pdps,cgprsAccPtDestAddrViolTpdus,cgprsAccPtDhcpAddrReleases,cgprsAccPtD
hcpAddrRequests,cgprsAccPtDownstreamPacketCount,cgprsAccPtGgsnDeactivat
edPdps,cgprsAccPtIpv6DownstreamPackets,cgprsAccPtIpv6DownstreamTrafficV
olume,cgprsAccPtIpv6GgsnDeactivatedPdps,cgprsAccPtIpv6GgsnSuccDeactivat
edPdps,cgprsAccPtIpv6MsActivatedDynPdps,cgprsAccPtIpv6MsActivatedPdps,c
gprsAccPtIpv6MsDeactivatedPdps,cgprsAccPtIpv6MsSuccActivatedDynPdps,cgpr
sAccPtIpv6MsSuccActivatedPdps,cgprsAccPtIpv6MsSuccDeactivatedPdps,cgpr
sAccPtIpv6NetworkInitDeactPdps,cgprsAccPtIpv6NetworkInitDeactSuccPdps,c
gprsAccPtIpv6UpstreamPackets,cgprsAccPtIpv6UpstreamTrafficVolume,cgprsA
ccPtMsActivatedDynPdps,cgprsAccPtMsActivatedPdps,cgprsAccPtMsDeactivate
dPdps,cgprsAccPtNetworkInitPdps,cgprsAccPtRedirInterMobilTraffic,cgprsA
ccPtRevDownstrTrafficVol,cgprsAccPtRevUpstreamTrafficVol,cgprsAccPtSour
ceAddrViolTpdus,cgprsAccPtSuccDhcpAddrRequests,cgprsAccPtSuccGgsDeactiv
atedPdps,cgprsAccPtSuccMsActivatedDynPdps,cgprsAccPtSuccMsActivatedPdps
,cgprsAccPtSuccMsDeactivatedPdps,cgprsAccPtSuccNetworkInitPdps,cgprsAcc
PtUpstreamPacketCount
A Generic CSV Interface: Created output file
'../spool/inter/PassiveCollector-IBM.pmgw_cgprsAccPtStatisticsEntry_13_
01_2009_11_47_42.csv-#-I.pif'

```

```
4 PIF_Handler closing output file
../spool/inter/PassiveCollector-IBM.pmgw_cgprsAccPtStatisticsEntry_13_0
1_2009_11_47_42.csv-#-I.pif
A Finished
'PassiveCollector-IBM.pmgw_cgprsAccPtStatisticsEntry_13_01_2009_11_47_4
2.csv'
3 Warning : Statistics obj not instantiated.
5 Clearing recovery file for key -1948647783
4 Deleted recovery file for
PassiveCollector-IBM.pmgw_cgprsAccPtStatisticsEntry_13_01_2009_11_47_42
.csv from input directory
A Engine: Processed 1 files
4 All 1 Files processed
A Engine: processed 1 files in total
A Finish Parser Engine
4 Parse Count Utility: Is not required
4 GenUtils: changing PIF handling mode to PIF_Handler
```

---

### 3.6.3 Running the PERLIZE stage

The PERLIZE stage is the most versatile stage, as you can run arbitrary code to process the header and data of a PIF file. It is also the most complex stage, as it requires a deep understanding of both Perl and gateway framework architecture.

Example 3-20 shows the audit file for the PERLIZE stage. It is quite simple, as the PERLIZE does not have a lot of built-in processing. The embedded code must perform additional logging or auditing if necessary using `AudMess` or `LogMess` methods.

*Example 3-20 The PERLIZE audit messages*

---

```
Start Post Parser
Processing rule 'Rename APN', type 'PERLIZE'
Processing
'../spool/inter/PassiveCollector-IBM.pmgw_cgprsAccPtStatisticsEntry_13_
01_2009_11_47_42.csv-#-I.pif'
```

---

Example 3-21 shows the log messages for the PERLIZE stage. We perform a great deal of processing, as indicated by the rule listings in 3.5.1, “Manipulating the data in the PERLIZE post parser rule” on page 58.

*Example 3-21 The PERLIZE log messages*

---

```
A Start Post Parser
5 PIF_Handler: Get_Files() Getting list of files from intermediate
  directory ../spool/inter
A Processing rule 'Rename APN', type 'PERLIZE'
5 Checking optional REDUNDANT_HEADER_COUNTERS which can be type ARRAY
5 Checking optional REDUNDANT_DATA_COUNTERS which can be type ARRAY
5 Checking optional DATA_COUNTERS_OP which can be type CODE
5 Checking optional HEADER_COUNTERS_OP which can be type CODE
5 Returning 0 errors from configuration check
5 GenUtils: calc_nprocs() no configuration for parallel processing or
  too few files (1) return 1 processor
1 GenUtils: process_pif_files() nprocs 1 files per process 1
A Processing
'../spool/inter/PassiveCollector-IBM.pmgw_cgprsAccPtStatisticsEntry_13_
01_2009_11_47_42.csv-#-I.pif'
4 PIF_Handler opening file
  ../spool/inter/PassiveCollector-IBM.pmgw_cgprsAccPtStatisticsEntry_13_0
  1_2009_11_47_42.csv-#-I.pif
4 PERLIZE - calling out to header counters operation
3 PostParser: DATA COUNTER 'cgprsAccPtDestAddrViolTpds' does not exist
3 PostParser: DATA COUNTER 'cgprsAccPtDhcpAddrReleases' does not exist
3 PostParser: DATA COUNTER 'cgprsAccPtDhcpAddrRequests' does not exist

  . . . << messages for redundant_data_counters >>

5 Sorted data counters
APN,PDP_CREATE_SUCCESSES,PDP_CREATE_FAILURES,MSINIT_PDPDELETE_SUCCESSES,M
SINIT_PDPDELETE_FAILURES,NWINIT_PDPDELETE_SUCCESSES,NWINIT_PDPDELETE_FA
ILURES
3 PostParser: DATA COUNTER 'cgprsAccPtActivePdps' does not exist
3 PostParser: DATA COUNTER 'cgprsAccPtDestAddrViolTpds' does not exist
3 PostParser: DATA COUNTER 'cgprsAccPtDhcpAddrReleases' does not exist

  . . . << messages for redundant_data_counters >>

4 PIF_Handler closing output file
  ../spool/inter/PassiveCollector-IBM.pmgw_cgprsAccPtStatisticsEntry_13_0
  1_2009_11_47_42.csv-#-I-#-PZ.pif
```

```
4 PIF_Handler closing input file
../spool/inter/PassiveCollector-IBM.pmgw_cgprsAccPtStatisticsEntry_13_0
1_2009_11_47_42.csv-#-I-#-PZ.pif
5 PIF_Handler: Get_Files() Getting list of files from intermediate
directory ../spool/inter
```

---

### 3.6.4 Running the PIF\_2\_CSV stage

The new PIF\_2\_CSV stage is provided to bolster the CSV generation for Service Quality Manager gateways. This allows custom file name generation, including a 3GPP compliant file name. The rule parameters are provided in 3.5.2, “Outputting data in the CSV format using the PIF\_2\_CSV rule” on page 63. Example 3-22 shows the audit messages for this stage.

*Example 3-22 PIF\_2\_CSV audit messages*

---

```
Processing rule 'Create CSV output', type 'PIF_2_CSV'
Processing
'../spool/inter/PassiveCollector-IBM.pmgw_cgprsAccPtStatisticsEntry_13_
01_2009_11_47_42.csv-#-I-#-PZ.pif'
Output file: 'A20090114.1330-20090114.1345_1_gsl-ggsn-3.pt'
Output file: 'A20090114.1330-20090114.1345_1_gsl-ggsn-3.pt'
Finish Post Parser
```

---

Example 3-23 lists the log messages for this stage. As indicated, the output file name is compliant to the 3GPP format.

*Example 3-23 PIF\_2\_CSV log messages*

---

```
A Processing rule 'Create CSV output', type 'PIF_2_CSV'
5 Checking optional OUTPUT_FILENAME_START which can be type SCALAR
5 Checking optional COUNTERS_ORDER which can be type ARRAY
5 Checking optional HEADER_FIELDS_FOR_OUTPUT_FILENAME which can be type
ARRAY
5 Checking optional OUTPUT_BLOCK_NAME which can be type SCALAR
5 Returning 0 errors from configuration check
5 GenUtils: calc_nprocs() no configuration for parallel processing or
too few files (1) return 1 processor
1 GenUtils: process_pif_files() nprocs 1 files per process 1
A Processing
'../spool/inter/PassiveCollector-IBM.pmgw_cgprsAccPtStatisticsEntry_13_
01_2009_11_47_42.csv-#-I-#-PZ.pif'
4 PIF_Handler opening file
../spool/inter/PassiveCollector-IBM.pmgw_cgprsAccPtStatisticsEntry_13_0
1_2009_11_47_42.csv-#-I-#-PZ.pif
```

```

A Output file: 'A20090114.1330-20090114.1345_1_gsl-ggsn-3.pt'
A Output file: 'A20090114.1330-20090114.1345_1_gsl-ggsn-3.pt'
5 Sorted data counters
GGSNNAME,APN,PDP_CREATE_SUCCESSES,PDP_CREATE_FAILURES,MSINIT_PDPDELETE_SU
CCESSES,MSINIT_PDPDELETE_FAILURES,NWINIT_PDPDELETE_SUCCESSES,NWINIT_PDP
DELETE_FAILURES
4 PIF_Handler closing output file
../spool/inter/A20090114.1330-20090114.1345_1_gsl-ggsn-3.pif
4 Renaming temp file to final name
../spool/loader/A20090114.1330-20090114.1345_1_gsl-ggsn-3.csv
4 PIF_Handler closing input file
../spool/inter/A20090114.1330-20090114.1345_1_gsl-ggsn-3.pif
5 PIF_Handler: Get_Files() Getting list of files from intermediate
directory ../spool/inter
A Finish Post Parser

```

---

The files being processed are stored in the spool subdirectory. The files that we have in the spool directory are listed in Example 3-24.

*Example 3-24 Listing of files in the spool directory*

---

```

[root@rtmach2 spool]# ls inter loader parse
inter:
A20090114.1330-20090114.1345_1_gsl-ggsn-3.pif
PassiveCollector-IBM.pmgw_cgprsAccPtStatisticsEntry_13_01_2009_11_47_42.csv-#-I.pif
PassiveCollector-IBM.pmgw_cgprsAccPtStatisticsEntry_13_01_2009_11_47_42.csv-#-I-#-PZ.pif

loader:
A20090114.1330-20090114.1345_1_gsl-ggsn-3.csv

parse:
PassiveCollector-IBM.pmgw_cgprsAccPtStatisticsEntry_13_01_2009_11_47_42.csv

```

---

### 3.6.5 Transferring file outbound

After the files are produced, the generated CSV file can be transferred for processing by the Service Quality Manager loader or service solution. If the Service Quality Manager application server is not the same as the gateway machine, the file must be transferred out using a file transfer mechanism. We use ftp, as shown in the configuration rule in 3.3.2, “Sending data to the adapter” on page 55. The audit and messages are very similar to the inbound transfer. Example 3-25 on page 73 shows the audit messages.

*Example 3-25 Outbound transfer audit file*

---

```
Start Transfer Engine: OUT
Processing Transfer rule Transfer files out
TransferEngine::Ftp: Putting files to remote host
Transfer::Engine:Ftp Copying:
A20090114.1330-20090114.1345_1_gs1-ggsn-3.csv
Transfer::Engine:Ftp Copied:
A20090114.1330-20090114.1345_1_gs1-ggsn-3.csv
Finished Transfer rule result 1
Finished Transfer Engine: OUT
```

---

Example 3-26 shows the log messages file.

*Example 3-26 Outbound transfer message file*

---

```
A Start Transfer Engine: OUT
A Processing Transfer rule Transfer files out
5 TransferEngine: PROTOCOL is ftp
4 TransferEngine: Protocol is ftp
5 Checking optional TIMEOUT which can be type SCALAR
5 Checking optional OVERWRITE_FILES which can be type SCALAR
5 Checking optional TMP_FILENAME_EXTENSION which can be type SCALAR
5 Checking optional PING_RETRY_ATTEMPTS which can be type SCALAR
5 Checking optional RETRY_INTERVAL which can be type SCALAR
5 Checking optional DELETE_ORIGINAL which can be type SCALAR
5 Checking optional PING_PROTOCOL which can be type SCALAR
5 Returning 0 errors from configuration check
3 GenUtils: check_dir(), checking directory
/gways/config/motorola-ggsn/spool/loader
5 Time stamp file ../timestamp_out doesn't exist
A TransferEngine::Ftp: Putting files to remote host
5 TransferEngine::Ftp: set transfer mode to binary
5 Transfer - created connection to 9.3.5.11
5 Getting files from /gways/config/motorola-ggsn/spool/loader to copy
to 9.3.5.11
5 Found 3 entries in directory
5 Matched file A20090114.1330-20090114.1345_1_gs1-ggsn-3.csv to type
CSV
A Transfer::Engine:Ftp Copying:
A20090114.1330-20090114.1345_1_gs1-ggsn-3.csv
A Transfer::Engine:Ftp Copied:
A20090114.1330-20090114.1345_1_gs1-ggsn-3.csv
5 TransferEngine::Ftp:ftp put deleting original
A20090114.1330-20090114.1345_1_gs1-ggsn-3.csv
5 Writing out new modification time
```

```
A Finished Transfer rule result 1
4 TransferEngine:: DIRECTION OUT - copied 1 files
A Finished Transfer Engine: OUT
5 unlock_process(..\spool\parse\.lock_gateway_main.pm.pid)
1 Destroying property service
```

---

Example 3-27 shows the .timestamp\_out file.

*Example 3-27 File .timestamp\_out*

---

```
$mod_time = {
  'PREV_FILES' => undef,
  'LAST_TIME' => 1231881135,
  'COPIED_FILES' => {
    'A20090114.1330-20090114.1345_1_gs1-ggsn-3.csv' => '0.3525390625'
  }
};
```

---

## Sample data and programs listing for gateways

This appendix provides a sample listing of data and control files for the gateway and service solution development. The files in this appendix can be downloaded from the ITSO Web site. See Appendix B, “Additional material” on page 95 for more details. The contents of this appendix are included in the following sections:

- ▶ “A sample Motorola GGSN CSV file” on page 76
- ▶ “Sample first level PIF data file” on page 76
- ▶ “Sample output CSV file” on page 78
- ▶ “CSV\_Writer\_ITSO.pm” on page 78
- ▶ “Motorola GGSN EngineConfig.pm” on page 85
- ▶ “Motorola GGSN UserConfig.pm” on page 88
- ▶ “Sample UserConfig.pm” on page 92

## A sample Motorola GGSN CSV file

Example A-1 lists a sample Motorola GGSN CSV file.

*Example: A-1 Sample GGSN raw data*

---

```
systemName,systemIPAddress,index,day,startTime,endTime,cgprsAccPtActive
Pdps,cgprsAccPtDestAddrViolTpds,cgprsAccPtDhcpAddrReleases,cgprsAccPtD
hcpAddrRequests,cgprsAccPtDownstreamPacketCount,cgprsAccPtGgsnDeactivat
edPdps,cgprsAccPtIpv6DownstreamPackets,cgprsAccPtIpv6DownstreamTrafficV
olume,cgprsAccPtIpv6GgsnDeactivatedPdps,cgprsAccPtIpv6GgsnSuccDeactivat
edPdps,cgprsAccPtIpv6MsActivatedDynPdps,cgprsAccPtIpv6MsActivatedPdps,c
gprsAccPtIpv6MsDeactivatedPdps,cgprsAccPtIpv6MsSuccActivatedDynPdps,cgp
rsAccPtIpv6MsSuccActivatedPdps,cgprsAccPtIpv6MsSuccDeactivatedPdps,cgpr
sAccPtIpv6NetworkInitDeactPdps,cgprsAccPtIpv6NetworkInitDeactSuccPdps,c
gprsAccPtIpv6UpstreamPackets,cgprsAccPtIpv6UpstreamTrafficVolume,cgprsA
ccPtMsActivatedDynPdps,cgprsAccPtMsActivatedPdps,cgprsAccPtMsDeactivate
dPdps,cgprsAccPtNetworkInitPdps,cgprsAccPtRedirInterMobilTraffic,cgprsA
ccPtRevDownstrTrafficVol,cgprsAccPtRevUpstreamTrafficVol,cgprsAccPtSour
ceAddrViolTpds,cgprsAccPtSuccDhcpAddrRequests,cgprsAccPtSuccGgsDeactiv
atedPdps,cgprsAccPtSuccMsActivatedDynPdps,cgprsAccPtSuccMsActivatedPdps
,cgprsAccPtSuccMsDeactivatedPdps,cgprsAccPtSuccNetworkInitPdps,cgprsAcc
PtUpstreamPacketCount
gs1-ggsn-3,212.47.101.248,3,07Oct2008,56420,57320,1826,0,0,0,5268959,14
7,,,,,,,,,,,,,762,762,636,0,0,4543225140,703822395,4012,0,148,763,763
,635,0,4923866
gs1-ggsn-3,212.47.101.248,1000,07Oct2008,56420,57320,0,0,0,0,0,0,,,,,
,,,,,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
gs1-ggsn-3,212.47.101.248,46,07Oct2008,56420,57320,0,0,0,0,0,0,,,,,
,,,,,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
gs1-ggsn-3,212.47.101.248,1,07Oct2008,56420,57320,0,0,0,0,0,0,,,,,
,,,,,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
gs1-ggsn-3,212.47.101.248,253,07Oct2008,56420,57320,66,0,0,0,63526,9,,
,,,,,34,34,31,0,0,39535903,15369164,0,0,9,35,35,31,0,93194
```

---

## Sample first level PIF data file

A sample first level PIF is shown in Example A-2 on page 77.

*Example: A-2 Sample first level PIF*

---

```
## Parser Intermediate File
##START|HEADER
day|startTime|BLOCKNAME|endTime|FILENAME|systemName
14Oct2008|28800|cgprsAccPtStatisticsEntry|29700|PassiveCollector-IBM.pm
gw_cgprsAccPtStatisticsEntry_14_10_2008_08_25_23.csv|gs1-ggsn-3
##END|HEADER
##START|cgprsAccPtStatisticsEntry
|cgprsAccPtDownstreamPacketCount|cgprsAccPtIpv6MsSuccActivatedPdps|cgpr
sAccPtUpstreamPacketCount|cgprsAccPtDestAddrViolTpds|cgprsAccPtDhcpAddr
rReleases|cgprsAccPtIpv6NetworkInitDeactSuccPdps|cgprsAccPtSourceAddrVi
olTpds|cgprsAccPtGgsnDeactivatedPdps|cgprsAccPtNetworkInitPdps|cgprsAc
cPtRevUpstreamTrafficVol|cgprsAccPtIpv6DownstreamTrafficVolume|cgprsAcc
PtSuccDhcpAddrRequests|cgprsAccPtIpv6UpstreamPackets|cgprsAccPtIpv6MsSu
ccActivatedDynPdps|cgprsAccPtActivePdps|cgprsAccPtMsActivatedDynPdps|cg
prsAccPtMsActivatedPdps|cgprsAccPtIpv6MsActivatedDynPdps|cgprsAccPtIpv6
GgsnSuccDeactivatedPdps|cgprsAccPtSuccMsActivatedDynPdps|index|cgprsAcc
PtSuccGgsDeactivatedPdps|cgprsAccPtRedirInterMobilTraffic|cgprsAccPtIpv
6UpstreamTrafficVolume|cgprsAccPtIpv6MsSuccDeactivatedPdps|cgprsAccPtSu
ccNetworkInitPdps|cgprsAccPtDhcpAddrRequests|cgprsAccPtSuccMsDeactivate
dPdps|cgprsAccPtIpv6MsActivatedPdps|cgprsAccPtIpv6DownstreamPackets|cgpr
sAccPtIpv6GgsnDeactivatedPdps|cgprsAccPtRevDownstrTrafficVol|cgprsAccP
tIpv6MsDeactivatedPdps|cgprsAccPtSuccMsActivatedPdps|cgprsAccPtMsDeacti
vatedPdps|cgprsAccPtIpv6NetworkInitDeactPdps|systemIPAddress
212.47.101.248|1613949||1592838|0|0||1813|71|0|192505726||0||803|402|4
02||401|3|71|0||0|0|289|||1548659852||401|289||212.47.101.248
212.47.101.248|0||0|0|0||0|0|0|0||0||0|0|0||0|1000|0|0||0|0|0|||0||
0|0||212.47.101.248
212.47.101.248|0||0|0|0||0|0|0|0||0||0|0|0||0|46|0|0||0|0|0|||0|0|
0||212.47.101.248
212.47.101.248|0||0|0|0||0|0|0|0||0||0|0|0||0|1|0|0||0|0|0|||0|0|0
||212.47.101.248
212.47.101.248|95498||91918|0|0||0|3|0|4860942||0||47|21|21||21|253|3
0||0|0|15|||105111146||21|15||212.47.101.248
##END|cgprsAccPtStatisticsEntry
```

---

## Sample output CSV file

The sample output CSV file from the gateway conforms to the 3GPP naming convention A20081010.1500-20081010.1515\_1\_GGSN.csv. Example A-3 shows the sample content of the CSV file.

*Example: A-3 Sample output CSV file*

---

```
GGSNNAME,APN,PDP_CREATE_SUCCESSES,PDP_CREATE_FAILURES,MSINIT_PDPDELETE_SUCCESSES,
MSINIT_PDPDELETE_FAILURES,NWINIT_PDPDELETE_SUCCESSES,NWINIT_PDPDELETE_FAILURES
gs1-ggsn-3,212.47.101.248:46,0,0,0,0,,
gs1-ggsn-3,212.47.101.248:3,1661,2,700,0,,
gs1-ggsn-3,212.47.101.248:1000,0,0,0,0,,
gs1-ggsn-3,212.47.101.248:253,56,0,19,0,,
gs1-ggsn-3,212.47.101.248:1,0,0,0,0,,
```

---

## CSV\_Writer\_ITSO.pm

As noted in 3.5, “Developing the post parser configuration for Motorola GGSN” on page 58, the CSV\_Writer.pm must be modified to suppress the comments at the beginning and ending of the output file. The modified CSV\_Writer\_ITSO.pm is shown in Example A-4.

*Example: A-4 Modified CSV\_Writer.pm*

---

```
#
#-----
# %Z% %M% %I% %G% %Z%
#-----
#
# File: CSV_Writer_ITSO.pm
#
# Desc: this package contains the fucntions that are used to write the
# information out in Comma Separated Value (CSV).
#
# (c) Copyright IBM Corporation 1998, 2008. All rights reserved.
#
package CSV_Writer_ITSO;

use strict;
use vars qw($VERSION @ISA @EXPORT @EXPORT_OK);

require Exporter;
require AutoLoader;
```

```

@ISA = qw(Exporter AutoLoader);
# Items to export into callers namespace by default. Note: do not
export
# names by default without a very good reason. Use EXPORT_OK instead.
# Do not simply export all your public functions/methods/constants.
@EXPORT = qw( );
$VERSION = '0.02';

use GenUtils;
use RecordsUtils;
use FileHandle;
use AudLog;

#####
#
# Sub: New
#
# Desc: Creates the output object instance
#
# Params: None
#
# Returns: The created object
#####
sub New {
    my $proto = shift;
    my $class = ref($proto) || $proto;
    my $self = {@_};

    # create records object when record entries exists
    if ( exists($self->{OUTPUT_RECORD_KEY}) ) {
        $self->{records_obj} = RecordsUtils->get_records_object(
            OUTPUT_RECORD_KEY => $self->{OUTPUT_RECORD_KEY},
            OUTPUT_RECORD_KEY_DELIMITER =>
        $self->{OUTPUT_RECORD_KEY_DELIMITER},
        );
    }

    $self->{filename} = undef;
    $self->{filehandle} = undef;
    $self->{store} = "";

    bless ($self, $class);
    return $self;
}

```

```
#####
#
# Sub: Open
#
# Desc: Open a output file
#
# Params: $file - the name of the output file
#
# Returns: Nothing
#####
sub Open($) {
    my $self = shift;
    my $filename = shift;

    # temp filename?
    if ($filename =~ /\.pt$/){
        $self->{filename} = $self->{tmp_file} = $filename;
        $self->{filename} =~ s/(\.pt)$/\.csv/;
    }
    else {
        $self->{filename} = $self->{tmp_file} = $filename;
        $self->{tmp_file} .= ".pt";
    }

    if( -e $self->{filename} ) {
        LogMess("CSV_Writer: WARNING: $self->{filename} already exists,
overwriting", 3);
    }

    $self->{tmp_file} = $self->{filename} . ".pt";

    $self->{filehandle} = new FileHandle;
    $self->{filehandle}->open("> $self->{tmp_file}")
        or LogMess("Can't open $self->{filename} for output", 1);

    # Write header lines to the file.
    # $self->{store} .= "# CSV Output File Start\n";
}

#####
#
# Sub: Open_Block
#
# Desc: Open a block in the output file. In the case of CSV
```

```

#         output nothing happens
#
# Params: $name - name of the block to open
#
# Returns: Nothing
#
#####
sub Open_Block() {
}

#####
#
# Sub: WriteToFile
#
# Desc: Writes the cached output to file
#
# Params: none;
#
# Returns: Nothing
#
#####
sub WriteToFile() {
    my $self = shift;

    $self->{filehandle}->print( $self->{store} );
    $self->{store} = "";
    return;
}

#####
#
# Sub: Write_Comment
#
# Desc: Writes out a comment to the output file
#
# Params: $comment - the comment to write
#
# Returns: nothing
#
#####
sub Write_Comment() {
    (shift)->{store} .= "# " . (shift) . "\n";
    return;
}

```

```
#####
#
# Sub: Block_Info
#
# Desc: This function prints all the elements of a hash to the
#       output string. The reference is supplied as an argument.
#       It checks if the value of the hash element has any length.
#       If there is no length then nothing is printed.
#
# Params: $href - reference to a counter=value hash
#         $counter_order - optional array specifying counter output
#         order
#
# Returns: Nothing
#####
sub Block_Info() {
    my ($self, $href, $counter_order) = @_;

    # save the header data as it's output per line
    if (defined($counter_order) && scalar(@$counter_order)){
        @{$self->{HEADER_NAMES}} = @$counter_order;
        @{$self->{HEADER_VALUES}} = map { (defined $href->{$_}) ?
$href->{$_} :
                                $self->{DEFAULT_NULL_VALUE}
        } @$counter_order;
    }
    else {
        @{$self->{HEADER_NAMES}} = keys %$href;
        @{$self->{HEADER_VALUES}} = values %$href;
    }
    $self->{CREATE_BLOCK} = undef;
    return;
}

#####
#
# Sub: Create_Block
#
# Desc: This function opens, writes the information and closes
#       the block.
#
# Params: $block - the name of the block
#         $cref - reference to counter=value hash
#         $counter_order - order of the data counters (optional)
#

```

```

# Returns: nothing
#
#####
sub Create_Block() {
    my( $self, $block, $cref, $counter_order ) = @_ ;

    my @block = ();# stores the header/counter info temporarily
    my $data;# stores the header/counter info data

    if ( defined($self->{records_obj}) &&
exists($self->{OUTPUT_RECORD_KEY}) ) {
        $self->{records_obj}->add_record_key(\ $cref, \ $counter_order) ;
    }

    unless ($self->{CREATE_BLOCK}){
        # write out the header and data counter names
        @{$self->{DATA_COUNTERS}} = defined($counter_order)
            && scalar(@$counter_order) ? @$counter_order : keys %$cref;
        $self->{store} .= join(",",@{$self->{HEADER_NAMES}}),
@{$self->{DATA_COUNTERS}});
        $self->{store} .= "\n";
        $self->{CREATE_BLOCK} = 1;
    }

    # write out the counter values
    $self->{store} .= join(",",@{$self->{HEADER_VALUES}}),
        map { (defined $cref->{$_}) ? $cref->{$_} :
            $self->{DEFAULT_NULL_VALUE} }
@{$self->{DATA_COUNTERS}});
    $self->{store} .= "\n";

    return;
}

#####
#
# Sub: Close
#
# Desc: Closes off the open blocks in the file and writes out
# the stored data to disk
#
# Params: Nothing
#
# Returns: Nothing
#

```

```
#####
sub Close() {
    my $self = shift;

    # Write end lines to the file.
    #$self->{store} .= "# CSV Output File End\n";

    $self->WriteToFile();

    $self->{filehandle}->close;
    LogMess("Renaming temp file to final name $self->{filename}",4);
    # rename from temporary name to final
    rename($self->{tmp_file},$self->{filename});
    return $self->{filename};
}

#####
#
# Sub: Close_Block
#
# Desc: Closes a block. Require as a dummy function
#
# Params: Nothing
#
# Returns: Nothing
#
#####
sub Close_Block() {
}

1;
__END__
# Below is the stub of documentation for your module. You better edit
it!

=head1 NAME

CSV_Writer - a Perl 5 object that produces a CSV Output
File.

=head1 SYNOPSIS

    use CSV_Writer;

    $objid = CSV_Writer->New();
```

```

$objjid->Open($filename);
$objjid->Open_Block( $blockname );
$objjid->Close_Block();
$objjid->WriteToFile();
$objjid->Write_Comment( "Comment string" );
$objjid->Block_Info( \%counter_hash );
$objjid->Create_Block( $blockname, \%counter_hash );
$objjid->Close();

```

=head1 DESCRIPTION

This perl object includes a set of subroutines that enable the easy production of a Comma Separated Value format (CSV) file, perhaps for output to a binner.

=head1 AUTHOR

Gateways Team, KL Engineering Centre

=head1 SEE ALSO

perl(1).

=cut

---

## Motorola GGSN EngineConfig.pm

The complete listing of the EngineConfig.pm that is used in 3.4, “Modifying the Generic CSV parser engine” on page 56 is shown in Example A-5.

*Example: A-5 EngineConfig.pm for motorola-ggsn gateway*

---

```

#-----
# %A%
#-----
#
# This perl module contains the configuration information for this
# parser. If you don't understand the format of this configuration file
# DON'T TOUCH IT.
#
# Ericsson GGSN R3
#

```

```

# Copyright (C) 2007 Vallent Software Systems UK, an IBM Company.
# All rights reserved.

package EngineConfig;

@ISA = qw(Exporter);
@EXPORT = qw(engine_config);

use strict;
use Exporter ();
use Data::Dumper;

use Date::Calc qw(Decode_Month Add_Delta_YMDHMS);

#-----
# Modify/Add rules here
#-----
my @rules =
(
    {
        'RULE_TYPE'           => 'GENERIC_CSV_GATEWAY',
        'RULE_DESC'           => 'Motorola GGSN v50',
        'INPUT_FILE_DESCRIPTION' => [ '^PassiveCollector.*$' ],
        'INPUT_DIR_DEPTH'     => 0,
        'ORDER_OF_FILES'      => 'FILENAME_ASCENDING',
        'FIELD_SEPARATOR'     => ',',
        'HEADER_FIELDS'       => [ 0,3,4,5 ],
        'DATA_FIELDS'         => [ 1,2,6..100 ],
        'INVALID_VALUE_MATCH' => [],
        'WHITESPACE_MODIFIER' => '_',
        'FIELDS_TO_KEY_PIF_FILENAME' => [],
        'DEFAULT_NULL_VALUE'  => '',
        'UPPERCASE_COUNTERS'  => 0,
        'GLOBAL_SUBSTITUTION' => {
            #      systemName => 'GGSNNAME',
            #      cgprsAccPtSuccMsDeactivatedPdps =>
            'MSINIT_PDPDELETE_SUCCESSES',
        },
        'FILENAME_HEADER_FIELDS' => {
            FILENAME           => '(.*)',
            BLOCKNAME          =>
                'PassiveCollector-\\w+\\. [A-Za-z\\.]* _([A-Za-z]*)_[0-9]{2}_[0-9]{2}_[0-9]{4}_[0-9]{2}_[0-9]{2}_[0-9]{2}\\.csv',
        },
        HEADER_DATA_RECORD_PROCESSING => sub

```

```

{
    my ($blk_name_ref, $h_ref, $d_ref) = @_;
    my $cname_new;
    my $cname_ori;

    # Set block name
    $$blk_name_ref = $h_ref->{BLOCKNAME};
    # print "raw data=>".Dumper($d_ref)."\n";

    $h_ref->{startTime} = timeRounding($h_ref->{startTime});
    $h_ref->{endTime} = timeRounding($h_ref->{endTime});
    if($h_ref->{startTime} == $h_ref->{endTime}){
        $h_ref->{endTime}= $h_ref->{endTime} + 900;
    }

    ## TESTING PURPOSES:
    foreach my $cname_ori (keys %{$d_ref}){
        $d_ref->{$cname_new} = $d_ref->{$cname_ori};
    }

    sub timeRounding(){
        my $time = shift;
        my $min = $time%3600;
        my $hr = ($time - $min)/3600;
        my $freq = 900;
        my $nearestTime = $time % $freq;

        if($nearestTime !=0 ){ ## check if it is already in
the "normalized" time standards
            $time = $time - $nearestTime;
        }

        ### DEBUG PURPOSES ONLY
        $min = $time %3600;
        $hr = ($time - $min)/3600;

        return $time;
    }
    return 0;
}
},
);

sub engine_config {

```

```

        return \@rules;
    }

    1;

```

---

## Motorola GGSN UserConfig.pm

The complete listing of the UserConfig.pm that is used in 3.5, “Developing the post parser configuration for Motorola GGSN” on page 58 is shown in Example A-6.

*Example: A-6 UserConfig.pm for motorola-ggsn gateway*

---

```

#
#-----
# %A%
#-----
#
# This perl module contains the configuration information for this
# parser. If you don't understand the format of this configuration file
# DON'T TOUCH IT.
#
#
# Copyright (C) 2007 Vallent Software Systems UK, an IBM company.
# All rights reserved.
#
package UserConfig;

@ISA = qw(Exporter);
@EXPORT = qw(postparser_config);

use strict;
use Exporter ();

#-----

my @rules = (
{
    RULE_TYPE          => 'PERLIZE',
    RULE_DESC          => 'Rename APN',
    INPUT_FILE_DESCRIPTION => '^*.pif$',
    HEADER_COUNTERS_OP => sub {
        use Date::Manip;

```

```

my $h_ref = shift;

# Manipulate date format
$h_ref->{"day"} = UnixDate($h_ref->{"day"},"%s");
$h_ref->{"startTime"} = $h_ref->{"day"} + $h_ref->{"startTime"};
$h_ref->{"startTime"} = localtime($h_ref->{"startTime"});
$h_ref->{"startTime"} =
UnixDate($h_ref->{"startTime"},"%Y%m%d.%H%M");
$h_ref->{"startTime"} = "A".$h_ref->{"startTime"};
$h_ref->{"endTime"} = $h_ref->{"day"} + $h_ref->{"endTime"};
$h_ref->{"endTime"} = localtime($h_ref->{"endTime"});
$h_ref->{"endTime"} = UnixDate($h_ref->{"endTime"},"%Y%m%d.%H%M");
$h_ref->{"endTime"} = "-".$h_ref->{"endTime"};
$h_ref->{"INDEX"} = "_1_";
# Renaming counter to form GGSNNAME
$h_ref->{"GGSNNAME"} = $h_ref->{"systemName"};
return 0;
},
    DATA_COUNTERS_OP                => sub {
my $c_ref = shift;
# Merge counters to form APN
$c_ref->{"APN"} =
$c_ref->{"systemIPAddress"}.".". $c_ref->{"index"};
# Calculate pdpCreate_successes
$c_ref->{"PDPCREATE_SUCCESSES"} =
    $c_ref->{"cgprsAccPtSuccMsActivatedPdps"} +
    $c_ref->{"cgprsAccPtSuccMsActivatedDynPdps"} +
    $c_ref->{"cgprsAccPtSuccNetworkInitPdps"};
# Calculate pdpDelete_failures
$c_ref->{"PDPCREATE_FAILURES"} =
    abs($c_ref->{"cgprsAccPtMsActivatedPdps"} +
    $c_ref->{"cgprsAccPtMsActivatedDynPdps"} +
    $c_ref->{"cgprsAccPtNetworkInitPdps"} -
    $c_ref->{"cgprsAccPtSuccMsActivatedPdps"} -
    $c_ref->{"cgprsAccPtSuccMsActivatedDynPdps"} -
    $c_ref->{"cgprsAccPtSuccNetworkInitPdps"});
# Renaming counter to form msInit_pdpDelete_successes
$c_ref->{"MSINIT_PDPDELETE_SUCCESSES"} =
    $c_ref->{"cgprsAccPtSuccMsDeactivatedPdps"};
# Calculate msInit_pdpDelete_failures
$c_ref->{"MSINIT_PDPDELETE_FAILURES"} =
    abs($c_ref->{"cgprsAccPtMsDeactivatedPdps"} -
    $c_ref->{"cgprsAccPtSuccMsDeactivatedPdps"});
# Renaming counter to form msInit_pdpDelete_successes
$c_ref->{"NWINIT_PDPDELETE_SUCCESSES"} = '';

```

```

$c_ref->{"NWINIT_PDPDELETE_FAILURES"} = '';
return 0;
},
    FILENAME_SUFFIX           => 'PZ',
    REDUNDANT_HEADER_COUNTERS => [qw(FILENAME BLOCKNAME day
systemName)],
    REDUNDANT_DATA_COUNTERS   => [
        '',
        'systemIPAddress',
        'index',
        'cgprsAccPtActivePdps',
        'cgprsAccPtDestAddrViolTpds',
        'cgprsAccPtDhcpAddrReleases',
        'cgprsAccPtDhcpAddrRequests',
        'cgprsAccPtDownstreamPacketCount',
        'cgprsAccPtGgsnDeactivatedPdps',
        'cgprsAccPtIpv6DownstreamPackets',
        'cgprsAccPtIpv6DownstreamTrafficVolume',
        'cgprsAccPtIpv6GgsnDeactivatedPdps',
        'cgprsAccPtIpv6GgsnSuccDeactivatedPdps',
        'cgprsAccPtIpv6MsActivatedDynPdps',
        'cgprsAccPtIpv6MsActivatedPdps',
        'cgprsAccPtIpv6MsDeactivatedPdps',
        'cgprsAccPtIpv6MsSuccActivatedDynPdps',
        'cgprsAccPtIpv6MsSuccActivatedPdps',
        'cgprsAccPtIpv6MsSuccDeactivatedPdps',
        'cgprsAccPtIpv6NetworkInitDeactPdps',
        'cgprsAccPtIpv6NetworkInitDeactSuccPdps',
        'cgprsAccPtIpv6UpstreamPackets',
        'cgprsAccPtIpv6UpstreamTrafficVolume',
        'cgprsAccPtMsActivatedDynPdps',
        'cgprsAccPtMsActivatedPdps',
        'cgprsAccPtMsDeactivatedPdps',
        'cgprsAccPtNetworkInitPdps',
        'cgprsAccPtRedirInterMobilTraffic',
        'cgprsAccPtRevDownstrTrafficVol',
        'cgprsAccPtRevUpstreamTrafficVol',
        'cgprsAccPtSourceAddrViolTpds',
        'cgprsAccPtSuccDhcpAddrRequests',
        'cgprsAccPtSuccGgsDeactivatedPdps',
        'cgprsAccPtSuccMsActivatedDynPdps',
        'cgprsAccPtSuccMsActivatedPdps',
        'cgprsAccPtSuccMsDeactivatedPdps',
        'cgprsAccPtSuccNetworkInitPdps',
        'cgprsAccPtUpstreamPacketCount',
    ]

```

```

DATA_COUNTERS_ORDER      ],
                          => [qw(APN
                                PDPCREATE_SUCCESSES
                                PDPCREATE_FAILURES
                                MSINIT_PDPDELETE_SUCCESSES
                                MSINIT_PDPDELETE_FAILURES
                                NWINIT_PDPDELETE_SUCCESSES
                                NWINIT_PDPDELETE_FAILURES)],
},
{
    RULE_TYPE              => 'PIF_2_CSV',
    RULE_DESC              => 'Create CSV output',
    INPUT_FILE_DESCRIPTION =>
['^(PassiveCollector-IBM.pmgw_cgprsAccPtStatisticsEntry.*)-#-I-#-PZ.pif
$'],
    OUTPUT_BLOCK_NAME      => 'GGSN',
    DEFAULT_NULL_VALUE     => '',
    PRODUCE_PIF            => 1,
    OUTPUT_FILENAME_START  => 'A',
    HEADER_FIELDS_FOR_OUTPUT_FILENAME => [qw(startTime endTime
SUFFIX GGSNNAME)],
    REDUNDANT_HEADER_COUNTERS => [qw(startTime endTime PREFIX
SUFFIX)],
    OUTPUT_FORMAT          => 'CSV_Writer_ITS0',
    COUNTERS_ORDER         => [qw(GGSNNAME APN
                                PDPCREATE_SUCCESSES
                                PDPCREATE_FAILURES
                                MSINIT_PDPDELETE_SUCCESSES
                                MSINIT_PDPDELETE_FAILURES
                                NWINIT_PDPDELETE_SUCCESSES
                                NWINIT_PDPDELETE_FAILURES)],
},
#-----
);
#-----

sub postparser_config {
    return \@rules;
}

1;

```

---

## Sample UserConfig.pm

The sample user post parser configuration that is used in the demonstration examples in 2.4, “Working with post parser rules” on page 23 is shown in Example A-7. This UserConfig.pm does not have any relation to customizing the Motorola GGSN gateway. This is provided for reference for understanding some features of the post parser rules.

*Example: A-7 Sample UserConfig.pm for testing rules*

---

```
#
package UserConfig;
@ISA = qw(Exporter);
@EXPORT = qw(postparser_config);
use strict;
use Exporter ();
my @rules = (
{
    RULE_TYPE          => 'ACCUMULATE',
    RULE_DESC          => 'Rename APN',
    INPUT_FILE_DESCRIPTION => '^ACCUMULATE.pif$',
    COUNTERS_TO_SORT_ON => [ 'Hostname' ],
    OUTPUT_BLOCK_NAME  => 'Accumulate',
    NON_ADDITIVE_COUNTERS => [ "fail" ],
    MAXIMUM_COUNTERS   => [ "fail" ],
    AVERAGE_COUNTERS  => [ "fail" ],
    MINIMUM_COUNTERS   => [ "fail" ],
    MAXIMUM_APPEND_STR => "MAX",
    MINIMUM_APPEND_STR => "MIN",
    AVERAGE_APPEND_STR => "AVG",
},
{
    RULE_TYPE          => 'AGGREGATE_LINE',
    INPUT_FILE_DESCRIPTION => '^AGGLINE.pif$',
    COUNTER_GROUPS      => { 'pscc' => 'pscc_(\d+)' },
    DEFAULT_NULL_VALUE  => 0,
    OUTPUT_BLOCK_NAME  => 'Aggline',
    REDUNDANT_DATA_COUNTERS => [ "pscc_1","pscc_2","pscc_3" ],
},
{
    RULE_TYPE          => 'CVAL_MANIP',
    CNAME_MANIP        => 'Result',
    MATCH              => [ 'OPER\s(\w+)', '([A-Za-z]+)(\w+)', '(\w+)' ],
    PATTERN             => [ '$1','$1','$1' ],
    INPUT_FILE_DESCRIPTION => '^CVAL.pif$'
```

```

},
{
  RULE_TYPE      => 'DATA_LINE_WHERE',
  COUNTER_NAMES  => [ {
    'COUNTER_NAME'=>'Result',
    'KEEP_WHERE'=>[ '\s+' ],
  }, ],
  ADD_NEW_COUNTER => 'testadd',
  PRODUCE_PIF    => "true",
  NEW_COUNTER_VALUE => 'abc',
  INPUT_FILE_DESCRIPTION => '^DATAWHERE.pif$',
},
{
  RULE_TYPE => 'FILE_SPLIT',
  INPUT_FILE_DESCRIPTION => '^FSPLIT.pif$',
  SPLIT_COUNTERS_ORDER  => [ 'Hostname' ],
  COUNTERS_USED_TO_SPLIT_FILE => { 'Hostname'=>'(.+)' },
  OLD_COUNTER_NAMES     => [ 'pscc' ],
  NEW_COUNTER_NAMES     => [ 'new_pscc' ],
  PRODUCE_PIF           => "true",
},
{
  RULE_TYPE => 'FILE_SPLIT_BY_COUNTERS',
  INPUT_FILE_DESCRIPTION => '^FSPLITC.pif$',
  SPLIT_CNAMES => {
    'split1' => [ 'Hostname','Result','pdur' ],
    'split2' => [ 'Hostname','pscc','pfile' ],
  },
  PRODUCE_PIF => "true",
},
{
  RULE_TYPE=> 'INFOINSERT',
  INPUT_FILE_DESCRIPTION => '^INFOINS.pif$',
  INFO_FILE_DESCRIPTION=> '^LOOKUP.pif$',
  NAMES_USED_TO_ID_DATA_RECORD => [ 'Hostname' ],
  NAMES_USED_TO_ID_INFORMATION => [ 'Server' ],
  ONLY_INSERT => [ 'Region' ],
  PRODUCE_PIF => "true",
  HEADER_NAMES_USED_TO_ID_DATA_RECORD => [],
  HEADER_NAMES_USED_TO_ID_INFORMATION => [],
},
{
  RULE_TYPE => 'JOIN',
  INPUT_FILE_DESCRIPTION => ['^JOIN1.pif$', '^JOIN2.pif$'],
  COUNTERS_TO_JOIN_ON=> [ 'Hostname','Port' ],

```

```

        OUTPUT_BLOCK_NAME => 'joindata',
        OUTPUT_FILENAME_START => 'JOINOUT',
        PRODUCE_PIF=> "true",
    },
    {
        RULE_TYPE => 'MERGE_RECORDS',
        INPUT_FILE_DESCRIPTION => '^MREC.pif$',
        COUNTERS_TO_SORT_ON=> [ 'Hostname' ],
        GROUP_KEY => 'numfail',
        PRODUCE_PIF=> "true",
    },
    {
        RULE_TYPE => 'SPLIT_RECORDS',
        INPUT_FILE_DESCRIPTION => '^SPLITR.pif$',
        SPLIT_CNAMES => {
            'splitkey1' => [ 'ohost','oport' ],
            'splitkey2' => [ 'ohost','oport' ],
        },
        NEW_CNAMES => [ 'host','port' ],
        NEW_REC_ID_NAME => 'newrecid',
        ONLY_INSERT => [ 'pscc','pfile' ],
    },
    #{
    # RULE_TYPE => '',
    # INPUT_FILE_DESCRIPTION => '^..pif$',
    #},
);

sub postparser_config {
    return \@rules;
}

1;

```

---

## Additional material

This book refers to additional material that can be downloaded from the Internet as described below.

### Locating the Web material

The Web material associated with this book is available in softcopy on the Internet from the IBM Redbooks Web server. Point your Web browser at:

<ftp://www.redbooks.ibm.com/redbooks/SG247689>

Alternatively, you can go to the IBM Redbooks Web site at:

[ibm.com/redbooks](http://ibm.com/redbooks)

Select the **Additional materials** and open the directory that corresponds with the IBM Redbooks form number, SG247689.

## Using the Web material

The additional Web material that accompanies this book includes the following files:

<i>File name</i>	<i>Description</i>
<b>7689perl.tar</b>	Gateway samples and listing

## System requirements for downloading the Web material

The following system configuration is recommended:

**Operating system:** AIX®, Linux, or Sun™ Solaris™

**Other requirements:** Refer to the IBM Tivoli Netcool Service Quality Manager gateway requirements

## How to use the Web material

Create a subdirectory (folder) on your workstation, and untar the contents of the Web material file into this folder.

# Abbreviations and acronyms

<b>3GPP</b>	3rd Generation Partnership Project	<b>OMC</b>	Operation Management Console
<b>AIX</b>	Advanced Interactive eXecutive	<b>OSS</b>	Operation Support System
<b>APAR</b>	Authorized Program Analysis Report	<b>PDF</b>	Portable Document Format
<b>APN</b>	Access Point Name	<b>PDP</b>	Packet Data Protocol
<b>ASCII</b>	American Standard Code for Information Interchange	<b>PIF</b>	Parser Intermediate File
<b>ASN1</b>	Abstract Syntax Notation 1	<b>RAN</b>	Radio Access Network
<b>BSS</b>	Business Support Systems	<b>RCP</b>	Remote Copy
<b>CEM</b>	Customer Experience Manager	<b>ROI</b>	Return on Investment
<b>CSV</b>	Comma Separated Values	<b>SCP</b>	Secure Copy
<b>DSL</b>	Digital Subscriber Line	<b>SFTP</b>	Secure FTP
<b>E2E</b>	End to End	<b>SGSN</b>	Serving GPRS Support Node
<b>ETL</b>	Extract, Transform, Load	<b>SLA</b>	Service Level Agreement
<b>FTP</b>	File Transfer Protocol	<b>SNMP</b>	Simple Network Management Protocol
<b>GGSN</b>	Gateway GPRS Support Node	<b>SOA</b>	Service Oriented Architecture
<b>GPRS</b>	General Packet Radio Service	<b>SQM</b>	Service Quality Management
<b>GSM</b>	Global System for Mobile communications	<b>URL</b>	Uniform Resource Locator
<b>IBM</b>	International Business Machines Corp.	<b>XML</b>	eXtensible Markup Language
<b>ITIL</b>	Information Technology Infrastructure Library		
<b>ITSO</b>	International Technical Support Organization		
<b>KPI</b>	Key Performance Indicator		
<b>KQI</b>	Key Quality Indicator		
<b>LIF</b>	Loader Input Format		
<b>NSS</b>	Network and Switching Subsystem		



# Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this book.

## IBM Redbooks

For information about ordering these publications, see “How to get Redbooks” on page 100. Note that some of the documents referenced here may be available in softcopy only.

- ▶ *End-to-End Planning for Availability and Performance Monitoring*, REDP-4371

## Other publications

These publications are also relevant as further information sources:

- ▶ *IBM Tivoli Netcool Service Quality Manager Business Objects Server Installation*, GC23-6597
- ▶ *IBM Tivoli Netcool Service Quality Manager Server Installation Guide*, GC23-6598
- ▶ *IBM Tivoli Netcool Service Quality Manager Client Installation Guide*, GC23-6599
- ▶ *IBM Tivoli Netcool Service Quality Manager AIX Server Installation Guide*, GC23-9485
- ▶ *IBM Tivoli Netcool Service Quality Manager Customer Experience Management Guide*, GC23-9489
- ▶ *IBM Tivoli Netcool Service Quality Manager Release Notes*, GI11-8203

## Online resources

These Web sites are also relevant as further information sources:

- ▶ Tivoli Netcool Service Quality Manager support pages  
<http://www-01.ibm.com/support/docview.wss?uid=swg24021846>  
<http://www-01.ibm.com/support/docview.wss?uid=swg27011991>
- ▶ IBM Tivoli Performance Manager for Wireless Web sites  
<http://www-01.ibm.com/software/tivoli/products/netcool-performance-mgr-wireless/>  
<http://www-01.ibm.com/software/tivoli/products/netcool-performance-mgr-wireless-techpacks/>

## How to get Redbooks

You can search for, view, or download Redbooks, Redpapers, Technotes, draft publications and Additional materials, as well as order hardcopy Redbooks, at this Web site:

[ibm.com/redbooks](http://ibm.com/redbooks)

## Help from IBM

IBM Support and downloads

[ibm.com/support](http://ibm.com/support)

IBM Global Services

[ibm.com/services](http://ibm.com/services)

# Index

## A

ADD\_NEW\_COUNTER 29  
AVERAGE\_APPEND\_STR 25  
AVERAGE\_COUNTERS 25

## B

BULK\_TRANSFER 22

## C

CALCULATE\_ROLLOVER 40  
CEM 2  
CNAME\_MANIP 27  
commands  
    cpio 22  
    gateway\_start.sh 16, 64  
    ping 21–22  
    rcp 22  
    ssh 22  
COUNTER\_GROUPS 26  
COUNTER\_NAME\_TO\_EXTRACT 26  
COUNTER\_NAMES 28  
COUNTER\_NULL\_VALUE 24  
COUNTERS\_ORDER 38  
COUNTERS\_TO\_JOIN\_ON 34  
COUNTERS\_TO\_SORT\_ON 24, 36, 40  
COUNTERS\_USED\_TO\_SPLIT\_FILE 30  
cpio command 22  
customer experience management, see CEM

## D

data mediation 6  
DATA\_COUNTERS\_OP 37  
DATA\_COUNTERS\_ORDER 23  
DATETIME\_COUNTERS 40  
DEFAULT\_NULL\_VALUE 26, 40  
DELETE\_ORIGINAL 21–22  
DEPTH 21–22  
Digital Subscriber Line, see DSL  
DIRECTION 20–21  
DSL 2

## E

ENABLE\_LOCAL\_COMPRESSION 21–22  
ENABLE\_PING 21–22  
ENABLE\_REMOTE\_COMPRESSION 22  
ETL 7  
extraction, transformation and loading, see ETL

## F

FIELD\_DELIMITER 37  
FILENAME\_ADDITION 29  
FILENAME\_HEADER\_FIELDS 56  
FILENAME\_SUFFIX 37

## G

Gateway GPRS Support Node, see GGSN  
gateway\_start.sh command 16, 64  
General Packet Radio Services, see GPRS  
GGSN 47  
Global System for Mobile communication, see GSM  
GPRS 2  
GROUP\_KEY 36  
GSM 2

## H

HEADER\_COUNTERS\_OP 37  
HEADER\_COUNTERS\_ORDER 23  
HEADER\_COUNTERS\_TO\_USE\_IN\_OUTPUT\_FILENAME 34  
HEADER\_FIELDS\_FOR\_OUTPUT\_FILENAME 37  
HEADER\_NAMES\_USED\_TO\_ID\_DATA\_RECORD 32  
HEADER\_NAMES\_USED\_TO\_ID\_INFORMATION 33  
HOST 20–21  
HOURS\_TO\_WAIT\_FOR\_PARTNER\_FILES 34

## I

INFO\_FILE\_DESCRIPTION 32  
INFO\_FILES\_STORAGE\_DIR 33  
INPUT\_DATE\_FORMAT 40  
INPUT\_DIR\_DEPTH 56

INPUT\_FILE\_DATETIME\_KEY 40  
INPUT\_FILE\_DESCRIPTION 21–22, 56

## K

KEEP\_RAW\_GRANULARITY 40  
key performance indicator, see KPI  
key quality indicator, see KQI  
KPI 3  
KQI 3

## L

LOCAL\_DIR 20, 22

## M

MANIP\_ONLY 36  
MATCH 27  
MAX\_PEG\_PIF\_AGE 40  
MAXIMUM\_APPEND\_STR 25  
MAXIMUM\_COUNTERS 25  
MINIMUM\_APPEND\_STR 25  
MINIMUM\_COUNTERS 25

## N

NAMES\_USED\_TO\_ID\_DATA\_RECORD 32  
NAMES\_USED\_TO\_ID\_INFORMATION 33  
NEW\_CNAMES 39  
NEW\_COUNTER\_NAMES 30  
NEW\_COUNTERS\_VALUE 29  
NEW\_HEADER\_COUNTERS 35, 37–38  
NEW\_REC\_ID\_NAME 39  
NON\_ADDITIVE\_COUNTERS 24  
NON\_MATCH\_RECORD 28  
NUMBER\_OF\_FILES\_TO\_PROCESS 21–22

## O

OLD\_COUNTER\_NAMES 30  
ONLY\_INSERT 31, 33, 39  
Operational Support System, see OSS  
ORDER\_OF\_FILES 56  
OSS 10  
OUTPUT\_BLOCK\_NAME 24, 26–27, 29, 33–34, 37–39  
OUTPUT\_DIR 28–29, 33, 35, 37–39  
OUTPUT\_FILENAME\_DELIMITER 38  
OUTPUT\_FILENAME\_START 33–34, 37–39  
OUTPUT\_FORMAT 21–22, 37–38  
OVERWRITE\_FILES 21–22

## P

Parser Intermediate File, see PIF  
PASS 21  
PATTERN 27  
PEG\_COUNTERS 40  
PEG\_FILENAME\_PREFIX 40  
PIF 12  
ping command 21–22  
PING\_PROTOCOL 21–22  
PING\_RETRY\_ATTEMPTS 21–22  
PROTOCOL\_PATH 22

## Q

QUOTE\_INPUT\_PIFS 35

## R

rcp command 22  
Redbooks Web site 100  
    Contact us xi  
REDUNDANT\_DATA\_COUNTERS 23  
REDUNDANT\_HEADER\_COUNTERS 23  
REMOTE\_DIR 20, 22  
REMOVE\_INFO\_FILES 33  
RETRY\_ATTEMPTS 22  
RETRY\_INTERVAL 21–22  
ROLLOVER\_WINDOW 40

## S

Service Level Agreement, see SLA  
service quality management, see SQM  
Serving GPRS Support Node, see SGSN  
SGSN 47  
SLA 4  
SPLIT\_CNAMES 31, 39  
SPLIT\_COUNTERS\_ORDER 30  
SQM 2  
ssh command 22  
STATISTICS\_FILENAME 21–22

## T

TIME\_JOINFILE\_PRODUCED 35  
TIMEOUT 21–22  
TIMESTAMP\_FILE 21–22  
TMP\_FILENAME\_EXTENSION 21–22  
TRANSFER\_MODE 21

## **U**

UNPEG\_FILE\_TYPE 40

UNPEG\_FILENAME\_PREFIX 40

USER 21

## **W**

WRITE\_DATA\_LINE 31, 33

Archived











# IBM Tivoli Netcool Service Quality Manager Data Mediation Gateway Development

## Service Quality Manager concepts and overview

## Data mediation in Service Quality Manager

## Development of your own interface

This IBM Redbooks publication discusses IBM Tivoli Netcool Service Quality Manager V4.1.1 data mediation development. It is aimed at implementers and system integrators connecting new data sources that will be managed by Service Quality Manager.

Data mediation in Service Quality Manager is performed by Perl gateways. Gateways extract data from vendor specific hardware monitors and prepare the data to be loaded into the Service Quality Manager application.

The development aspect of data mediation for gateways uses Perl extensively. The control files or rules are written as Perl subroutines. We discuss the detailed specifications and processing of these mediation rules.

## INTERNATIONAL TECHNICAL SUPPORT ORGANIZATION

## BUILDING TECHNICAL INFORMATION BASED ON PRACTICAL EXPERIENCE

IBM Redbooks are developed by the IBM International Technical Support Organization. Experts from IBM, Customers and Partners from around the world create timely technical information based on realistic scenarios. Specific recommendations are provided to help you implement IT solutions more effectively in your environment.

**For more information:**  
[ibm.com/redbooks](http://ibm.com/redbooks)