3GPP XML Gateway User Guide

Release: 3.4.3

Date: November 28, 2008

Contents

References

Name	Description
Gateway Framework User Guide	This use guide describes in detail the functionality of the Gateway Framework, and the standard suite of tools available.
3GPP TS 32.401	The 3GPP TS 32.401 document is part of the 32.400-series covering the 3rd Generation Partnership Project technical specification of Telecom performance management requirements. The document describes the requirements for the management of performance measurements and the collection of performance measurement result data across GSM and UMTS networks.
3GPP TS 32.615	The 3GPP TS 32.615 technical specification captures the configuration management XML file format definition required for defining Network Resource Models (NRMs).
3GPP TS 32.403	Describes the mechanisms involved in the collection of the data and the definition of the UMTS and combined UMTS/GSM performance data itself.

Glossary

3GPP	3rd Generation Partnership Project
ASN.1	Abstract Syntax Notation 1
FTP	File Transfer Protocol
GSM	Global System for Mobile communications
GPRS	General Packet Radio Service
NE	Network Element
NM	Network Manager
NRM	Network Resource Models.
OA&M	Operation, Administration and Maintenance
TS	Technical Specification
UMTS	Universal Mobile Telecommunications System
UTRAN	Universal Terrestrial Radio Access Network
XML	eXtended Markup Language

Preface

About this Guide

This guide details the vendor specific information on the 3GPP XML Gateway. It contains the following information:

- *Chapter 1. Overview.* This chapter gives a brief description of the 3GPP XML Gateway and the raw data format it parses.
- *Chapter 2: Engine Rules and Configuration.* This chapter details the vendor specific rules for parsing the raw data and their configuration.
- *Chapter 3: Post Parser Rules and Configuration.* This chapter describes any vendor specific Post Parser rules and their configuration.
- *Chapter 4: Tech Pack Support.* This chapter describes any standard support for Tech Packs included with the Gateway.
- *Chapter 5: Installation specific information.* This chapter contains the customer installation specific information.

Conventions

The following conventions are used in this guide:

`Fixed width`	Highlights a block of example code, a configuration entry, or a command line instruction

1. Overview

1.1 The Gateway Framework

The 3GPP XML uses the Gateway Framework as a container for the execution of its engine and post parser stages. The Gateway Framework and 3GPP XML Gateway are decoupled into two separate installations. The Gateway Framework consists of a library of Perl modules that provide functionality such as:

- a container for the execution of the 3GPP XML Engine and Post Parser rules for data transformation
- Intermediate (PIF) and output data (LIF) storage and management
- logging utilities
- cleanup and crash recovery
- statistics gathering

The 3GPP XML Gateway plugs into the Gateway Framework and extends this functionality to provide the final Gateway that parses the 3GPP XML data. More information on the standard Gateway configuration is contained in the [Gateway Framework User Guide].

Only 3GPP XML Gateway specific configuration details will be described in this document.

1.2 3GPP XML Overview

1.2.1 3GPP

3GPP stands for 3rd Generation Partnership Project. 3GPP is a collaboration agreement that brings together a number of telecommunications standards bodies, which are known as organizational partners. The main purpose of the partnership is to produce technical specifications and reports for the 3rd Generation mobile system.

One of these specifications, the [3GPP TS 32.401] covers the requirements for the collection and management of performance measurements result data across GSM and UMTS networks. It defines the administration of measurement schedules by the Network Element Manager (EM), the generation of measurement results in the Network Elements (NEs) and the transfer of these results to one or more systems. An example of such systems, north-bound from the data source are the Vallent performance management products.

Another specification of interest is the [3GPP TS 32.615] which defines the configuration management XML file format definition required for defining Network Resource Models (NRM).

1.2.2 Network Details

Several of the vendors who supply GPRS and UMTS equipment to Vallent customers produce performance statistics and network configuration management data from their equipment and OA&M management systems that complies with the 3GPP specifications in XML format. These vendors include Ericsson, Nortel, Alcatel and Motorola.

Data is produced by the NEs to support several areas of performance evaluation including user and signalling traffic, verification of the network configuration, resource access measurements, Quality of Service (e.g. delays during call set-up, packet throughput, etc) and resource availability.

1.2.3 Data Types

The XML performance measurement data derives from GPRS or UMTS network elements, typically via the OA&M management collection point for the network. The periodic measurement files, which can be sizeable, are delivered by the Network OA&M system for collection by northbound OSS applications. This delivery mechanism simplifies the Gateway interface with the network to that of a synchronous file transfer at predefined intervals.

The 3GPP XML Gateway processes two basic data types. These are the performance data and configuration data.

The performance data will typically contain counter measurements for 15 minute intervals, and cover measurements relating to the operation of NodeB, RNC, SGSN, GGSN UMTS network elements.

Each element type may contribute several different data data types, which can be contained in the same measurement results file. Examples of data types for each element type, based on the [3GPP TS 32.402], are:

RNC:

- Radio Access Bearer measurements (RAB)
- Signalling Connection Establishment (SIG)
- Radio Resource Control connection establishment (RRC)
- Radio Resource Control connection re-establishment (RRC)
- Radio Resource Control connection release (RRC)
- Radio Link Control connection (RLC)
- Soft hand-over (SHO)
- Radio link addition procedure (UTRAN side)
- Hard hand-over (HHO)
- Relocation (RELOC)
- Circuit switched inter-RAT hand-over
- Packet switched inter-RAT hand-over
- Iu connection release

SGSN:

- Mobility Management (MM)
- Subscriber Management (SUB)
- SRNS Relocation
- Security (SEC)

SMS
- Session Management (SM)
- CAMEL Measurements (CAM)
- UMTS-GSM Intersystem Change
- UMTS GTP Measurements (GTP)
- UMTS Bearer Service (UBS)

GGSN:
- Session Management (SM)
- Per APN measurements
- GTP measurements (GTP)
- GTP' measurements (GTPP)
- IP measurements (IP)

These groupings listed are generic to UMTS standards and are defined in the [3GPP TS 32.402] document. These groupings may be included in the counter measurement names. Examples of counters from these groupings are:

- HHO.SuccOutIntraCell;
- MM.AttachedSubs.Max;
- RAB.EstabAttCS.Conversational
- RRC.ConnEstab.Cause

Note that vendors will have a high percentage of vendor specific measurement groups and counters. This is especially true for UMTS.

The following prefixes are added for non-UMTS measurements:

- VS for Vendor Specific counters
- Q3 for Q3 measurements
- MIB for IETF measurements (ATM, IP)
- OS for other standards measurements.

These prefixes may be removed from the Gateway output data using the appropriate configuration.

1.2.3.1 Counter Types.

An important aspect of the parsing of the performance data concerns knowing the counter type for each counter. This affects the manner in which the data should be processed by the Gateway. The 3GPP defined counter types are:

CC - Cumulative Counter
The NE maintains a running count of the event being counted. The counter is reset to a well-defined value (usually "0") at the beginning of each granularity period.

Pegged Counters
These counters gradually increment, in accordance with the event being counted, until the defined counter type size is reached (32 bit integer) and the counter rolls over, back to zero. These counters need to be un-pegged to capture the delta between the current and previous measurement data for each pegged counter.

GAUGE Counters
These are dynamic variable counters used when data being measured can vary up or down during the period of measurement. Gauges represent dynamic variables that may change in either direction (e.g. cabinet temperature). It may be initialized at the beginning of each granularity period or at the start of a recording interval.

Discrete Event Registration (DER) Counters
When data related to a particular event are captured every nth event is registered, where n can be 1 or larger.

SI - Status Inspection.
These counts are read at a predetermined rate, the rate is usually based upon the expected rate of change of the count value. Status inspection measurements shall be reset at the beginning of the granularity period and will only have a valid result at the end of the granularity period.

1.2.4 Data/File Formats

The [3GPP TS 32.401] standard supports both the ASN.1 and XML data format. Here we are only concerned with the XML option. Also, only DTD based validation is supported by the 3GPP XML Gateway.

The following outlines the data and filename formats processed by the 3GPP XML Gateway.

1.2.4.1 Performance Data Layout

The generated performance measurement data shall be in ASCII format and in a standardised XML structure.

A sample 3GPP XML file is shown in Appendix B. The definition of all the contents of the XML file are described in table 1.

ASN.1 Tag	DTD based XML Tag	XML schema based XML tag	Description
MeasDataCollection	Mdc	MeasCollecFile	This is the top-level tag, which identifies the file as a collection of measurement data. The file content is made up of a header ("measFileHeader"), the collection of measurement result items ("measData"), and a measurement file footer ("measFileFooter").
MeasFileHeader	Mfh	FileHeader	This is the measurement result file header to be inserted in each file. It includes a version indicator, the name, type and vendor name of the sending network node, and a time stamp ("collectionBeginTime").

ASN.1 Tag	DTD based XML Tag	XML schema based XML tag	Description
MeasData	Md	MeasData	The "measData" construct represents the sequence of zero or more measurement result items contained in the file. It can be empty in case no measurement data can be provided. Each "measData" element contains the name of the NE ("nEId") and the list of measurement results pertaining to that NE ("measInfo").
MeasFileFooter	Mff	FileFooter	The measurement result file footer to be inserted in each file. It includes a time stamp, which refers to the end of the overall measurement collection interval that is covered by the collected measurement results being stored in this file.
FileFormatVersion	Ffv	fileHeader fileFormatVersion	This parameter identifies the file format version applied by the sender. The format version defined in the present document shall be the abridged number and version of the 3GPP document for XML formats and the ASN.1 format alike.
SenderName	Sn	fileHeader dn Prefix and fileSender localDn	The senderName uniquely identifies the NE or EM that assembled this measurement file by its Distinguished Name (DN), according to the definitions in 3GPP TS 32.300 [10]. In the case of the NE-based approach, it is identical to the sender's "nEDistinguishedName". For ASN.1 and DTD based XML format, the string may be empty (i.e. string size =0) in case the DN is not configured in the sender.
SenderType	St	fileSender elementType	This is a user configurable identifier of the type of network node that generated the file, e.g. NodeB, EM, SGSN. The string may be empty (i.e. string size =0) in case the "senderType" is not configured in the sender.
VendorName	Vn	fileHeader vendorName	The "vendorName" identifies the vendor of the equipment that provided the measurement file.
CollectionBeginTime	Cbt	measCollec beginTime	The "collectionBeginTime" is a time stamp that refers to the start of the first measurement collection interval (granularity period) that is covered by the collected measurement results that are stored in this file.
NEId	Neid	ManagedElement	The unique identification of the NE in the system. It includes the user name ("nEUserName"), the distinguished name ("nEDistinguishedName") and the software version ("nESoftwareVersion") of the NE.
NEUserName	Neun	ManagedElement userLabel	This is the user definable name ("userLabel") defined for the NE. The string may be empty (i.e. string size =0) if the "nEUserName" is not configured in the CM applications.
NEDistinguishedName	Nedn	fileHeader dnPrefix and managedElement localDn	This is the Distinguished Name (DN) defined for the NE. It is unique across an operator's 3G network. The string may be empty (i.e. string size =0) if the "nEDistinguishedName" is not configured in the CM applications.
NESoftwareVersion	Nesw	ManagedElement swVersion	This is the software version ("`swVersion`") defined for the NE. This is an optional parameter which allows post-processing systems to take care of vendor specific measurements modified between software versions.
MeasInfo	Mi	MeasInfo	The sequence of measurements, values and related information. It includes a list of measurement types ("measTypes") and the corresponding results ("measValues"), together with the time stamp ("measTimeStamp") and granularity period ("granularityPeriod") pertaining to these measurements.
MeasTimeStamp	Mts	GranPeriod endTime	Time stamp referring to the end of the granularity period.

ASN.1 Tag	DTD based XML Tag	XML schema based XML tag	Description
JobId	Jobid	job jobId	The "jobId" represents the job with which measurement result contained in the file is associated. The "jobId" is mandatory when PMIRP is supported.
GranularityPeriod	Gp	granPeriod duration	Granularity period of the measurement(s) in seconds.
ReportingPeriod	Rp	repPeriod duration	Reporting period of the measurement(s) in seconds. The "reportingPeriod" is mandatory when PMIRP is supported.
MeasTypes	Mt	MeasTypes or measType	This is the list of measurement types (counter names) for which the following, analogous list of measurement values ("measValues") pertains.
MeasValues	Mv	MeasValue	This parameter contains the list of measurement results for the resource being measured, e.g. trunk, cell. It includes an identifier of the resource ("measObjInstId"), the list of measurement result values ("measResults") and a flag that indicates whether the data is reliable ("suspectFlag").
MeasObjInstId	Moid	measValue measObjLdn	The "measObjInstId" field contains the local distinguished name (LDN) of the measured object within the scope defined by the "nEDistinguishedName". The concatenation of the "nEDistinguishedName" and the "measObjInstId" yields the DN of the measured object. The "measObjInstId" is therefore empty if the "nEDistinguishedName" already specifies completely the DN of the measured object, which is the case for all measurements specified on NE level. For example, if the measured object is a "ManagedElement" representing RNC "RNC-Gbg-1", then the "nEDistinguishedName" will be for instance "DC=a1.companyNN.com,SubNetwork=1,IRPAgent=1,SubNetwork=CountryNN,MeContext=MEC-Gbg-1,ManagedElement=RNC-Gbg-1", and the "measObjInstId" will be empty. On the other hand, if the measured object is a "UtranCell" representing cell "Gbg-997" managed by that RNC, then the "nEDistinguishedName" will be for instance the same as above, i.e. "DC=a1.companyNN.com,SubNetwork=1,IRPAgent=1,SubNetwork=CountryNN,MeContext=MEC-Gbg-1,ManagedElement=RNC-Gbg-1", and the "measObjInstId" will be for instance "RncFunction=RF-1,UtranCell=Gbg-997".
MeasResults	R	MeasResults or R	This parameter contains the sequence of result values for the observed measurement types. The "measResults" sequence shall have the same number of elements, which follow the same order as the measTypes sequence. Normal values are INTEGERs and REALs. The NULL value is reserved to indicate that the measurement item is not applicable or could not be retrieved for the object instance.
SuspectFlag	Sf	Suspect	Used as an indication of quality of the scanned data. FALSE in the case of reliable data, TRUE if not reliable. The default value is "FALSE", in case the suspect flag has its default value it may be omitted.
TimeStamp	Ts	MeasCollec endTime	ASN.1 GeneralizedTime format. The minimum required information within timestamp is year, month, day, hour, minute, and second.

Table 1 - Mapping of ASN.1 Measurement Report File Format tags to XML tags

1.2.4.2 File naming specification

A 3GPP compliant XML file will have to follow the naming convention as shown below:

<Type><Startdate>.<Starttime>-[<Enddate>.]<Endtime>_[<UniqueId>][_-_<RC>]

- <Type> consist of (A/B/C/D)
  - A - Single NE, single granularity period.
  - B - Multiple NEs, single granularity period.
  - C - Single NE, multiple granularity periods.
  - D - Multiple NEs, multiple granularity periods.

- <Startdate> is the date when the granularity period begins (for A and B) or the first granularity period of the measurement result comes to availability (C and D). It takes the format of YYYYMMDD, where
  - YYYY - 4 digits year.
  - MM - 2 digits month (01-12).
  - DD - 2 digits day (01-31).

- <Starttime> is the time when the scenario mentioned above happens. It takes the format of HHMMshhmm, where
  - HH - 2 digits hour (00-23)
  - MM - 2 digits minute (00-59)
  - s - the sign for UTC time (+ or -)
  - hh - 2 digits hour for local time differential from UTC time (00-23)
  - mm - 2 digits minute for local time differential from UTC time (00-59)

The following are optional tokens in the filename:

- <Enddate> only exists if the type is C or D. It indicates the end of the last granularity period. The format is the same as <Startdate>.
- <Endtime> indicates when the granularity period (the last granularity period if type is C or D) ends. The format is same as <Starttime> but the minute portion is only in multiple of 5, i.e. (00, 05, 10, 15 etc.).
- <UniqueId> is the name of the NE, EM or domain. This field is optional.
- <RC> is a running count. It shall be appended only if the filename is not unanimous.

Example of 3GPP XML performance data files are as follows:

C20011004.1015-20011004.1045_SGSN-OAM32

C20011004.1045-20011004.1115_SGSN-OAM32

A20040127.1200-1215_SubNetworkItaliaRMERNC0095MeContextRMERNC0095.xml

1.2.5 Architectural extensions

The Perl based 3GPP XML engine is The interface is designed to convert from 3GPP compliant XML data to the
Gateway PIF format. The interface uses the XML::Parser parser library to facilitate XML parsing. XML DTD and Schema validation is not currently supported. The core 3rd party tools are the XML::Parser Perl modules which interfaces to the Expat C based XML parser.

The XML::Parser XML parser complies with SAX 1.0 and 2.0 and DOM 1.0 and 2.0 specifications. The 3GPP XML interface uses SAX callback routines to handle XML elements, attributes etc. This approach facilitates fast parsing.

1.2.5.1 DTD Validation

No DTD validation is supported in the Gateway. Deployments requiring such validation should seek to have this process completed by the equipment vendor element management systems supplying the XML.

2. Engine Rules and Configuration

2.1 XML_3GPP

The essential task of the XML_3GPP engine is to parse the XML data into output PIF files.

There will typically be a one-to-many relationship between one XML file, which will probably map to several PIF files. One PIF file will be produced for every <mi></mi> element in the data, as this XML block represents the counter names and values for a particular network abstraction for a particular measurement period. Note that the PIF file output will also include the header data common to the XML file.

A key goal of a successful configuration is to ensure that all the XML data is outputted in PIF files, and that no PIF duplication occurs. This duplication may occur if the configurations such as the PIF filename tokens do not give a unique filename for that particular measurement.

2.1.1 Processing Sequence

The XML_3GPP processes the performance data in several steps as listed below:

1. Validate the filename so that it is conform to the 3GPP filename convention.
2. Te XML file is parsed and a unique PIF file is create for each PIF mapping. Counter names and counter values are mapped to a single row in PIF.

2.1.2 Rule Configuration

The following details the vendor specific rule entries for the 3GPP XML engine rule.

- XML_PIF_HEADER_ELEMENT_NAMES: This scalar or array entry facilitates the inclusion of XML element data in the output PIF header block. This data is common to all PIF output files.

```
XML_PIF_HEADER_ELEMENT_NAMES => ['sn','st','vn','gp','cbt','mts']
```

- XML_UNIQUE_PIF_BLOCK_HEADER_ELEMENT_NAMES: These elements contain data to be included in the header block for the PIF file generated for each <mi> block. The data included here is unique to each <mi> block of XML, such as <mts> and <gp>.

```
XML_UNIQUE_PIF_BLOCK_HEADER_ELEMENT_NAMES  => ['mts','gp','moid']
```

- XML_UNIQUE_PIF_BLOCK_ELEMENT_NAME: This entry configures the element, which contains the data to constitute a unique PIF file. The 3GPP <mi> element logically contains this data. A new PIF handle is created for each new <mi> block.

```
XML_UNIQUE_PIF_BLOCK_ELEMENT_NAME  => 'mi'
```

- XML_3GPP_FILENAME_VALIDATION: This entry consists of tokens of the 3GPP measurement results filename and the corresponding Perl Regular Expressions that matches the filename token. Further validation such as range checking is done for configured items. The follow entries are mandatory in the filename and must appear in the XML_3GGP rule configuration

```
XML_3GPP_FILENAME_VALIDATION => {
                Type      => '^(A|B|C|D).+$',
                StartDate => '^\w{1}(\d{8}).+$',
                StartTime => '^\w{1\d{8}\.(\d{4})[+-](\d{4}).+$',
}
```

  The following entries are optional in the filename:
  EndDate
  Endtime
  UniqueID
  RC

- XML_HEADER_INFO_FOR_PIF_FILENAME: Values of PIF header entries can be included in the output PIF filename by including their element names in this array. It is critical that enough header counters are included here, so as to ensure a unique PIF file.

```
XML_HEADER_INFO_FOR_PIF_FILENAME=>['Type','StartDate','StartTime']
```

- XML_OUTPUT_BLOCK_ELEMENT_NAME: This scalar entry contains the element tag whose value will be used for the PIF block name in the PIF files. It is also the first item in the PIF filename.

```
XML_OUTPUT_BLOCK_ELEMENT_NAME => 'moid'
```

- XML_OUTPUT_BLOCK_DESCRIPTION: The XML_OUTPUT_BLOCK_DESCRIPTION Regular Expression entry facilitates the configuration of the blockname based on

a pattern match and extraction of the value of the entry. For example, for the following moid:

```
<moid>RncFunction="0",UtranCell="18"</moid>

XML_OUTPUT_BLOCK_DESCRIPTION => '^(\w+)=\"*\w+-*\w*\"*\,*(\w*).*$'
```

First, the pattern must match the value of the <moid> element. A failure to match will be logged, and the block name will default to "BLOCKNAME". Having matched the <moid> value, the tokens extracted by () are used to build up the blockname. In this example, the blockname will be

```
RncFunction_UtranCell
```

based on the RE configured. Care must be taken to ensure that the RE handles one, two, or three name=value entries which may be present in the element value chosen to extract the blockname from.

- XML_ELEMENT_ATTRIBUTE_DESCRIPTION: The Regular Expression of this entry facilitates the configuration of the XML tag attributes based on a pattern match and extraction of the values of the XML tag attributes. For example, for the following XML tag:

```
<measValue
measObjLdn="nedType=solaris,PacketId=1,SymbolicNodeName=pw28,ArrayIdWith
Index=Processes[0]">
  ...
</measValue>

XML_ELEMENT_ATTRIBUTE_DESCRIPTION => '([^=,]+)=\"?([^",]*)\"?',
```

The above <measValue> tag has the attribute "measObjLdn" which has values consisting of a few "name=value" pairs. This entry allows the "name=value" pairs to be extracted by the use of Regular Expression. The "name=value" pairs are extracted by a pair of parenthesed elements, () in the Regular Expression where the first element will be the counter name and the second element will be the counter value. Using the above example, the following counters will be produced:

Counter Name	Counter Value
measValue_nedType	solaris
measValue_PacketId	1
measValue_SymbolicNodeName	pw28
measValue_ArrayIdWithIndex	Processes[0]

- XML_ALTERNATIVE_OUTPUT_BLOCK_NAMES: This entry facilitates the substitution of output block names with alternatives.

```
'XML_ALTERNATIVE_OUTPUT_BLOCK_NAMES' => {
                                         "GGSN" => "GGSN_Circuit",
                                        },
```

- XML_MEASUREMENT_TYPE_ELEMENT_SEQ: This scalar entry contains the element tag whose value contains a counter name. A sequence of these element

tags contains all the counter names for a particular PIF block.

```
XML_MEASUREMENT_TYPE_ELEMENT_SEQ => 'mt'
```

- XML_MEASUREMENT_VALUE_ELEMENT_NAME: This element tag's sub-elements contain a list of measurement results for the resource being measured. The sub-elements will map to one PIF record entry. It contains the list of measurement results for the resource being measured.

```
XML_MEASUREMENT_VALUE_ELEMENT_NAME => 'mv'
```

- XML_MEASUREMENT_RESULTS_ELEMENT_SEQ: This scalar entry contains the element tag whose value contains a counter value. A sequence of these element tags contains all the counter values for a particular PIF record in a PIF block. The Measurement Results sequence shall have the same number of elements, which follow the same order as the Measurement Types sequence <mt>.

```
XML_MEASUREMENT_RESULTS_ELEMENT_SEQ => 'r'
```

- XML_OBJECT_INSTANCE_ID_ELEMENT_NAME: This scalar entry contains name of the element tag whose value contains the measurement Object Instance lD. It identifies the relative distinguished name of the measured object within the scope defined by the nEDistinguisedName. The concatenation of the nEDistinguisedName and the measObjInstld yields the DN of the measured object. This scalar entry allows for the ID of the current measured object to be included in each record.

```
XML_OBJECT_INSTANCE_ID_ELEMENT_NAME => 'moid'
```

- XML_ENABLE_PIF_PER_MOID: Normally the engine rule will create a PIF for each mi element. This can lead to a large number of difficult to manange files being created. If this option is set to true, the rule will create a PIF file for each moid key. So is there are a number of mi entries in the file for the same moid, they will output to the same PIF. If the counters associated with the moid change a new PIF is created.

```
XML_ENABLE_PIF_PER_MOID => 0,
```

- XML_SUSPECT_FLAG_ELEMENT_NAME: This scalar entry contains the name of the element tag whose value contains the indication of the quality of the scanned data. This flag defaults to FALSE in the case of reliable data, If it's set to TRUE, then the record is unreliable, and the current PIF record is omitted from the PIF file. It is an optional item in the XML.

```
XML_SUSPECT_FLAG_ELEMENT_NAME = "sf"
```

- XML_LIST_DELIMITER: This scalar entry contains the delimiter that is being used in the XML to separate counter names or values in an XML tag. For instance, the corresponding counter names and values are residing in <measType></measType> and <measResults></measResults> for Motorola UTRAN.

```
XML_LIST_DELIMITER = " "
```

- XML_DATA_FOR_PIF_RECORD: This is an optional array of scalar. It contains the element name that is used for PIF header population. These counter name/value pair shall be populated into each PIF records.

```
XML_DATA_FOR_PIF_RECORD = [ qw( mts SubNetwork ) ],
```

- XML_ALTERNATIVE_COUNTER_NULL_VALUE: This scalar entry allows for the substitution of a defined value wherever NULL values are.

```
XML_ALTERNATIVE_COUNTER_NULL_VALUE  => '-9999'
```

- XML_MAX_COUNTER_NAME_SIZE: This entry defines the maximum size that counter names in the PIF files should be. The counters will actually be truncated to this size.

```
XML_MAX_COUNTER_NAME_SIZE => 30
```

- XML_ALTERNATIVE_COUNTER_NAMES: This entry facilitates the configuration of regular expressions, which can be used to process counter names. The array of arrays entry consists of pairs of RE's; the first entry of each pair designed to match the counter and the second to provide the replacement RE, based on extractions for the match. For example, the counter name **VS.HLR.TONENATT** can be matched and replaced as follows:

```
XML_ALTERNATIVE_COUNTER_NAMES => [
                                   ['VS\.HLR\.(.+)' , '$1'],
                                 ]
```

  The output counter name for this example is **TONENATT**. The processing of counter name is thus flexible and extensible based on regular expressions.

- XML_LINE_COUNT_TO_TOP_TRIM: This scalar optional entry allows for a configurable number of lines from the top of the XML file to be removed before passing to the XML::Parser. These lines may include entries unsupported by XML::Parser, and need to be removed before parsing.

```
XML_LINE_COUNT_TO_TOP_TRIM => 2
```

- XML_ENABLE_PIF_COUNTER_IN_FILENAME: This is an optional scalar entry, when set to "True", enables the inclusion of a PIF counter entry in the PIF filename. This is required when all other PIF tokens used to make up a PIF filename are identical over two or more PIF files to prevent PIF files from being overwritten. Sample configurations:

```
XML_ENABLE_PIF_COUNTER_IN_FILENAME  => 'True',
XML_ENABLE_PIF_COUNTER_IN_FILENAME  => 0
```

  The following are some sample PIF output files containing counter values:

```
AtmInterface_80-#-04Oct2001_11:00:00-#-C-#-0-#-I.pif
AtmInterface_80-#-04Oct2001_11:00:00-#-C-#-1-#-I.pif
AtmInterface_80-#-04Oct2001_11:00:00-#-C-#-2-#-I.pif
```

- HEADER_RECORD_PROCESSING: This is an optional code entry. Extra logic can be written in this entry. The code shall be executed right before the header record is being written into PIF file. Note that only header records are being treated within this entry (modify or add counters/values). This entry should not exist in the engine rule if it is not needed, to avoid performance issue.

```
HEADER_RECORD_PROCESSING  => sub {
    my ($op_hash_ref) = @_;

    if ( exists($op_hash_ref->{sn_RNC}) ) {
        $op_hash_ref->{sn_RNC} =~ /^0*(\d+)\/(.+)/;
        if ( $1 && $2 ) {
            $op_hash_ref->{'sn_RNC_ID'} = $1;
            $op_hash_ref->{'sn_RNC_Name'} = $2;
            delete $op_hash_ref->{sn_RNC};
        }
    }

    return 0;
},
```

- DATA_RECORD_PROCESSING: This is an optional code entry. This entry is similar to HEADER_RECORD_PROCESSING. The difference is only records from data block are treated instead of header records. This entry should not exist in the engine rule if it is not needed, to avoid performance issue.

```
DATA_RECORD_PROCESSING  => sub {
    my ($op_hash_ref) = @_;

    return 0;
},
```

- XML_DDM_MANIP_RULES: This points to a subroutine, ddm_manip_config, which is imported from DDM_MANIP_Config.pm. This subroutine returns the configuration for manipulation of the DDM elements.

```
XML_DDM_MANIP_RULES  => ddm_manip_config()
```

- SUSPECT_FLAG_VALIDITY: This is an optional scalar entry. If set to "True", raw file will be turned to bad when raw data entries with suspect flag set to TRUE is found.
- XML_MEASUREMENT_VALUE_VALIDITY: This is an optional scalar entry. If set to "True", raw file will be turned to bad when raw data entries with empty/blank measurement value is found.
- XML_PARSE_ELEMENT_NAME_ATTR: This an optional scalar or array entry. When parser detect a element tag that has been specified here, parser will parse the attributes of the tag and ouput them into the pif record.

```
XML_PARSE_ELEMENT_NAME_ATTR => ['measValue'],
```

2.1.3 DDM_MANIP_Config

The DDM_MANIP_Config contains configuration for those measurement types that consist of a list of values within one element e.g.

```
<r>-1.5,62,6,54,-1,-1</r>
```

These values are manipulated (totalled, averaged, printed etc.) to produce the PIF/LIF counters depending on the configuration stored in DDM_MANIP_Config.pm.

Before explaining the configuration entries, below is a brief explanation of the manipulation options available for each list of values within a measurement type:

1. **ALL** - This will take each value in the input data and output them in the PIF.
2. **TOTAL** - For a configured range as part of the rule, this rule will total all values in that range.
3. **AVERAGE** - For a configured range as part of the rule, this rule will calculate the average of the values in that range, ignoring invalid values.
4. **WEIGHTED_AVG** – For a configured range, this rule will calculate the weighted average of the values in that range based on the configured referenced values.
5. **REFERENCE** – For configured range, this rule will return the configured referenced value of the reference type, either the FIRST or LAST position.
6. **TOP** - For a configured N (default 5), the rule will output the TOP N values and their position (index) in the array.

The DDM_MANIP_RULES hash contains the following configuration entries:

- INVALID_VALUES: The array of values, which are considered to be invalid and will be ignored for all operations.

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

- MISSING_VALUE: The value to use in the output counter if it cannot be derived, or is one of the INVALID_VALUES. For example if the average is being calculated and all the entries from which it is being calculated are invalid (-1), the counter will be outputted in the PIF with the MISSING_VALUE

  ```
  MISSING_VALUE  => 'NULL',
  ```

- PRECISION: For the AVERAGE operation, it is likely that the average result will be a floating number. This option sets the number of digits to be printed after the decimal point:

  ```
  PRECISION      => '2',
  ```

  In the case above average values will be 1.50, 45.66 etc. If no decimal digits are required set it to 0, and the whole number portion will be outputted.

- COUNTERS_TO_MANIP: A hash containing the list of DDM measurement types (counters), which are to be manipulated. There is one entry in this hash for each counter.

```
COUNTERS_TO_MANIP =>
     {
         pmDpchCodePower =>
```

Each counter then contains an array. This array lists has hash entries of the rules to apply to the list of values. Therefore more than one manipulation (ALL, AVERAGE, TOTAL etc) can be applied to each counter. The options and configuration for these values are explained under the section DDM Manipulation and Configuration.

2.2 XML_3GPP_CONFIGURATION

The processing steps of XML_3GPP_CONFIGURATION are similar to XML_3GPP. The only difference is that it is dealing with bulk configuration data that is coming from the network.

2.2.1 Rule Configuration

- IGNORE_ELEMENTS: This array entry contains the list of elements that should be ignored by the rule. All sub-elements within the element will also be ignored.

  ```
  IGNORE_ELEMENTS => [],
  ```

- COMMON_HEADER_DATA_OBJECTS: This array entry lists the atomic element names, which should be included in the PIF header as counters. These entries are outside the OBJECT_MAPPING XML.

- OBJECT_MAPPING: This mandatory hash entry contains the basic object mapping of the XML to PIF. Each OBJECT_MAPPING hash key contains the name of the XML element that commences a new PIF output for the contents of that element. The id attribute for this element is placed in the header of the PIF. Each OBJECT_MAPPING key entry in turn references a hash containing configuration entries, which determine certain mappings for the element in question. These configuration entries are:

  - COMMON_HEADER_DATA_OBJECTS: This array entry lists the atomic element names, which should be included in the PIF header as counters.

  - PIF_FILENAME_KEYS: This array entry lists the counter names in either the header or record data that should be used in formulating the PIF filename.

  - CHILD_ELEMENTS: This hash entry contains the PIF record mapping details. Each key contains an element name that maps to the point in the XML when a new PIF record commences. Each key references another hash entry that contains configuration information relevant to the PIF record. These configuration entries are:

    - NON_UNIQUE_CLASS_BLOCKS: This array entry contains the names of elements within the child element, which appear more than once. Because of the potential for overwriting on these element counters, these counter entries in the PIF filename are pre-pended with a tag. This tag is made up of the NON_UNIQUE_CLASS_BLOCKS element name and its "id" attribute.

NON_UNIQUE_CLASS_BLOCKS entries must have an "id" attribute in order to ensure uniqueness.

- ENABLE_UNIQUE_PIF: Setting the ENABLE_UNIQUE_PIF to "TRUE" facilitate the placing of each PIF record in a unique PIF file.
- OUTPUT_BLOCK_NAME: This scalar entry allows for setting the name of the output block name for a particular CHILD_ELEMENTS entry.
- DUPLICATE_COUNTERS: An atomic element name, appearing as an actual counter in the PIF file should be included in this array list, if the name of the counter appears more than once in the XML block for the PIF record. This facilitates the placing of a unique incrementing numeric value on these values (vsDataType_1, vsDataType_2 etc.).

A sample OBJECT_MAPPING entry is as follows:

```
OBJECT_MAPPING                    => {
   'RncFunction'  => {
      COMMON_HEADER_DATA_OBJECTS  => [qw(RncFunction_id userLabel
                                         mcc mnc rncId)],
      PIF_FILENAME_KEYS           => [qw(RncFunction_id rncId)],
      CHILD_ELEMENTS => {
         'UtranCell' => {
            NON_UNIQUE_CLASS_BLOCKS => [qw(UtranRelation)],
            ENABLE_UNIQUE_PIF       => 0,
            OUTPUT_BLOCK_NAME       => 'UtranCell',
            DUPLICATE_COUNTERS      => [qw(vsDataType)],
         },
      },
   },
},
```

- SUSPECT_FLAG_VALIDITY: This is an optional scalar entry. If set to “True”, raw file will be turned to bad when raw data entries with suspect flag set to TRUE is found.
- XML_MEASUREMENT_VALUE_VALIDITY: This is an optional scalar entry. If set to “True”, raw file will be turned to bad when raw data entries with empty/blank measurement value is found.

3. DDM Manipulation and Configuration

COUNTERS_TO_MANIP contains the list of DDM measurement types (counters), which are to be manipulated. There is one entry in this hash for each counter.

```
COUNTERS_TO_MANIP =>
     {
         pmDpchCodePower =>
```

Each counter then contains an array. This array lists consists hash entries of the rules to be applied to the list of values. Therefore more than one manipulation

(ALL, AVERAGE, TOTAL etc) can be applied to each counter. The options and configuration for these values are explained below.

3.1 ALL

The **ALL** operation prints all elements in the list, appending _<index> to the counter name where index is the position in the list. For example, for counter pmDpchCodePower with 4 list entries as follows in the input XML file:

```
17, 14, 45, 24
```

This will result in the following counters in the output PIF.

```
pmDpchCodePower_0|pmDpchCodePower_1|pmDpchCodePower_2|pmDpchCodePower_3
17|14|45|24
```

Sample configuration for this rule:

```
COUNTERS_TO_MANIP =>
  {
      pmDpchCodePower =>
         [
           { MANIP_TYPE => 'ALL' },
         ],
   },
```

3.2 TOTAL

The **TOTAL** rule totals all values within a range, ignoring invalid values. If all values with the totalled range are invalid (-1), then the total counter will be outputted with the configured MISSING_VALUE. If the range specified in the configuration goes beyond the length of the actual list, a warning will be printed to the log file. The configurations within the TOTAL MANIP_RULE are:

- RANGE_INDEX: An array of anon. hashes, each of which specifies an application of the TOTAL operation to a range of values. Within each hash entry the configuration entries are:
    - START_INDEX: The starting index with the list to total from
    - END_INDEX: The end index to calculate to. NOTE: the calculation will include this value.
    - OUTPUT_COUNTER_NAME: This is the counter name of the totalled values.

  For example for counter pmDpchCodePower with 4 list entries as follows in the input XML file:

  ```
  17, 14, 45, 24
  ```

  This will result in the following counters in the output PIF.

  ```
  pmDpchCodePower_total|...
  ```

```
99
```

The configuration to achieve this is below.

```
COUNTERS_TO_MANIP =>
   {
      pmDpchCodePower =>
         [
            {
               MANIP_TYPE => 'TOTAL',
               RANGE_INDEX =>
                  [
                     {
                        START_INDEX => '0',
                        END_INDEX   => '3',
                        OUTPUT_COUNTER_NAME =>
'pmTransmittedCarrierPower_total_0_3'
                     },
                  ],
            },
         ],
   }
```

3.3 AVERAGE

The **AVERAGE** manipulation is very similar to the TOTAL, except that the outputted counter value is the average and not the TOTAL. If all values with the average range are invalid (-1), then the OUTPUT_COUNTER will be outputted with the configured MISSING_VALUE. If the range specified in the configuration goes beyond the length of the actual list a warning will be printed to the log file. The configurations within the AVERAGE MANIP_RULE are

- RANGE_INDEX: An array of anon. hashes, each of which specifies an application of the AVERAGE operation to a range of values. Within each hash entry the configuration entries are:
  - START_INDEX: The starting index with the list to average from
  - END_INDEX: The end index to calculate to. NOTE: the calculation will include this value.
  - OUTPUT_COUNTER_NAME: This is the counter name of the calculated average.

For example for counter pmDpchCodePower with 4 list entries as follows in the input XML file:

```
17, 14, 45, 24
```

This will result in the following counter in the output PIF. The PRECISION value has been configured as 2.

```
pmDpchCodePower_avg|...
25.00
```

The configuration to achieve this is below.

```
COUNTERS_TO_MANIP =>
   {
      pmDpchCodePower =>
         [
            {
               MANIP_TYPE => 'AVERAGE',
               RANGE_INDEX =>
                  [
                     {
                        START_INDEX => '0',
                        END_INDEX   => '3',
                        OUTPUT_COUNTER_NAME => '
pmDpchCodePower_avg
                     },
                  ],
            },
         ],
   }
```

3.4 WEIGHTED_AVG

The weighted average manipulation is a combination of AVERAGE and REFERENCE. The reported counter values are multiplied by a list of weighted values and divided by the total of the counter values. The configurations within the WEIGHTED_AVG MANIP_RULE are

- RANGE_INDEX: An array of anon. hashes, each of which specifies an application of the WEIGHTED_AVG operation to a range of values. Within each hash entry the configuration entries are:
  - START_INDEX: The starting index with the list to average from
  - END_INDEX: The end index to calculate to. NOTE: the calculation will include this value.
  - OUTPUT_COUNTER_NAME: This is the counter name of the calculated average.
  - AVG_TYPE: The type of averaging algorithm used. If not specified the default of 'NORMAL' will be used. Supported types are :

    'NORMAL' : The default. All values in the array are totaled up and averaged based on the defined weight.

    'LOGARITHM' : The array values are assumed to be logarithmic, and hence non-linear. Inverse logarithm is applied to each array elements, and then averaged. Logarithm is then applied to the final averaged value.

    Value = Frequency * 10^(Weight/10)

    $Average = 10\log_{10}(Value_{avg})$

'dB' : The array values are in dB and hence non-linear. The same kind of operation is applied as in 'LOGARITHM' above.

'dBm' : The array values are in dBm and hence non-linear. The same kind of operation is applied as in 'dB' and 'LOGARITHM', but the unit used is smaller in 'dBm' instead of 'dB'.

```
Value = Frequency * (0.001) * 10^(Weight/10)
```

$$Average = 30 + 10\log_{10}(Value_{avg})$$

- WEIGHTED_START: The starting value of the linear weighted array relating to START_INDEX
- WEIGHTED_STEP: The stepping value of the linear weighted array for the range
- WEIGHTED_ARRAY: An array of values for non-linear weighted average

Note: The entries can either be a combination of WEIGHTED_START and WEIGHTED_STEP only, or WEIGHTED_ARRAY only:

For example for counter pmDpchCodePower with 4 list entries as follows in the input XML file:

```
17, 14, 45, 24
```

using the configuration below,

```
COUNTERS_TO_MANIP =>
   {
      pmDpchCodePower =>
         [
            {
               MANIP_TYPE => 'WEIGHTED_AVG',
               RANGE_INDEX =>
                  [
                     {
                        START_INDEX => '0',
                        END_INDEX   => '3',
                        WEIGHTED_START => '6.0',
                        WEIGHTED_STEP => '+0.5',
                        OUTPUT_COUNTER_NAME =>
'pmDpchCodePower_avg'
                     },
                  ],
            },
         ],
   }
```

The referenced value list would be:

```
6.0, 6.5, 7.0, 7.5
```

The calculation of weighted average is:

$$\frac{(17 \times 6.0) + (14 \times 6.5) + (45 \times 7.0) + (24 \times 7.5)}{17 + 14 + 45 + 24}$$

This will result in the following counter in the output PIF:

```
pmDpchCodePower_avg|...
6.88|...
```

3.5 REFERENCE

The **REFERENCE** manipulation reports the value from a list that is related to the first or last non-zero counter values. The configurations within the REFERENCE MANIP_RULE are

- RANGE_INDEX: An array of anon. hashes, each of which specifies an application of the AVERAGE operation to a range of values. Within each hash entry the configuration entries are:
  - START_INDEX: The starting index with the list to average from
  - END_INDEX: The end index to calculate to. NOTE: the calculation will include this value.
  - OUTPUT_COUNTER_NAME: This is the counter name of the calculated average.
  - REFERENCE_TYPE: To refer either the FIRST of LAST non-zero counter value
  - REFERENCE_START: The starting value of the linear reference array relating to START_INDEX
  - REFERENCE_STEP: The stepping value of the linear reference array for the range
  - REFERENCE_ARRAY: An array of values for non-linear reference array

For example for counter pmDpchCodePower with 8 list entries as follows in the input XML file:

```
0, 0, 0, 17, 14, 45, 24, 0, 0
```

with the configuration below,

```
COUNTERS_TO_MANIP =>
   {
      pmDpchCodePower => [
         { MANIP_TYPE => 'REFERENCE',
           RANGE_INDEX => [
```

```
                {
                   START_INDEX => '0',
                   END_INDEX   => '8',
                   REFERENCE_TYPE => 'LAST',
                   REFERENCE_START => '6.0',
                   REFERENCE_STEP => '+0.5',
                   OUTPUT_COUNTER_NAME => 'pmDpchCodePower_Max',
                },
             ],
          },
       ],
    }
```

the referenced value list would be:

```
6.0, 6.5, 7.0, 7.5, 8.0, 8.5, 9.0, 9.5, 10.0
```

This will result in the following counter in the output PIF, where the last non-zero value in the list is 24 at index 6:

```
pmDpchCodePower_Max|...
9.0|...
```

3.6 TOP

The TOP manipulation searches the list for the TOP N entries in the list and prints them and their position (index) in the list. If OUTPUT_COUNT is not configured, then by default the top 5 will be derived. 2 output counters are written to the PIF for each value:

<counter_name>_pos_max_<pos> - The position (index of the value)
<counter_name>_pos_val_<pos> - The value of the list entry

For example for counter pmDpchCodePower with 4 list entries as follows in the input XML file:

```
17, 14, 45, 24
```

This will result in the following counters in the output PIF. Note, as in the configuration below the top 2 counters are being derived.

```
pmDpchCodePower_val_max_1|pmDpchCodePower_pos_max_1|pmDpchCodePowe
r_val_max_2|pmDpchCodePower_pos_max_2|
45|2|24|3
```

i.e. the top value is 45, which was found at index 2, the 2nd highest value is 24, which was found at index 3.

```
COUNTERS_TO_MANIP =>
   {
      pmDpchCodePower =>
         [
            {
               MANIP_TYPE => 'TOP',
```

```
                OUTPUT_COUNT => 2,
            },
        ],
    }
```

4. Post Parser Rules and Configuration

4.1 ADD_KEY_INFO_TO_FILENAME

This post parser rule allows new PIF/LIF files to be created, the files named from information extracted from counter names and counter values. The matching counters are also output to the new file. For example for GnIsp moid, there are a number of groups of counter reported, which leads to a number of GnIsp PIF files being created. These files can be renamed correctly using this rule.

The non-standard configuration entries with this rule are listed below.

4.1.1 Rule Configuration

- COUNTERS_USED_TO_SPLIT_FILE: A hashmap containing a list of counter name and the value to extract from them for the creation of the output file key. This extracted value will be used as part of the key to uniquely identify the file.

```
COUNTERS_USED_TO_SPLIT_FILE => {'moid_GnIsp' => '^(\d+)'},
```

- EXTRACT_BLOCK_NAME_FROM_COUNTERS: A hashmap containing a map from the new block name to the RE to match the counters for this block. When a match is found the new block name will be used in the output file. If no match is found, then the block name in the input PIF is used.

```
EXTRACT_BLOCK_NAME_FROM_COUNTERS => {
                            'GgsnFunction_GnIsp_GTP' => 'VS_GTP_.*',
                            'GgsnFunction_GnIsp_SM' => 'VS_SM_.*'
},
```

- HEADER_COUNTERS_TO_USE_IN_OUTPUT_FILENAME: An array containing the list of header counters to use in the output filename.

```
HEADER_COUNTERS_TO_USE_IN_OUTPUT_FILENAME =>
                    ['nedn_ManagedElement','mts_Date','mts_Time'],
```

A full sample configuration is included below:

```
{
    RULE_TYPE       => 'ADD_KEY_INFO_TO_FILENAME',
    RULE_DESC       => 'Add the subtype info to GgsnFunction_GnIsp data',
    INPUT_FILE_DESCRIPTION          => 'GgsnFunction_GnIsp-#-.*-#-I.pif',
    COUNTERS_USED_TO_SPLIT_FILE           => {'moid_GnIsp' => '^(\d+)'},
    EXTRACT_BLOCK_NAME_FROM_COUNTERS      => {
                            'GgsnFunction_GnIsp_GTP' => 'VS_GTP_.*',
                            'GgsnFunction_GnIsp_SM' => 'VS_SM_.*'
    },
    HEADER_COUNTERS_TO_USE_IN_OUTPUT_FILENAME =>
```

```
                          ['nedn_ManagedElement','mts_Date','mts_Time'],
    PRODUCE_PIF     => 0,
    OUTPUT_FORMAT   => 'LIF_Writer',
},
```

4.1.2 Sample Usage

Given the following input PIF

```
## Metrica Parser Intermediate File
##START|HEADER
measCollec_beginTime|st|vn|duration|StartTime|Type|gp_endTime|GMT_differ
ence|StartDate|measCollec_endTime
20040825T00:00:00|NODEB|Motorola|PT1800S|22:45|A|20040825T00:30:00|+60|1
4Feb2002|20040825T12:30:00
##END|HEADER
##START|NodeBFunction_MAIA_IC24_UMTS_nodeBSite_1
iubDedMeasFailedUnknownSend|iubNbErrMsgIncompatRcvrStateRe|measValue_Nod
eBFunction|iubCellCapabilityChanges|iubNbErrTransferSyntaxErrorRec|iubNb
ErrMsgIncompatRcvrStateSe|iubAuditRequired|measValue_PhysicalUnit|iubLoc
alCellCapabilityChanges|iubNbErrUnspecifiedRec|iubDedMeasFailedHwSend
0|0|omcu1_ns:MAIA_IC24_UMTS|0|0|0|0|nodeBSite-1|0|0|0
0|0|omcu2_ns:MAIA_IC24_UMTS|0|0|0|0|nodeBSite-1|0|0|0
0|0|omcu3_ns:MAIA_IC24_UMTS|0|0|0|0|nodeBSite-1|0|0|0
0|0|omcu1_ns:MAIA_IC24_UMTS|0|0|0|0|nodeBSite-1|1|1|1
0|0|omcu2_ns:MAIA_IC24_UMTS|0|0|0|0|nodeBSite-1|2|2|2
0|0|omcu3_ns:MAIA_IC24_UMTS|0|0|0|0|nodeBSite-1|3|3|3
##END|NodeBFunction_MAIA_IC24_UMTS_nodeBSite_1
```

Three LIFs are generated based on measValue_NodeBFunction; header value measCollec_beginTime and measCollec_endTime are added to the LIF filename; and the block name to be used is PhysicalUnit. The LIF files that are generated are as follow:

```
measValue_PhysicalUnit-#-2004-08-25T00:00:00-#-2004-08-25T12:30:00-#-
omcu1_ns-#-AKI.lif
measValue_PhysicalUnit-#-2004-08-25T00:00:00-#-2004-08-25T12:30:00-#-
omcu2_ns-#-AKI.lif
measValue_PhysicalUnit-#-2004-08-25T00:00:00-#-2004-08-25T12:30:00-#-
omcu3_ns-#-AKI.lif
```

The configuration is as follow:

```
{
    RULE_TYPE    => 'ADD_KEY_INFO_TO_FILENAME',
    RULE_DESC    => 'Add the begin/end time to physical unit data',
    INPUT_FILE_DESCRIPTION           => '^NodeBFunction_.*pif$',
    COUNTERS_USED_TO_SPLIT_FILE      => {
                              'measValue_NodeBFunction' => '^(.+):.*'
                                      },
    EXTRACT_BLOCK_NAME_FROM_COUNTERS => {
                              'measValue_PhysicalUnit' => 'measValue.*',
                                      },
    HEADER_COUNTERS_TO_USE_IN_OUTPUT_FILENAME => [
                              'measCollec_beginTime','measCollec_endTime'
                                             ],
    PRODUCE_PIF                      => 0,
```

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

4.2 COPY_HEADER_N_BODY_COUNTERS

This post parser rule allows counters to be copied from header to body, and vice versa. The counters that are going to be copied shall be listed out in the specific array.

The non-standard configuration entries with this rule are listed below:

4.2.1 Rule Configuration

- COPY_HEADER_COUNTERS_2_BODY: A mandatory field that consists of an array of header counters. These counters shall be copied to the PIF body.

```
COPY_HEADER_COUNTERS_2_BODY => [ 'RNC_RNCName',
                                 'RNC_RNCIdentifier', ],
```

- COPY_BODY_COUNTERS_2_HEADER: A mandatory field that consists of an array of PIF body record counters. These counters shall be copied to the PIF header.

```
COPY_BODY_COUNTERS_2_HEADER => [ 'NODEB_NODEBIPAddress',
                                 'RNCCELL_RncNodeBIdentifier' ],
```

- REDUNDANT_HEADER_COUNTERS: An optional field that consists of an array of header counters. These counters shall be removed.

```
REDUNDANT_HEADER_COUNTERS => [ 'RNC_RNCName',
                               'RNC_RNCIdentifier',],
```

- REDUNDANT_BODY_COUNTERS: An optional field that consists of an array of body record counters. These counters shall be removed.

```
REDUNDANT_BODY_COUNTERS => [ 'NODEB_NODEBIPAddress',
                             'RNCCELL_RncNodeBIdentifier', ],
```

A full sample configuration is included below:

```
{
    RULE_TYPE => 'COPY_HEADER_N_BODY_COUNTERS',
    RULE_DESC => 'move counters from body to header for
                  NodeBFunction PIFs',
    INPUT_FILE_DESCRIPTION => [ '^NodeBFunction.*-#-I-#-S\.pif', ],
    COPY_HEADER_COUNTERS_2_BODY => [ ],
    COPY_BODY_COUNTERS_2_HEADER => [ 'NODEB_NODEBIPAddress',
                                     'RNC_RNCName',
                                     'RNCCELL_RncNodeBIdentifier',
                                     'RNC_RNCIdentifier', ],
    REDUNDANT_HEADER_COUNTERS   => [ ],
    REDUNDANT_BODY_COUNTERS     => [ 'NODEB_NODEBIPAddress',
                                     'RNC_RNCName',
                                     'RNCCELL_RncNodeBIdentifier',
                                     'RNC_RNCIdentifier', ],
    PRODUCE_PIF                 => 'True',
```

```
    OUTPUT_FORMAT                => 0,
},
```

4.2.2 Sample Usage

Given the following input file, and with the configuration above:

```
## Metrica Parser Intermediate File
##START|HEADER
granPeriod_endTime_GMT_Difference|fileSender_elementType|fileHeader_vend
orName|granPeriod_duration|measCollec_endTime_Time|fileSender_SubNetwork
|StartTime|granPeriod_endTime_Date|fileSender_ManagedElement|Type|GMT_di
fference|measCollec_beginTime_Date|granPeriod_endTime_Time|measCollec_en
dTime_GMT_Difference|measCollec_beginTime_GMT_Difference|fileHeader_file
FormatVersion|ID|measCollec_endTime_Date|measCollec_beginTime_Time|Start
Date
+00:00|NODEB|Motorola|1800|17:30|temip.omcu1_director|15:30|31Oct2004|PO
RTO_SRA_DA_LUZ_UMTS|C|+0|31Oct2004|17:30|+00:00|+00:00|"32.401V5.1"|PORT
O_SRA_DA_LUZ_UMTS|31Oct2004|15:30|31Oct2004
##END|HEADER
##START|NODEB
iubNbErrMsgIncompatRcvrStateSent|measValue_PhysicalUnit_nodeBSite_ID|iub
CellCapabilityChanges|iubAuditRequired|iubNbErrTransferSyntaxErrorRec|iu
bLocalCellCapabilityChanges|NODEB_NODEBIPAddress|iubNbErrUnspecifiedRec|
iubDedMeasFailedHwSend|RNC_RNCIdentifier|iubDedMeasFailedUnknownSend|RNC
CELL_CellName|measValue_NodeBFunction|Record_measurement_count|RNCCELL_C
ellIdentifier|RNCCELL_RncNodeBIdentifier|iubNbErrMsgIncompatRcvrStateRec
|RNC_RNCName
0|1|0|0|0|0|10.132.13.17|0|0|52|0|099C6_3|PORTO_SRA_DA_LUZ_UMTS|11|9963|
362|0|RNC-52
##END|NODEB
```

The specified 4 body record counters shall be moved to the header as shown below:

```
## Metrica Parser Intermediate File
##START|HEADER
granPeriod_endTime_GMT_Difference|fileSender_elementType|fileHeader_vend
orName|granPeriod_duration|fileSender_SubNetwork|measCollec_endTime_Time
|granPeriod_endTime_Date|StartTime|fileSender_ManagedElement|NODEB_NODEB
IPAddress|Type|GMT_difference|RNC_RNCIdentifier|measCollec_beginTime_Dat
e|granPeriod_endTime_Time|measCollec_endTime_GMT_Difference|fileHeader_f
ileFormatVersion|measCollec_beginTime_GMT_Difference|RNCCELL_RncNodeBIde
ntifier|measCollec_endTime_Date|ID|measCollec_beginTime_Time|RNC_RNCName
|StartDate
+00:00|NODEB|Motorola|1800|temip.omcu1_director|17:30|31Oct2004|15:30|PO
RTO_SRA_DA_LUZ_UMTS|10.132.13.17|C|+0|52|31Oct2004|17:30|+00:00|"32.401V
5.1"|+00:00|362|31Oct2004|PORTO_SRA_DA_LUZ_UMTS|15:30|RNC-52|31Oct2004
##END|HEADER
##START|NODEB
measValue_PhysicalUnit_nodeBSite_ID|iubNbErrMsgIncompatRcvrStateSent|iub
DedMeasFailedUnknownSend|iubCellCapabilityChanges|Record_measurement_cou
nt|measValue_NodeBFunction|RNCCELL_CellName|iubNbErrTransferSyntaxErrorR
ec|iubAuditRequired|RNCCELL_CellIdentifier|iubNbErrMsgIncompatRcvrStateR
ec|iubLocalCellCapabilityChanges|iubNbErrUnspecifiedRec|iubDedMeasFailed
HwSend
1|0|0|0|11|PORTO_SRA_DA_LUZ_UMTS|099C6_3|0|0|9963|0|0|0|0
```

```
##END|NODEB
```

4.3 EXTRACT_INFO

This post parser rule allows the changes to be made on the counter name. The data from the original counter shall be extracted by using the regular expression and copied to the new counter, if specified.

The non-standard configuration entries with this rule are listed below.

4.3.1 Rule Configuration

- EXTRACTION_EXPRESSION: A mandatory field that picks the data selectively from the PIF file using the provided regular expression. Only scalar value is accepted.

```
EXTRACTION_EXPRESSION => '^(.*)$',
```

- OUTPUT_BLOCK_NAME: The new block name for the new PIF file generated. This is an optional entry. It only accepts scalar value.

```
OUTPUT_BLOCK_NAME => 'UtranCell',
```

- REDUNDANT_COUNTERS: Optional entry. It accepts array of counter names. Counter name that matches the entry here shall be removed.

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

- COUNTERS_TO_PROCESS: Mandatory entry. This is the array of counter names that this post parser rule is suppose to look into, and extract the value from it.

```
COUNTERS_TO_PROCESS => [ 'UtranCell_id', ],
```

- COUNTERNAMES_TO_REPLACE: Optional entry. This is the array of counter names that is going to replace the specific counter names in COUNTERS_TO_PROCESS.

```
COUNTERNAMES_TO_REPLACE => [ 'region_id', ],
```

- DELETE_ORIGINALS: Optional entry. If this entry is set to true, the original (counter names from COUNTERS_TO_PROCESS) shall be deleted.

```
DELETE_ORIGINALS => 0,
```

A full sample configuration is included below:

```
{
        'RULE_TYPE'               => 'EXTRACT_INFO',
        'RULE_DESC'               => 'Counter name manipulation of
                                      hierarchy data to allow joining
                                      with cell Location level data',
        'INPUT_FILE_DESCRIPTION'  => ['^UtranCell-#-(.*)-#-I.pif'],
        'PRODUCE_PIF'             => 'True',
        'PRODUCE_LIF'             => 0,
```

```
        'EXTRACTION_EXPRESSION'   => '^(.*)$',
        'OUTPUT_BLOCK_NAME'       => 'UtranCell',
        'REDUNDANT_COUNTERS'      => [],
        'COUNTERS_TO_PROCESS'     => [ 'UtranCell_id' ],
        'COUNTERNAMES_TO_REPLACE' => [ 'region_id' ],
        'DELETE_ORIGINALS'        => 0,
},
```

4.3.2 Sample Usage

Given the following input file:

```
## Metrica Parser Intermediate File
##START|HEADER
rncId|mcc|userLabel|RncFunction_id|mnc
1|228|"RncFunction 1"|1|1
##END|HEADER
##START|UtranCell
maximumTransmissionPower|utranCellIubLink_IubLink|UtranCell|primaryCpich
Power|rac|UtranCell_id
414|13151910||270|1|RNC01
414|12031919||270|1|RNC01
404|Iub_DENG||270|1|RNC01
404|12012206||270|1|RNC01
404|Iub_DACE||270|1|RNC01
405|02211909||270|1|RNC01
424|13151910||270|1|RNC01
390|Iub_DENG||270|1|RNC01
##END|UtranCell
```

The UtranCell_id shall be modifying to region_id according to the rule specified in the previous section. The result of the post parser rule is as shown below:

```
## Metrica Parser Intermediate File
##START|HEADER
rncId|mcc|RncFunction_id|userLabel|mnc
1|228|1|"RncFunction 1"|1
##END|HEADER
##START|UtranCell
utranCellIubLink_IubLink|maximumTransmissionPower|UtranCell|region_id|ra
c|primaryCpichPower
13151910|414||RNC01|1|270
12031919|414||RNC01|1|270
Iub_DENG|404||RNC01|1|270
12012206|404||RNC01|1|270
Iub_DACE|404||RNC01|1|270
02211909|405||RNC01|1|270
13151910|424||RNC01|1|270
Iub_DENG|390||RNC01|1|270
##END|UtranCell
```

4.4 MODIFY_BLOCKNAME_ON_RUNTIME

This post parser rule changes the blockname on conditions. Header counters and/or record counters are being compared. Blockname shall change according to the result

of these comparisons (value of header counter is independent to the value of record counter).

The non-standard configuration entries with this rule are listed below.

4.4.1 Rule Configuration

- OUTPUT_BLOCK_NAME: A mandatory scalar field. This is the part of the complete blockname to replace. It shall combine with values from either PREFIX_BLOCKNAME_IF_TRUE or PREFIX_BLOCKNAME_IF_FALSE.

```
OUTPUT_BLOCK_NAME => 'Plug_In_Unit',
```

- PREFIX_BLOCKNAME_IF_TRUE: This is a mandatory scalar field. This is also the prefix of OUTPUT_BLOCK_NAME if the result of configured conditions is true.

```
PREFIX_BLOCKNAME_IF_TRUE => ' RNC_',
```

- PREFIX_BLOCKNAME_IF_FALSE: A mandatory scalar field. This is the prefix of OUTPUT_BLOCK_NAME if the result of configured conditions is false.

```
PREFIX_BLOCKNAME_IF_FALSE => 'NODEB_',
```

- HEADER_COUNTERS_TO_COMPARE: A mandatory array field. Here listed the header counters that are going to be compared.

```
HEADER_COUNTERS_TO_COMPARE => [ 'sn_SubNetwork', 'sn_MeContext' ],
```

- RECORD_COUNTERS_TO_COMPARE: A mandatory array field. This is the list of record counters that are going to be compared.

```
RECORD_COUNTERS_TO_COMPARE => [ ],
```

A full sample configuration is included below:

```
{
    RULE_TYPE => 'MODIFY_BLOCKNAME_ON_RUNTIME',
    RULE_DESC => 'Prefix NODEB_ to blockname if
                  sn_SubNetwork != sn_MeContext,
                  otherwise prefix RNC_ for PlugInUnit',
    INPUT_FILE_DESCRIPTION =>
'ManagedElement_Equipment_Subrack_Slot_PlugInUnit-#-(.*_.*)-#-A-#-
I\.pif',
    OUTPUT_BLOCK_NAME          => 'Plug_In_Unit',
    PREFIX_BLOCKNAME_IF_TRUE   => 'RNC_',
    PREFIX_BLOCKNAME_IF_FALSE  => 'NODEB_',
    HEADER_COUNTERS_TO_COMPARE => [ 'sn_SubNetwork', 'sn_MeContext' ],
    RECORD_COUNTERS_TO_COMPARE => [],
    PRODUCE_PIF                => 'True',
    OUTPUT_FORMAT              => 0,
},
```

4.4.2 Sample Usage

Given the following input file with the above configuration:

```
## Metrica Parser Intermediate File
##START|HEADER
moid_E1PhysPathTerm|nedn_SubNetwork|MeContext|moid_PlugInUnit|moid_Etm1|
gp|sn_SubNetwork|moid_Subrack|moid_Equipment|nedn_MeContext|SubNetwork|S
tartTime|moid_ManagedElement|cbt|mts|Type|moid_Slot|StartDate|sn_MeConte
xt
pp2|RMERNC0095|RM5NODEB4048_statsfile|1|1|900|RMERNC0095|1|1|RM5NODEB404
8||12:45|1|20040127124500Z|20040127130000Z|A|2|27Jan2004|RM5NODEB4048
##END|HEADER
##START|ManagedElement_Equipment_Subrack_Slot_PlugInUnit
mts|moid_Equipment|moid_ManagedElement|moid_Subrack|moid_PlugInUnit|moid
_E1PhysPathTerm|pmEs|pmSes|moid_Etm1|moid_Slot|Record_measurement_count
20040127130000Z|1|1|1|1|pp2|2|1|1|2|10
20040127130000Z|1|1|1|1|pp1|468|444|1|2|10
20040127130000Z|1|1|1|1|pp8|488198|488198|1|2|10
20040127130000Z|1|1|1|1|pp7|488198|488198|1|2|10
20040127130000Z|1|1|1|1|pp6|488198|488198|1|2|10
20040127130000Z|1|1|1|1|pp5|488198|488198|1|2|10
20040127130000Z|1|1|1|1|pp4|488198|488198|1|2|10
20040127130000Z|1|1|1|1|pp3|2|1|1|2|10
##END|ManagedElement_Equipment_Subrack_Slot_PlugInUnit
```

Since the value of sn_SubNetwork != sn_MeContext, the blockname shall be changed as below (highlighted):

```
## Metrica Parser Intermediate File
##START|HEADER
moid_E1PhysPathTerm|nedn_SubNetwork|MeContext|moid_PlugInUnit|moid_Etm1|
gp|sn_SubNetwork|moid_Subrack|moid_Equipment|nedn_MeContext|SubNetwork|S
tartTime|moid_ManagedElement|cbt|mts|Type|moid_Slot|StartDate|sn_MeConte
xt
pp2|RMERNC0095|RM5NODEB4048_statsfile|1|1|900|RMERNC0095|1|1|RM5NODEB404
8||12:45|1|20040127124500Z|20040127130000Z|A|2|27Jan2004|RM5NODEB4048
##END|HEADER
##START|NODEB_Plug_In_Unit
mts|moid_Equipment|moid_ManagedElement|moid_Subrack|moid_PlugInUnit|moid
_E1PhysPathTerm|pmEs|pmSes|moid_Etm1|moid_Slot|Record_measurement_count
20040127130000Z|1|1|1|1|pp2|2|1|1|2|10
20040127130000Z|1|1|1|1|pp1|468|444|1|2|10
20040127130000Z|1|1|1|1|pp8|488198|488198|1|2|10
20040127130000Z|1|1|1|1|pp7|488198|488198|1|2|10
20040127130000Z|1|1|1|1|pp6|488198|488198|1|2|10
20040127130000Z|1|1|1|1|pp5|488198|488198|1|2|10
20040127130000Z|1|1|1|1|pp4|488198|488198|1|2|10
20040127130000Z|1|1|1|1|pp3|2|1|1|2|10
##END|NODEB_Plug_In_Unit
```

5. Appendix A - 3GPP XML DTD Details

From [3GPP TS 32.401], a 3GPP compliant XML shall conform to the DTD as shown below:

```
<!-- MeasDataCollection.dtd version 2.0-->
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT mdc (mfh, md*, mff)>
<!ELEMENT mfh (ffv, sn, st, vn, cbt)>
<!ELEMENT md (neid, mi*)>
<!ELEMENT neid (neun, nedn, nesw?)>
<!ELEMENT mi (mts, jobid?, gp, rp?, mt*, mv*)>
<!ELEMENT mv (moid, r*, sf?)>
<!ELEMENT mff (ts)>
<!ELEMENT jobid (#PCDATA)>
<!ELEMENT rp (#PCDATA)>
<!ELEMENT ts (#PCDATA)>
<!ELEMENT sf (#PCDATA)>
<!ELEMENT r (#PCDATA)>
<!ATTLIST r p CDATA "">
<!ELEMENT mt (#PCDATA)>
<!ATTLIST mt p CDATA "">
<!ELEMENT moid (#PCDATA)>
<!ELEMENT gp (#PCDATA)>
<!ELEMENT mts (#PCDATA)>
<!ELEMENT nedn (#PCDATA)>
<!ELEMENT neun (#PCDATA)>
<!ELEMENT nesw (#PCDATA)>
<!ELEMENT cbt (#PCDATA)>
<!ELEMENT vn (#PCDATA)>
<!ELEMENT st (#PCDATA)>
<!ELEMENT sn (#PCDATA)>
<!ELEMENT ffv (#PCDATA)>
<!-- end of MeasDataCollection.dtd -->
```

6. Appendix B – Sample 3GPP XML

```
<?xml version="1.0" encoding="UTF8"?>
<!DOCTYPE mdc SYSTEM "32.104-01.dtd">
<mdc xmlns:HTML="http://www.w3.org/TR/REC-xml">
      <mfh>
            <ffv>1</ffv>
            <sn>G3ManagedElement=SGSN-OAM32</sn>
            <st>SGSN</st>
            <vn>Nortel Networks</vn>
            <cbt>20011004091500</cbt>
      </mfh>
      <md>
            <neid>
                        <neun>OAM32</neun>
                  <nedn>G3ManagedElement=SGSN-OAM32</nedn>
            </neid>
            <mi>
                        <mts>20011004093000</mts>
                        <gp>900</gp>
                        <mt>VS.SGSN.actualRate</mt>
                        <mt>VS.SGSN.customerIdentifier</mt>
                        <mt>VS.SGSN.provRate</mt>
                        <mt>VS.SGSN.rxAvgCellRate</mt>
                  <mv>
                              <moid>AtmInterface=80</moid>
                              <r>353207</r>
                              <r>1</r>
                              <r>353207</r>
                              <r>578</r>
                  </mv>
            </mi>
      </md>
      <mff>
            <ts>20011004094500</ts>
      </mff>
</mdc>
```

Table 2 – Sample 3GPP compliant XML performance data file contents.