

# **Marconi OpticalCore Gateway User Guide**

Gateway Release: 3.4.0

Date: 10 March 2008

---

## Contents

<b>CONTENTS</b>	<b>2</b>
<b>REFERENCES</b>	<b>3</b>
<b>GLOSSARY</b>	<b>3</b>
<b>PREFACE</b>	<b>4</b>
ABOUT THIS GUIDE	4
<b>1. OVERVIEW</b>	<b>5</b>
1.1 THE GATEWAY FRAMEWORK	5
1.2 MARCONI OPTICALCORE GATEWAY KIT OVERVIEW	5
1.2.1 Data Types	5
1.2.2 Data Version Support	5
1.2.3 Data/File Formats	6
<b>2. ENGINE RULES AND CONFIGURATION</b>	<b>9</b>
2.1 RULE_TYPE	9
2.1.1 Rule Configuration	9
<b>3. POST PARSER RULES AND CONFIGURATION</b>	<b>13</b>
<b>4. TECH PACK SUPPORT</b>	<b>13</b>
<b>5. INSTALLATION SPECIFIC INFORMATION</b>	<b>13</b>

## **References**

[Gateway Framework User Guide]

## **Glossary**

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

## Preface

### About this Guide

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

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

## 1. Overview

### 1.1 The Gateway Framework

The Marconi OpticalCore Gateway Kit uses the Gateway Framework as a container for the execution of its engine and post parser stages. The Gateway Framework and Marconi OpticalCore Gateway Kit are de-coupled into two separate installations. The Gateway Framework consists of a library of perl modules that provide functionality such as:

- a container for the execution of the Marconi OpticalCore Gateway Kit and Post Parser rules for of data transformation
- Intermediate (PIF) and output data (LIF,CSV,XML) storage and management
- logging utilities
- cleanup and crash recovery
- statistics gathering

The Marconi OpticalCore Gateway Kit simply plugs into the Gateway Framework and provides the functionality to parse the Marconi OpticalCore XML format.

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

Only Marconi OpticalCore Gateway Kit configuration details will be described in this document.

### 1.2 Marconi OpticalCore Gateway Kit Overview

#### 1.2.1 Data Types

The *Marconi OpticalCore Gateway Kit* can be configured to parse the Marconi OpticalCore XML format.

#### 1.2.2 Data Version Support

These are examples of systems currently using the Marconi OpticalCore Gateway Kit

Vendor Subsystem	Release	Format
Marconi OpticalCore	MV36_Core2	XML format

### 1.2.3 Data/File Formats

The *Marconi OpticalCore Gateway Kit* was developed to support the proprietary XML format of Marconi OpticalCore based raw files. The raw files consist of two parts; the first part of the XML files describes the information regarding the network elements and the second part is where the counters are located.

The XML tag element is used to identify the types of network element and the IDs and counters can exist as attributes or text data. The following table will describe each network element and its XML format:

Object	XML Format
EM	<pre>&lt;EM EMId="0" Id="EM-0"&gt;   &lt;EMName CORE="2"   version="15.3.2"&gt;soo&lt;/EMName&gt; &lt;/EM&gt;</pre> <p>EM_ID (from the attribute 'Id' in the XML element EM) e.g. EM_ID = EM-0</p> <p>EM_CORE (from the attribute 'CORE') e.g. EM_CORE = 2</p> <p>EM_VERSION (from the attribute 'version') e.g. EM_VERSION = 15.3.2</p>
SDH_Network_Element	<pre>&lt;NE Id="NE-0-99" NEIdOnEM="99" emId="EM-0"&gt;   &lt;NEIdentity&gt;     &lt;NEName longName="KUKUSH-1 5 2"&gt;       &lt;NEShortName&gt;KUKUSH- 1.6&lt;/NEShortName&gt;       &lt;NESuffix&gt;6&lt;/NESuffix&gt;     &lt;/NEName&gt;     &lt;EMPlugin NEBasicTypeId="20" version="11.4.1"&gt;SMA&lt;/EMPlugin&gt;     &lt;NEType NEModel="" NETypeId="76"&gt;SMA1/4c [Series 3]&lt;/NEType&gt;</pre> <p>NE_ID (from the attribute 'Id' in the XML element NE) e.g. NE_ID = NE-0-99</p> <p>EM_ID (from the attribute 'emId') e.g. EM_ID = EM-0</p> <p>NE_NAME (from the attribute 'longname' in the XML element NEName) e.g. NE_NAME = KUKUSH-1_5_2</p> <p>NE_TYPE (from the XML element NEType) e.g. NE_TYPE = SMA1_4c_Series 3</p>

Object	XML Format
Marconi Shelf	<pre data-bbox="828 243 1377 359">&lt;Shelf Id="SH-0-99-0" ShelfIdOnEM="0" ShelfIdOnNM="1" emId="EM-0" neId="NE-0-99"&gt;   &lt;ShelfIdentity shelfTypeId="0"&gt; -- &lt;/ShelfIdentity&gt; &lt;/Shelf&gt;</pre> <p data-bbox="828 411 1308 485">SHELF_ID (from the attribute 'Id' in the XML element Shelf) e.g. SHELF_ID = <a href="#">SH-0-99-0</a></p> <p data-bbox="828 516 1235 590">SHELF_ID_On_EM (from the attribute 'ShelfIdOnEM') e.g. SHELF_ID_ON_EM = <a href="#">0</a></p> <p data-bbox="828 621 1235 695">SHELF_ID_On_NM (from the attribute 'ShelfIdOnNM') e.g. SHELF_ID_ON_NM = <a href="#">1</a></p> <p data-bbox="828 726 1192 779">EM_ID (from the attribute 'emId') e.g. EM_ID = <a href="#">EM-0</a></p> <p data-bbox="828 810 1183 863">NE_ID (from the attribute 'neId') e.g. NE_ID = <a href="#">NE-0-99</a></p> <p data-bbox="828 894 1271 957">SHELF_IDENTITY (from the XML element ShelfIdentity) e.g. SHELF_IDENTITY = <a href="#">--</a></p> <p data-bbox="828 989 1325 1041">SHELF_TYPE (from the attribute 'shelfTypeId') e.g. SHELF_TYPE = <a href="#">0</a></p>
Marconi_Equipment_Card	<pre data-bbox="828 1073 1365 1220">&lt;Equip Id="EQ-0-99-0-0-2" emId="EM-0" neId="NE-0-99" shelfId="SH-0-99-0"&gt;   &lt;EquipIdentity cardId="2" cardTypeId="1793" physicalSlotId="0"&gt;STM-1 G703 Electrical LWA&lt;/EquipIdentity&gt; &lt;/Equip&gt;</pre> <p data-bbox="828 1251 1304 1325">EQUIP_ID (from the attribute 'Id' in the XML element Equip) e.g. EQUIP_ID = <a href="#">EQ-0-99-0-0-2</a></p> <p data-bbox="828 1356 1192 1409">EM_ID (from the attribute 'emId') e.g. EM_ID = <a href="#">EM-0</a></p> <p data-bbox="828 1440 1183 1493">NE_ID (from the attribute 'neId') e.g. NE_ID = <a href="#">NE-0-99</a></p> <p data-bbox="828 1524 1247 1577">SHELF_ID (from the attribute 'shelfId') e.g. SHELF_ID = <a href="#">SH-0-99-0</a></p> <p data-bbox="828 1608 1266 1682">EQUIP_IDENTITY (from the XML element EquipIdentity) e.g. EQUIP_IDENTITY = <a href="#">STM-1_G703_Electrical_LWA</a></p>

Object	XML Format
SDH Entity	<pre data-bbox="828 291 1380 651">&lt;Source Id="SO-0-99-0-0-2-2-1-0-0-0-0" emId="EM-0" equipId="EQ-0-99-0-0-2" neId="NE-0-99" shelfId="SH-0-99-0"&gt;   &lt;SourceInfo schemeName="SDH"&gt;     &lt;SchemeSDH j="0" k="0" l="0" m="0" port="1"/&gt;   &lt;/SourceInfo&gt; &lt;/Source&gt;  &lt;Entity Id="EN-0-99-0-0-2-2-1-0-0-0-0-0" emId="EM-0" equipId="EQ-0-99-0-0-2" neId="NE-0-99" shelfId="SH-0-99-0" sourceId="SO-0-99-0-0-2-2-1-0-0-0-0"&gt;  &lt;EntityIdentity entityTypeId="0"&gt;MS Near End &lt;/EntityIdentity&gt;</pre> <p data-bbox="828 661 1380 745">ENTITY_ID (from the attribute 'Id' in the XML element Entity) e.g. ENTITY_ID = <a href="#">EN-0-99-0-0-2-2-1-0-0-0-0-0</a></p> <p data-bbox="828 766 1380 850">EQUIP_ID (from the attribute 'equipId' in the XML element Entity) e.g. EQUIP_ID = <a href="#">EQ-0-99-0-0-2</a></p> <p data-bbox="828 871 1380 955">NE_ID (from the attribute 'neId' in the XML element Entity) e.g. NE_ID = <a href="#">NE-0-99</a></p> <p data-bbox="828 976 1380 1060">SHELF_ID (from the attribute 'shelfId' in the XML element Entity) e.g. SHELF_ID = <a href="#">SH-0-99-0</a></p> <p data-bbox="828 1081 1380 1333">Based on the attribute 'sourceId' in the XML element Entity, retrieve the corresponding attributes 'port', 'j', 'k', 'l', 'm' from the XML element Source. e.g. PORT_ID = 1 J = 0 K = 0 L = 0 M = 0</p> <p data-bbox="828 1354 1380 1438">ENTITY_TYPE (from the XML element EntityIdentity) e.g. ENTITY_TYPE = <a href="#">MS_Near_End</a></p>

## 2. Engine Rules and Configuration

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

### 2.1 RULE\_TYPE

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

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

#### 2.1.1 Rule Configuration

The following details the vendor specific rule entries for the Marconi OpticalCore engine rule.

##### 2.1.1.1 Mandatory Configuration Entries

- RULE\_TYPE:

```
RULE_TYPE    =>    'MARCONI_OPTICALCORE',
```

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

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

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

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

- INPUT\_DIR\_DEPTH: Specification for how deep the gateway framework will search for files in directory trees in the specified property "IN\_DIR". 0 is the default. Mandatory entry.

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

- NUMBER\_OF\_FILES\_TO\_PROCESS: Specification for how many files from "IN\_DIR" the gateway framework will attempt to run through the gateway in any single run. E.g. 20 = 20 files at a time. This parameter is a tuning parameter. Mandatory entry.

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

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

Valid options: YOUNGEST\_FIRST, OLDEST\_FIRST, DIRECTORY\_ORDER (comment out this configuration option to speed up the first stage of the parser at expense of data presentation contiguity to the loader). This parameter is a tuning parameter.

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

- **DIRECTORY\_HEADER\_FIELDS:** Specification for the directory name or part of the directory name to appear as a counter in the PIF header.

```
DIRECTORY_HEADER_FIELDS => {
    NETWORK_ID => '\\\\w+\\\\w{7}_\\\\w+\\\\w{6}_w+'
}
```

- **FILENAME\_HEADER\_FIELDS:** Specification for the raw filename or part of the raw filename to appear as a counter in the PIF header.

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

- **HEADER\_FIELDS\_FOR\_PIF\_FILENAME:** Specification for value of counters in the PIF header to be appended to the initial PIF filename.

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

- **DATA\_FIELDS\_FOR\_PIF\_FILENAME:** Specification for value of counters in the PIF counter blocks to be appended to the initial PIF filename.

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

- **XML\_HANDLER\_MODULE:** Specification on the handler to be used to process the Marconi OpticalCore XML files. The handler will be a perl module and the name of the module is to be provided.

```
XML_HANDLER_MODULE => "MARCONI_OPTICALCORE_HANDLER",
```

- **OUTPUT\_PIF\_FILENAME\_START:** Specification of the string to be appended to the start of every PIF file produced by the parser.

```
OUTPUT_PIF_FILENAME_START => 'EMOS',
```

- **NETWORK\_ELEMENT\_TAGS:** An array containing all the element tags that the parser will look for and extract all the attributes and text data. Attributes will be extracted as key/value pairs while text data will use the tag name as the key.

Example: <NE id=NE-0-99>0<NE/>; id= NE-0-99 and NE=0

```
NETWORK_ELEMENT_TAGS => [qw(EM NE Shelf Equip Source)],
```

- **NETWORK\_ELEMENT\_UNIQUE\_KEYS:** Each network element needs a unique identifier, this option enables the user to specify which attributes contains this unique identifier.

```
NETWORK_ELEMENT_UNIQUE_KEYS => "Id",
```

- **OUTPUT\_BLOCK\_ELEMENT\_NAME:** The counters for the Marconi OpticalCore XML is specific to one type of network element. This option specifies for which network element are the counters being reported for.

```
OUTPUT_BLOCK_ELEMENT_NAME => "Entity",
```

- **OUTPUT\_BLOCK\_ELEMENT\_TYPE:** A network element might contain a type being associated with it. This option specifies where is the tag that contains the type information.

```
OUTPUT_BLOCK_ELEMENT_TYPE => "EntityIdentity",
```

- **OUTPUT\_BLOCK\_ELEMENT\_GROUP\_ELEMENT:** The counters for Marconi OpticalCore are reported in groups and one group represents one timestamp. This option specifies the tag that groups the counters together.

```
OUTPUT_BLOCK_COUNTER_GROUP_ELEMENT => "Details",
```

- **OUTPUT\_BLOCK\_COUNTER\_ELEMENT:** Specification of the tag that contains the information of a counter.

```
OUTPUT_BLOCK_COUNTER_ELEMENT => "Counter",
```

- **OUTPUT\_BLOCK\_TIMESTAMP\_ELEMENT:** Specification of the tag representing the timestamp of the counters.

```
OUTPUT_BLOCK_TIMESTAMP_ELEMENT => "TimeStamp",
```

### 2.1.1.2 Optional Configuration Entries

- **HEADER\_NAME\_MAPPING:** A hash that contains the mappings of existing counter names to new counter names in the header of a PIF.

```
HEADER_NAME_MAPPING => {  
    emId => 'EM_ID',  
    CORE => 'EM_CORE',  
    version => 'EM_VERSION',  
    neId => 'NE_ID',  
}
```

- **COUNTER\_NAME\_MAPPING:** A hash that contains the mappings of existing counter names to new counter names in the data portion of a PIF.

```
COUNTER_NAME_MAPPING => {  
    Id => 'ENTITY_ID',  
    EntityIdentity => 'ENTITY_TYPE' }  
}
```

### **3. Post Parser Rules and Configuration**

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

### **4. Tech Pack Support**

Tech pack support is included in the Marconi OpticalCore Gateway for the following Performance Manager solutions.

- Transmission Marconi OpticalCore MV36\_Core2

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

### **5. Installation Specific Information**

None.