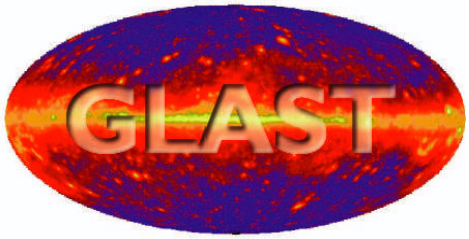

LAT Flight Software

LATC Users' Guide

Type:	Users' Guide
Version:	V7-0-0
Author:	J. Swain
Created:	15 March 2005
Updated:	2 August 2005
Printed:	15 August 2005

This document provides an introduction to the LAT configuration package, LATC.

Contents

0 Introduction

0.0 Scope

This document provides a guide for users of the LAT Configuration utility (LATC). Maintainers of the LATC FSW package should refer to the more detailed *LATC Design*.

0.1 References

1. LAT-TD-00606 *LAT Inter-module Communications: A reference manual*.
2. LAT-TD-01380 *LAT Communication Board Driver: Software architecture and interfaces*.
3. LAT-TD-01547 *The Command/Response Unit: Programming ICD specification*.
4. LAT-TD-01546 *The Event Builder Module: Design specification*.
5. LAT-TD-01545 *The GLT Electronics Module: Programming ICD specification*.
6. LAT-TD-00639 *The ACD Electronics Module (AEM): A primer*.
7. LAT-SS-00363 *ACD-LAT ICD: Mechanical, thermal and electrical*.
8. LAT-SS-00363 *LAT Dataflow Specification: ACD-AEM interface*.
9. LAT-TF-00605 *The Tower Electronics Module (TEM): Programming ICD specification*.
10. *The GLAST acronym list*.

0.2 Request For Comments

Please post corrections or questions as replies to the SNITZ release announcement for the relevant LATC version. Failing this, email jswain@slac.stanford.edu.

0.3 Acronyms

LAT – Large Area Telescope

SIU – Spacecraft Interface Unit

PDU – Power Distribution Unit

EPU – Event Processing Unit

TEM – Tower Electronics Module

CRU – Command Response Unit

ACD – Anti-Coincidence Detector

DAQ – Data AcQuisition

DAB – DAQ Board

LATp – LAT (communications) protocol

EEPROM – Electronic Erasable Programmable Read Only Memory

LCB – LAT Communications Board

GEM – Global trigger Electronics Module

EBM – Event Builder Module

AEM – ACD Electronics Module

FREE – FRont End Electronics

P/R – Primary/Redundant

C/R – Command/Response

1 Introduction

The behaviour of the LAT in response to external stimuli (physics events) is modified by setting the various registers of the LAT. These registers contain approximately four billion bits of data. The LAT Configuration utility (LATC) provides a framework for the segmentation and application of this configuration data.

The LAT is made up of four subsystems, ACD, CAL, TKR and DAQ. Each of these subsystems is further divided into components. The AEM, ARC and AFE for the ACD; the TEM and TIC shared by the CAL and TKR; the CCC, CRC and CFE for the CAL; the TCC, TRC and TFE for the TKR; and the PDU, CRU, EBM and GEM (further divided) for the DAQ. Some of the registers of the LAT are controlled by applications such as housekeeping (LHK) and the power and intialisation software (PIG). These registers are not exposed through LATC. In some cases this means that entire components are not included in LATC. Specifically, there is no mention of PDU, CRU or EBM.

The tracker front ends constitute the bulk of the configuration data. Due to the design of the tracker they also require special handling. The TFEs are arranged into layers and can be configured to accept commands and transmit data through the TRC at either end of the layer. It is beneficial, therefore, to invent a psuedo-component called SPT (split) that corresponds to each layer and identifies which TFE(s) communicate with which TRC. The TFE contains three large bit

masks and one register containing DAC values. Given the orthogonal nature of these two types of data, another pseudo-component called TDC was invented. This component contains the two seven bit DAC values for the TFE.

There may be many instances of each of these components. For example, the TEM component has sixteen instances on the LAT. One of the design assumptions of LATC was that many of these instances would be configured in the same way. Therefore, LATC contains the concept of a default (DFT) or "golden" configuration. This default configuration contains one block of data for each component, specifying the default values of that component's registers. When the LAT is configured this default configuration is applied first. Any instances that have non-default (sometimes termed *exceptional*) configurations are then reloaded with new values.

The life cycle of a LAT configuration begins with the specification of the register values in an XML format. This XML configuration is processed by a host-based LATC utility, `LATC_parser`. The output of this utility is one or more binary configuration files. If default values have been specified in the XML then a default binary file can be produced. Files for the AEM and GEM can also be produced, containing a subset of the default configuration. If non-default values have been specified then a binary file for each component with a non-default instance will also be produced. If, for example, TEM 3 and TEM 4 both have exceptional configurations then a single TEM binary file will be produced containing a block of data for TEM 3 and a block of data for TEM 4.

The parser may be invoked many times with many different XML files. The binary files can then be combined into a single configuration by listing the files that make up that configuration in a Configuration Master file. It is this file that will eventually be presented to LATC on the instrument. One binary file may appear in different configurations.

Once the binary files and the Configuration Master have been uploaded to the LAT in the normal manner then the LAT can be configured. The embedded LATC utilities are intialised as part of the SIU boot sequence, and all the resources needed are allocated at that time. When data collection (either physics acquisition or charge injection calibration) is initiated a LAT Configuration Master file will be passed as a parameter to the data collection task. This task will then use the `LATC_configure` function to read the all the binary files specified by the Configuration Master file and build an in-memory model of the LAT registers. This model is then used to generate a series of register load commands that will drive the LAT into the required configuration.

Some time later, the `LATC_capture` command can be used to read the LAT registers and build another in-memory model of the LAT, this time containing the current LAT configuration. This model can be compared with the model created from the binary files and can also be passed to the SSR for downlink to the ground.

Finally, LATC will provide a host-based utility to access the captured LAT configuration.

2 Preparation

The input to LATC comes in the form of one or more XML files that conform to the LATC DTD (found at /afs/slac/g/glast/flight/ICS/source/LATC/prod/LATC/latc.dtd). The LATC package provides and XML parser running in the host environment to convert this XML file into one or more binary files that form the configuration. The parser is run from the command line.

```
LATC_parser -i=file0.xml[,file1.xml,...,filen.xml] -o=/tmp/latc/ [-f=DFT,TEM,....] [-v]
```

The application takes between two and four command line arguments. The first (-i) provides a comma-separated list of input files. There must be at least one, but may be more. The second (-o) specifies an output stub. This gives the directory and (optionally) the start of the output filenames. The tail of the filenames is fixed by LATC to be XXX.lcb, where XXX is one of fifteen TLAs used by LATC to identify the LAT components. The TLAs are

1. DFT : Default
2. GEM : The four blocks of the GEM (WIN, TAM, ROI, TIE) configured by LATC.
3. AEM : The AEM common controller.
4. ARC
5. AFE
6. TEM : The TEM common controller
7. TIC
8. CCC
9. CRC
10. CFE
11. TCC
12. TRC
13. SPT : Layer splits for the tracker front ends.
14. TFE
15. TDC : Tracker thresholds.

The third argument uses a comma-separated list of the same TLAs to filter the output. Only those components that appear in the list will have corresponding binary files. Of course, some of the components may not generate any binary data anyway. If the filter argument is omitted then all files will be emitted. The fourth argument is generally only of use during LATC development. It causes the parser to produce a large amount of output to stdout.

Optionally, LATC can also be configured to ignore part of the LAT during capture and verification operations. A binary file containing a map identifying the portions of the LAT to be ignored is generated using another XML parser.

```
LATC_map -i=file0.xml[,file1.xml,...,filen.xml] -o=file.map [-v]
```

The application takes two or three command line arguments. The first (-i) provides a comma-separated list of input files. There must be at least one, but may be more. The second (-o) specifies an output file. The third argument is generally only of use during LATC development. It causes the parser to produce a large amount of output to stdout.

2.0 Configuration XML Vocabulary

The XML vocabulary used by LATC is presented below in the form of sample XML files. It is broken into three sections, GEM, AEM and TEM. In practice the XML configuration files can be segmented any way that is convenient for the user. Note that the LATC XML parser breaks the XML specification for multiply defined tags. The last value specified is the one that is used, whereas the XML specification requires that the first value be used.

Each of the sample files beings with the standard opening lines, identifying the XML version used and the location of the document type definition applicable to these XML files. The tags in capitals are component tags. Component tags can contain zero or more register tags or zero or more component tags. Where there is more than one instance of a component on the LAT then the component has an attribute, ID, that identifies the instance to which the configuration applies. There is a keyword ID, BCAST, that indicates the configuration specified is the default configuration. Note that BCAST instances can only contain other BCAST instances and can only be contained by other BCAST instances. For simplicity the example files given below contain only BCAST instances. The ID attribute is usually an integer, for example [0,16] for the TEM. For the SPT component, however, the +/-x/y[0,3] layer identification strings are used.

Register tags can contain zero or more field tags or a character element (hexadecimal or decimal integer) specifying the value of the register. Field tags can only contain a character element specifying the value of the field. Character elements are indicated by three hash symbols (###).

These sample XML files are given as examples only. For an up-to-date list of valid XML tags the user should check either the DTD (stored at /afs/slac/g/glast/flight/ICS/source/LATC/prod/LATC/latc.dtd) or the XML files from which it is derived (/afs/slac/g/glast/flight/ICS/source/LATC/prod/lrd/lrd_*.xml).

2.0.0 GEM

The GEM component contains four component tags, WIN, TAM, ROI and TIE, and no registers. None of the components take an ID attribute because they are singletons.

The `engine_n` register tags of the TAM contain field tags, but for brevity only `engine_0` has these explicitly listed in the sample XML below. Similarly, each of the ROI register tags contains one or two field tags, but only one set of these is listed explicitly.

```
    <?xml version='1.0' standalone='yes' ?>
    <!DOCTYPE LATC_XML SYSTEM
"/afs/slac/g/glast/flight/ICS/source/LATC/prod/LATC/latc.dtd">
    <LATC_XML>
      <GEM>
        <WIN>
          <window_width>
            <width> ### </width>
          </window_width>
        </WIN>
        <TAM>
          <engine_0>
            <prescale     >  ### </prescale     >
            <inhibit      >   ### </inhibit      >
            <calstrobe    >   ### </calstrobe    >
            <tack         >   ### </tack         >
            <four_range   >   ### </four_range   >
            <zero_suppress>   ### </zero_suppress>
            <marker       >   ### </marker       >
            <destination  >   ### </destination  >
          </engine_0
          <engine_1> ### <engine_1>
          <engine_2> ### <engine_2>
          <engine_3> ### <engine_3>
          <engine_4> ### <engine_4>
          <engine_5> ### <engine_5>
          <engine_6> ### <engine_6>
          <engine_7> ### <engine_7>
          <engine_8> ### <engine_8>
          <engine_9> ### <engine_9>
          <engine_a> ### <engine_a>
          <engine_b> ### <engine_b>
          <engine_c> ### <engine_c>
          <engine_d> ### <engine_d>
          <engine_e> ### <engine_e>
          <engine_f> ### <engine_f>
        </TAM>
        <SCH>
          <conditions_00_07> ### </conditions_00_07>
          <conditions_08_0F> ### </conditions_08_0F>
          <conditions_10_17> ### </conditions_10_17>
          <conditions_18_1F> ### </conditions_18_1F>
          <conditions_20_27> ### </conditions_20_27>
          <conditions_28_2F> ### </conditions_28_2F>
          <conditions_30_37> ### </conditions_30_37>
          <conditions_38_3F> ### </conditions_38_3F>
          <conditions_40_47> ### </conditions_40_47>
          <conditions_48_4F> ### </conditions_48_4F>
          <conditions_50_57> ### </conditions_50_57>
          <conditions_58_5F> ### </conditions_58_5F>
          <conditions_60_67> ### </conditions_60_67>
          <conditions_68_6F> ### </conditions_68_6F>
          <conditions_70_77> ### </conditions_70_77>
          <conditions_78_7F> ### </conditions_78_7F>
          <conditions_80_87> ### </conditions_80_87>
          <conditions_88_8F> ### </conditions_88_8F>
```

```
    <conditions_90_97> ### </conditions_90_97>
    <conditions_98_9F> ### </conditions_98_9F>
    <conditions_A0_A7> ### </conditions_A0_A7>
    <conditions_A8_AF> ### </conditions_A8_AF>
    <conditions_B0_B7> ### </conditions_B0_B7>
    <conditions_B8_BF> ### </conditions_B8_BF>
    <conditions_C0_C7> ### </conditions_C0_C7>
    <conditions_C8_CF> ### </conditions_C8_CF>
    <conditions_D0_D7> ### </conditions_D0_D7>
    <conditions_D8_DF> ### </conditions_D8_DF>
    <conditions_E0_E7> ### </conditions_E0_E7>
    <conditions_E8_EF> ### </conditions_E8_EF>
    <conditions_F0_F7> ### </conditions_F0_F7>
    <conditions_F8_FF> ### </conditions_F8_FF>
  </SCH>
  <ROI>
    <r_000_001>
      <t_000> ### </t_000>
      <t_001>    ### </t_001>
    </r_000_001>
    <r_002_003> ### </r_002_003>
    <r_004_010> ### </r_004_010>
    <r_011_012> ### </r_011_012>
    <r_013_014> ### </r_013_014>
    <r_020_021> ### </r_020_021>
    <r_022_023> ### </r_022_023>
    <r_024_030> ### </r_024_030>
    <r_031_032> ### </r_031_032>
    <r_033_034> ### </r_033_034>
    <r_040_041> ### </r_040_041>
    <r_042_043> ### </r_042_043>
    <r_044    > ### </r_044    >
    <r_100    > ### </r_100    >
    <r_101_102> ### </r_101_102>
    <r_103_104> ### </r_103_104>
    <r_110_111> ### </r_110_111>
    <r_112_113> ### </r_112_113>
    <r_114_120> ### </r_114_120>
    <r_121_122> ### </r_121_122>
    <r_123_124> ### </r_123_124>
    <r_130    > ### </r_130    >
    <r_200    > ### </r_200    >
    <r_201_202> ### </r_201_202>
    <r_203_204> ### </r_203_204>
    <r_210_211> ### </r_210_211>
    <r_212_213> ### </r_212_213>
    <r_214_220> ### </r_214_220>
    <r_221_222> ### </r_221_222>
    <r_223_224> ### </r_223_224>
    <r_230    > ### </r_230    >
    <r_300    > ### </r_300    >
    <r_301_302> ### </r_301_302>
    <r_303_304> ### </r_303_304>
    <r_310_311> ### </r_310_311>
    <r_312_313> ### </r_312_313>
    <r_314_320> ### </r_314_320>
```

```
          <r_321_322> ### </r_321_322>
          <r_323_324> ### </r_323_324>
          <r_330    > ### </r_330    >
          <r_400    > ### </r_400    >
          <r_401_402> ### </r_401_402>
          <r_403_404> ### </r_403_404>
          <r_410_411> ### </r_410_411>
          <r_412_413> ### </r_412_413>
          <r_414_420> ### </r_414_420>
          <r_421_422> ### </r_421_422>
          <r_423_424> ### </r_423_424>
          <r_430    > ### </r_430    >
          <r_500    > ### </r_500    >
          <r_501_502> ### </r_501_502>
          <r_503_600> ### </r_503_600>
          <r_601_602> ### </r_601_602>
          <r_603    > ### </r_603    >
        </ROI>
        <TIE>
          <towers_0_3    > ### </towers_0_3    >
          <towers_4_7    > ### </towers_4_7    >
          <towers_8_b    > ### </towers_8_b    >
          <towers_c_f    > ### </towers_c_f    >
          <acd_cno       > ### </acd_cno       >
          <tiles_000_013 > ### </tiles_000_013 >
          <tiles_014_023 > ### </tiles_014_023 >
          <tiles_033_NA3 > ### </tiles_033_NA3 >
          <tiles_100_113 > ### </tiles_100_113 >
          <tiles_114_NA5 > ### </tiles_114_NA5 >
          <tiles_200_213 > ### </tiles_200_213 >
          <tiles_214_NA7 > ### </tiles_214_NA7 >
          <tiles_300_313 > ### </tiles_300_313 >
          <tiles_314_NA9 > ### </tiles_314_NA9 >
          <tiles_400_413 > ### </tiles_400_413 >
          <tiles_414_NA1 > ### </tiles_414_NA1 >
          <tiles_600_NA10> ### </tiles_600_NA10>
          <tower_busy    > ### </tower_busy    >
        </TIE>
      </GEM>
    </LATC_XML>
```

2.0.1 AEM

The AEM component contains the ARC component tag, which takes an ID attribute, and several register tags. The ARC component tag, in turn, contains the AFE component tag (with an ID attribute) and several register tags. The AFE contains only register tags.

```
    <?xml version=1.0 standalone=yes ?>
    <!DOCTYPE LATC_XML SYSTEM
"/afs/slac/g/glast/flight/ICS/source/LATC/prod/LATC/latc.dtd">
    <LATC_XML>
      <AEM>
        <aem_configuration>
          <data_mask > ### </data_mask>
        </aem_configuration>
        <trgseq>
          <cal_strobe> ### </cal_strobe>
          <tack      > ### </tack      >
        </trgseq>
        <ARC ID = '[0,11]|BCAST[1]'>
          <pha_en_0 > ### </pha_en_0>
          <veto_en_0> ### </veto_en_0>
          <pha_en_1>
            <enable> ### </enable>
          </pha_en_1>
          <veto_en_1>
            <enable> ### </enable>
          </veto_en_1>
          <max_pha>
           <number> ### </number>
          </max_pha>
          <pha_threshold_0> ### </pha_threshold_0>
          <pha_threshold_1>  ### </pha_threshold_1>
          <pha_threshold_2>  ### </pha_threshold_2>
          <pha_threshold_3>  ### </pha_threshold_3>
          <pha_threshold_4>  ### </pha_threshold_4>
          <pha_threshold_5>  ### </pha_threshold_5>
          <pha_threshold_6>  ### </pha_threshold_6>
          <pha_threshold_7>  ### </pha_threshold_7>
          <pha_threshold_8>  ### </pha_threshold_8>
          <pha_threshold_9>  ### </pha_threshold_9>
          <pha_threshold_1>  ### </pha_threshold_1>
          <pha_threshold_1>  ### </pha_threshold_1>
          <pha_threshold_1>  ### </pha_threshold_1>
          <pha_threshold_1>  ### </pha_threshold_1>
          <pha_threshold_1>  ### </pha_threshold_1>
          <pha_threshold_1>  ### </pha_threshold_1>
          <pha_threshold_1>  ### </pha_threshold_1>
          <pha_threshold_1>  ### </pha_threshold_1>
          <acd_tacq       >  ### </acd_tacq       >
          <AFE ID = '[0,17]|BCAST'>
            <config_reg>
              <high_TCI          > ### </high_TCI          >
              <HLD_discriminator > ### </HLD_discriminator >
              <veto_discriminator> ### </veto_discriminator>
              <high_gain_range   > ### </high_gain_range   >
              <manual_gain_range > ### </manual_gain_range >
              <TCI               > ### </TCI               >
             </config_reg>
```

[1] This notation indicates that the ID attribute can take any of the integer values from zero to twelve, or the keyword BCAST, subject to the limitation that all nodes in the hierarchy must BCAST if any of them are

```
            <veto_dac>
              <value> ### </value>
            </veto_dac>
            <veto_vernier>
             <value> ### </value>
           </veto_vernier>
           <hld_dac>
             <value> ### </value>
           </hld_dac>
           <bias_dac>
             <value> ### </value>
           </bias_dac>
           <tci_dac>
             <value> ### </value>
           </tci_dac>
         </AFE>
       </ARC>
     </AEM>
   </LATC_XML>
```

2.0.2 TEM

```
    <?xml version='1.0' standalone='yes' ?>
    <!DOCTYPE LATC_XML SYSTEM
"/afs/slac/g/glast/flight/ICS/source/LATC/prod/LATC/latc.dtd">
    <LATC_XML>
      <TEM ID = '[0,15]|BCAST>
        <data_masks>
          <diagnostic> ### </diagnostic>
          <cal_mask  > ### </cal_mask  >
          <tkr_mask  > ### </tkr_mask  >
        </data_masks>
        <tkr_trgseq>
          <calstrobe_delay> ### </calstrobe_delay>
          <tack_delay     >  ### </tack_delay     >
        </tkr_trgseq>
        <cal_trgseq>
          <calstrobe_delay> ### </calstrobe_delay>
          <tack_delay     >  ### </tack_delay     >
        </cal_trgseq>
        <TIC>
          <cal_input_mask>
            <low_energy > ### </low_energy >
             <high_energy> ### </high_energy>
          </cal_input_mask>
          <tkr_layer_enable_0>
            <layer_enable> ### </layer_enable>
          </tkr_layer_enable_0>
          <tkr_layer_enable_1>
            <layer_enable> ### </layer_enable>
          </tkr_layer_enable_1>
          <tkr_out_mask>
            <out_mask> ### </out_mask>
          </tkr_out_mask>
          <tkr_bias_dac>
```

```
          <input> ### </input>
        </tkr_bias_dac>
        <cal_bias_dac>
         <input> ### </input>
        </cal_bias_dac>
      </TIC>
      <CCC ID = '[0,3]|BCAST'>
        <ccc_configuration>
          <data_fifo    > ### </data_fifo    >
          <error_fifo   >  ### </error_fifo   >
          <sum_diag_fifo>  ### </sum_diag_fifo>
           <listen_msb   >  ### </listen_msb   >
           <output_enable>  ### </output_enable>
        </ccc_configuration>
       <layer_mask_0>
          <pos_log_mask > ### </pos_log_mask >
          <pos_le_mask  >  ### </pos_le_mask  >
          <pos_he_mask  >  ### </pos_he_mask  >
          <neg_log_mask >  ### </neg_log_mask >
          <neg_le_mask  >  ### </neg_le_mask  >
          <neg_he_mask  >  ### </neg_he_mask  >
        </layer_mask_0>
        <layer_mask_1>
          <pos_log_mask> ### </pos_log_mask>
          <pos_le_mask > ### </pos_le_mask >
          <pos_he_mask > ### </pos_he_mask >
          <neg_log_mask> ### </neg_log_mask>
          <neg_le_mask > ### </neg_le_mask >
          <neg_he_mask > ### </neg_he_mask >
        </layer_mask_1>
        <ccc_trg_alignment>
          <prim_align> ### </prim_align>
           <stretch   > ### </stretch   >
          <shape_time> ### </shape_time>
        </ccc_trg_alignment>
        <CRC ID = '[0,3]|BCAST'>
         <delay_1> ### </delay_1>
         <delay_2> ### </delay_2>
         <delay_3> ### </delay_3>
         <crc_dac>
          <injection> ### </injection>
         </crc_dac>
          <config> ### </config>
          <CFE ID = '[0,9]|BCAST'>
            <config_0>
              <le_gain_select > ### </le_gain_select >
              <he_gain_select > ### </he_gain_select >
              <le_range_enable> ### </le_range_enable>
              <he_range_enable> ### </he_range_enable>
              <use_first_range> ### </use_first_range>
              <first_range    > ### </first_range    >
              <overwrite      > ### </overwrite      >
            </config_0>
            <config_1>
              <preamp_auto_reset> ### </preamp_auto_reset>
              <le_trigger_enable> ### </le_trigger_enable>
```

```
            <he_trigger_enable> ### </he_trigger_enable>
            <calibration_gain > ### </calibration_gain >
            <calibration_le   > ### </calibration_le   >
            <calibration_he   > ### </calibration_he   >
            </config_1>
             <fle_dac>
               <led    > ### </led    >
             </fle_dac>
             <fhe_dac>
               <hed> ### </hed>
             </fhe_dac>
             <log_acpt>
               <log> ### </log>
             </log_acpt>
           <rng_uld_dac>
             <rng_uld> ### </rng_uld>
           </rng_uld_dac>
            <ref_dac>
              <ref> ### </ref>
            </ref_dac>
        </CFE>
      </CRC>
    </CCC>
    <TCC ID = '[0,11]|BCAST'>
      <tcc_configuration>
        <output_enable> ### </output_enable>
         <cable_length >  ### </cable_length >
        <summary_full >   ### </summary_full >
        <error_full   >   ### </error_full   >
        <data_full    >   ### </data_full    >
      </tcc_configuration>
      <input_mask>
        <mask        > ### </mask        >
         <trigger_mask> ### </trigger_mask>
      </input_mask>
      <tcc_trg_align>
        <shape_time> ### </shape_time>
        <prim_align> ### </prim_align>
      </tcc_trg_align>
      <TRC ID = 'BCAST'>
        <trc_csr>
          <size_write_en> ### </size_write_en>
          <size         > ### </size         >
        </trc_csr>
       </TRC>
     </TCC>
     <SPT ID = '±x/y[0,3]|BCAST'>
       <low > ### </low >
       <high> ### </high>
       <TFE ID = '[0,23]|BCAST'>
         <data_mask  > ### </data_mask  >
         <calib_mask > ### </calib_mask >
        <trig_enable> ### </trig_enable>
         <TDC>
          <tfe_dac>
            <threshold> ### </threshold>
```

```
                <injection> ### </injection>
              </register>
            </TDC>
          </TFE>
        </SPT>
      </TEM>
    </LATC_XML>
```

2.1 Mapping XML Vocabulary

The XML vocabulary used to construct a binary file containing a map of the LAT components to ignore LATC is presented below in the form of sample XML files. As with the configuration XML presented earlier, the mapping files can be segmented any way that is convenient for the user. Note that the LATC mapping XML parser breaks the XML specification for multiply defined tags. The last value specified is the one that is used, whereas the XML specification requires that the first value be used.

Each of the sample files beings with the standard opening lines, identifying the XML version used and the location of the document type definition applicable to these XML files. The only tags that this vocabulary contains are the component tags from the previous section, since the LATC maps operate on the level of components, not registers or field. The only interesting components are those with multiple instances, since the singletons are not mapped. As before, a specific instance is identified by the ID attribute. However, there is no BCAST keyword in the mapping vocabulary.

Each tag can contain one of the two keywords SET or CLEAR, or it can be empty. The keyword SET indicates that the corresponding map bit should be set, while CLEAR indicates that the corresponding map bit should be cleared. If the tag is empty then the corresponding map bit is not touched. Many of the tags also contain other component tags.

These sample XML files are given as examples only. For an up-to-date list of valid XML tags the user should check either the DTD (stored at /afs/slac/g/glast/flight/ICS/source/LATC/prod/LATC/map.dtd) or the XML files from which it is derived (/afs/slac/g/glast/flight/ICS/source/LATC/prod/lrd/lrd_*.xml).

2.1.0 GEM

The GEM is not included in the mapping XML vocabulary, since there is only one of them.

2.1.1 AEM

The AEM itself does not appear in the map, since there is only one of them, but the AEM tag acts as a container for the ARC component tag.

```
    <?xml version=1.0 standalone=yes ?>
    <!DOCTYPE LATC_MAP_XML SYSTEM
"/afs/slac/g/glast/flight/ICS/source/LATC/prod/LATC/map.dtd">
    <LATC_MAP_XML>
      <AEM>
        <ARC ID = '[1,12]'>
          ###
          <AFE ID = '[0,17]'>
            ###
          </AFE>
        </ARC>
      </AEM>
    </LATC_MAP_XML>
```

2.1.2 TEM

Note that there **are** TIC and TDC maps.

```
    <?xml version='1.0' standalone='yes' ?>
    <!DOCTYPE LATC_MAP_XML SYSTEM
"/afs/slac/g/glast/flight/ICS/source/LATC/prod/LATC/latc.dtd">
    <LATC_XML>
      <TEM ID = '[0,15]>
        <TIC>
        </TIC>
        <CCC ID = '[0,3]'>
          <CRC ID = '[0,3]'>
            <CFE ID = '[0,15]'>
           </CFE>
          </CRC>
        </CCC>
        <TCC ID = '[0,7]'>
          <TRC ID = '[0,8]'>
           </TRC>
         </TCC>
         <SPT ID = '±x/y[0,3] '>
           <TFE ID = '[0,23]'>
             <TDC>
            </TDC>
          </TFE>
        </SPT>
      </TEM>
    </LATC_MAP_XML>
```

3 Execution

At the end of the preparation stage the binary configuration files and the configuration master file(s) should be present on the SIU file system. These can now be used to configure the LAT.

3.0 Initialise

LATC will be initialized during FSW startup.

```
unsigned LATC_initialise(void);
```

This function allocates the resources required by LATC.

The only errors that this function can return are allocation errors.

3.1 Cache

The configuration binary files are read from the file system and used to populate an in-memory model of the LAT registers called the *cache*.

```
unsigned LATC_cache(unsigned fid);
```

This function takes the ID of the LAT configuration master file that describes the desired configuration.

This function will return errors if there are problems reading the file(s).

3.2 Configure

The register values contained by the cache are loaded onto the LAT.

```
unsigned LATC_configure(void);
```

LATC generates a set of register load commands that drive the LAT into the desired configuration.

The function will return errors if there are problems loading the LAT register.

3.3 Capture

Some time after the LAT has been configured it will be necessary to verify that the configuration was successful.

```
unsigned LATC_capture(void);
```

LATC generates a set of register read commands that populate a second in-memory model of the LAT with the current configuration.

Timeout errors on the register reads are ignored, but the function will return an error if there is a problem with the LCB during the read process.

3.4 Ignore

In environments where some portions of the LAT are known to be absent (test-stands) or non-functioning it can be undesirable to wait for the read operations to time out.

```
void LATC_ignore(unsigned fid);
```

This function takes the ID of a binary file containing a map of the LAT components indicating those that should be ignored during capture and validation.

This function will return errors if there are problems reading the binary file.

3.5 Verify

A data-collecting task may wish to know if the captured configuration matches the requested configuration.

```
unsigned LATC_verify(void);
```

This function compares the cached in-memory model created during the configure operation with the model created during the capture operation.

This function returns an error if the two in-memory models of the LAT are different.

3.6 Consign

It will occasionally be desirable to transmit the captured configuration to the ground, via the SSR.

```
unsigned LCI_consign(unsigned dst)
```

LATC formats the in-memory mode of the data and sends it to either the SSR (if fid < 32) or to a file with the file ID of *dst*.