

EDDY CER Factory - Design & Development Guide

Software Release – 0.5.2-4
Document Version - 1.20
Publication Date – June 9, 2008

EDDY Development Team
Carnegie Mellon University
5000 Forbes Avenue
Room 215 Cyert Hall
Pittsburgh, Pennsylvania 15213
Eddy-info@lists.andrew.cmu.edu

Chapter

1

Overview

Document Scope

The EDDY CER Factory Design & Development Guide is intended for the following audiences:

- **EDDY Developers**

This group includes software developers who want to learn about and understand how to start developing EDDY Agents using the EDDY CER Factory. A Quick Start guide precedes an in-depth discussion of the CER Factory to get developers up and running with the CER Factory quickly.

- **EDDY Evaluators**

This group includes those who want to learn more about EDDY and particularly the CER Factory and what it accomplishes and what its capabilities are as well as understanding its underlying architecture.

CER Factory Introduction

The concept of the EDDY CER Factory arose out of the need to simplify the creation of EDDY Common Event Records (CERs) from event data by reducing the effort and complexity in doing so. Prior to the introduction of the CER Factory, developers were required to have a strong knowledge of XML, Java and object-oriented programming techniques.

With the advent of the EDDY CER Factory, developers are freed from the complexity of previous CER generation techniques. Instead of building an EDDY Normalization Agent derived from the EDDY Java Framework, a developer is now able to define a simple CER Factory Template containing a minimal set of CER fields using function calls as simple as a C 'printf()'.

Once created, the CER Factory Template is sent over TCP or UNIX Domain Socket to a waiting CER Factory service which receives the Template and generates a full and complete EDDY CER from it and routes it via the EDDY Backplane. The client generating the simple CER Factory Template and transmitting it to the waiting service can be written in any programming language, whether it's Perl, C, Java, etc.

The CER Factory approach is possible because of the nature that many of the fields that are a required part of a full EDDY CER can be populated with reasonable defaults. This fact allows the template approach to generating EDDY CERs by requiring only a minimal set of CER elements to be defined and accepting the defaults for the rest. The CER Factory design also allows any of the full EDDY CER elements to be included in the Template when needed and these optional elements are copied into the full EDDY CER the CER Factory Server produces.

A CER Factory Template is defined as an XML document that conforms to the schema definition described in Appendix D of this guide. In order for the CER Factory Template Server to produce a full CER from it, the Template must be well-formed XML in addition to including the set of required and any optional elements. It may include other elements not bound by the schema which will be turned into <userTag> elements if configured to handle these extraneous elements.

This release of the EDDY CER Factory Agent supports Template transport plug-ins for TCP sockets (unencrypted and SSL/TLS) and well as UNIX Domain Sockets. The design of the Template transport plug-in model allows other transport mechanisms to be added by implementing the routines of a CER Factory Java class interface. This topic is beyond the scope of this guide, but the capability is available to provide transport plug-ins for other transports such as HTTP, SOAP, UDP, etc.

The CER Factory Template Server supports multiple concurrent connections from clients. In this release of the CER Factory Agent, only one transport plug-in at a time is supported for each running instance of the CER Factory Template Server.

The CER Factory Template Server provides an access control mechanism for Template Clients who are transporting Templates to it. Access control is managed through the Template Server's eddyconfig.xml configuration file. In this file, IP addresses, subnets and full qualified domain names (FQDNs) may be specified to limit access to particular Template Clients. A further level of access granularity is provided by specifying which EDDY OIDs are accepted from the listed hosts.

Chapter

2

CER Factory Quick Start

Introduction

The EDDY CER Factory Quick Start Guide is intended to give instructions and pointers in order to download, install and verify the installation of the CER Factory Software Development Kit on a local development host.

The instructions contained in the ‘Downloading & Installing’ section will show you how to download and configure the CER Factory SDK in order to run locally within a single host. Steps are included to install the Java Development Kit that is included with the CER Factory SDK as well as creating and installing private X.509 certificates and keystores used for EDDY Agent to Agent communication.

The ‘Testing & Verification’ section will guide you through the steps needed to run and test the CER Factory Template Client & Server Agents. In addition to running the CER Factory Template Server, an EDDY Display Agent is also started that receives EDDY CERs generated by the CER Factory Template Server routed to it in response to Templates sent from the CER Factory Template Client.

The EDDY Display Agent will display the elements of CBPD (<http://www.arc.cmu.edu/cbpd>) Environmental CERs it reads and the instructions in this ‘Testing & Verification’ section will show how to send CBPD Templates from the CER Factory Client application. These steps will demonstrate an end-to-end example where Templates are sent from the CER Factory Client to the CER Factory Server which in turn generates a full EDDY CER which is then routed to a Display Agent which then displays the elements of the CERs it receives.

Downloading & Installing

The following instructions describe the steps required in order to download and install the EDDY CER Factory SDK. It is not a requirement to download and install the CER Factory SDK as the root user and it is better to be a regular user from a system security standpoint.

- **CER Factory SDK Installation**
Download the UNIX ‘tarball’ of the CER Factory SDK distribution from the following EDDY source:

<http://www.cmu.edu/eddy/docs/dist/eddy-0.5.2-4.tar.gz>

to a directory of your choosing. For the steps that follow, it will be assumed that the CER Factory SDK is downloaded to the /tmp directory.

- Bring up a command shell. For the remainder of the instructions, it will be assumed that you will be using the ‘bash’ shell since this is the default shell with newer Fedora and Red Hat Linux installations. The bash shell is not required, though the commands for defining the environmental variables needed by the EDDY CER Factory Agents will be bash specific. If you are using a shell other than bash, the ‘export’ commands that are described would typically be replaced with ‘setenv’ commands to accomplish the same thing.
- Unzip and un-tar the CER Factory SDK with the following commands in the command shell:

```
cd /tmp  
tar -xvzf eddy-agents-0.5.2-4.tar.gz
```

The EDDY CER Factory is now installed in the ./eddy directory.

- **Java Development Kit Installation**
Download the latest Java Development Kit from Sun here into the /tmp directory:
<http://java.sun.com/javase/downloads/index.jsp>

Install the JDK with the following commands in the currently opened command shell:

```
./jdk-6u3-linux-i586.bin
```

Note that the actual filename of the JDK may vary from the one above. As of the time of the writing of this Quick Start chapter, this was the latest JDK that was available.

The JDK will now start its installation procedure. Hit the space bar multiple times and accept the software license by typing ‘yes’ and then hit the return key. The Java JDK will now install and will return to a command prompt once it is finished. The JDK is now installed in the ./jdk1.6.0_03 (or similar) directory.

- **EDDY Certificates Installation**
The next installation step is to generate the x.509 certificates and keystores that are needed in order for EDDY Agents to communicate with each other. In development deployments of EDDY, the EDDY group would typically sign a host’s certificates with the EDDY Certificate Authority’s private key and give that to the administrator of the host for installation.

In testing and development environments, users of EDDY SDKs can create their own private EDDY Certificate Authority and sign certificates locally without having to submit signing requests to the EDDY group.

Creating and installing the required certificates and keystores can be performed with the following steps in the open command shell:

```
export JAVA_HOME=/tmp/jdk1.6.0_03
cd eddy/agents/0.5.2-1/eddycertificates/bin
./eddycacert
./eddyhostcert
cp ../lib/jssecacerts /tmp/jdk1.6.0_03/jre/lib/security
```

- The EDDY CER Factory SDK and Java Development Kit are now properly installed and configured so we can now test and verify the installation of the software. Exit the command shell now.

Testing & Verification

The instructions in this section are used to verify the proper installation and configuration of the EDDY CER Factory SDK and Java Development Kit. We will be testing the installation of the software by running two EDDY Agents, the CER Factory Template Server and the CBPD Environmental Event Display Agent.

We will then run occurrences of the CER Factory Template Client using both the Java and C versions of the Template Client to test and demonstrate the functionality of the CER Factory Server/CBPD Display Agent.

Instructions will also be given to illustrate error-reporting features of the CER Factory Server when it receives malformed Templates from the CER Factory Client.

Lastly, some of the configuration options of the CER Factory Template Client & Server and CBPD Display Agents will be explored to show how to control error reporting and alternate displays of Template and full EDDY CER data.

- **Agent Environmental Variables**

The first step in testing the EDDY CER Factory SDK is to bring up three bash command shells and issue the following shell commands in each of them:

```
cd /tmp/eddy/agents/0.5.2-4
export EDDY_CONFIG=../conf/eddyconfig.xml
export JAVA_HOME=/tmp/jdk1.6.0_03
export AGENT_MEM=512
```

The three environmental variables are used by EDDY Agents as part of their configuration on start up. The EDDY_CONFIG environment variable points to the file location of the configuration file for the Agent. The JAVA_HOME variable points to the root location of the Java Development Kit and the AGENT_MEM variable indicates the amount of memory (in megabytes) to allocate to the EDDY Agent.

- Now we want to change to the directories where the CER Factory software will be run from. Issue the following commands in each respective command shell:

```
shell #1: cd eddycerfactory/bin
shell #2: cd eddycerfactory/bin
shell #3: cd eddycbpdisplay/bin
```

- **CER Factory SDK Verification**

Start up the CBPD Environmental Display Agent in command shell #3 with the following command:

```
./eddycbpdisplay.dev
```

If the CBPD Display Agent is running correctly, you will not see any output in the command shell.

- Start up the CER Factory Template Server in command shell #2 with the following command:

```
./eddycerfactory.dev
```

If the CER Factory Template Server is running properly you will also not see any output on the command shell.

- Now run the Java version of the CER Factory Template Client utility by issuing the following command in shell #1:

```
./cerfclientjava.dev -d -f../misc/CBPD1CERF.xml -p1400 -slocalhost
```

Be sure to not have any spaces between the -f and -p selectors and their arguments. If the CER Factory Client & Server and the CBPD Display Agent are running properly, you'll see the following output in each of the three command shells (the times will be different):

shell #1:

```
<cerFactory>
  <eventInfo.oid>1005</eventInfo.oid>
  <!-- CBPD -->
  <dataPayload.payloadType>2</dataPayload.payloadType>
  <!-- Cooked -->
  <dataPayload.payload>
```

```
<cbpd>
  <cbpd-1.0.0>
    <heatSensor>
      <coverageArea>0.2</coverageArea>
      <heatSensorSetPoint>0.111</heatSensorSetPoint>
      <heatSensorLowerRange>0.003</heatSensorLowerRange>
      <heatSensorUpperRange>0.280</heatSensorUpperRange>
      <heatSensorAccuracy>0.001</heatSensorAccuracy>
      <timeConstant>0.1</timeConstant>
    </heatSensor>
  </cbpd-1.0.0>
</cbpd>
</dataPayload.payload>
</cerFactory>
```

The CER Factory Template Client that is executed in shell #1 displays the Template that is specified with the `-f` option specifier. A description of the options supported by the CER Factory Template Client can be seen by issuing the following command: `./cerfclientjava.dev -h`

shell #2:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<commonEventRecord>
  <header>
    <version>0.69</version>
    <ttl>128</ttl>
  </header>
  <eventInfo>
    <version>0.69</version>
    <oid>1005</oid>
    <eventID>0</eventID>
    <occurredStamp>
      <startTime>
        <utcTime>2007-08-30T18:44:40.659Z</utcTime>
      </startTime>
      <timeMethod>1</timeMethod>
      <agentTime>
        <utcTime>2007-08-30T18:44:40.661Z</utcTime>
      </agentTime>
      <agentTimeMethod>1</agentTimeMethod>
    </occurredStamp>
    <eventHostname>localhost.localdomain</eventHostname>
    <eventHostAddr>
      <ipv4Addr>127.0.0.1</ipv4Addr>
    </eventHostAddr>
    <normalizerHostname>localhost.localdomain</normalizerHostname>
    <normalizerAddr>
      <ipv4Addr>127.0.0.1</ipv4Addr>
    </normalizerAddr>
    <normalizerAddrType>
      <ipv4Addr>127.0.0.1</ipv4Addr>
    </normalizerAddrType>
    <corrDescriptor>
      <version>0.69</version>
      <eventClass>5</eventClass>
      <warningLevel>7</warningLevel>
      <eventCorrDescriptor>
        <environmentalEventCorrelationDescriptor>
          <version>0.69</version>
        </environmentalEventCorrelationDescriptor>
      </eventCorrDescriptor>
    </corrDescriptor>
  </eventInfo>
</dataPayload>
  <cookedDataPayload>
    <environmental>
      <cbpd>
```

```
<cbpd-1.0.0>
  <heatSensor>
    <coverageArea>0.2</coverageArea>
    <heatSensorSetPoint>0.111</heatSensorSetPoint>
    <heatSensorLowerRange>0.003</heatSensorLowerRange>
    <heatSensorUpperRange>0.280</heatSensorUpperRange>
    <heatSensorAccuracy>0.001</heatSensorAccuracy>
    <timeConstant>0.1</timeConstant>
  </heatSensor>
</cbpd-1.0.0>
</cbpd>
</environmental>
</cookedDataPayload>
</dataPayload>
</commonEventRecord>
```

The CER Factory Template Server that is executed in shell #2 displays the EDDY CER that is constructed from the Template it receives from the CER Factory Template Client.

shell #3:

```
header.version: 0.69
header.ttl: 127
eventInfo.version: 0.69
eventInfo.oid: 1005
eventInfo.eventID: 0
eventInfo.occurredStartTime: 2007-08-30T18:44:40.659Z
eventInfo.occurredTimeMethod: 1
eventInfo.occurredAgentTime: 2007-08-30T18:44:40.661Z
eventInfo.occurredAgentTimeMethod: 1
eventInfo.eventHostname: localhost.localdomain
eventInfo.eventHostAddr: 127.0.0.1
eventInfo.normalizerHostname: localhost.localdomain
eventInfo.normalizerAddr: 127.0.0.1
eventInfo.normalizerAddrType: 127.0.0.1
eventInfo.corrVersion: 0.69
eventInfo.corrEventClass: 5
eventInfo.corrWarningLevel: 7
eventInfo.corrEventEnvironmentalVersion: 0.69
```

The CBPD Display Agent that is executed in shell #3 displays some of the parsed out elements of the EDDY CER XML document it receives from the CER Factory Template Server. It does not display the CER payload and that part of the code is left as an exercise in the Display Agent. Some of the CBPD sensor element structures are parsed out and displayed and those serve as the basis for the remainder.

If you have received the same display output (except for the timestamps which would be different), your installation and configuration of the CER Factory SDK and Java Development Kit is correct and complete.

- **CER Factory Error-Reporting**

The next test will be to demonstrate the error-reporting capabilities of the CER Factory Template Server. The CER Factory Template Server makes a ‘best-effort’ to generate a full EDDY CER from the Templates that it receives. If it can’t produce an EDDY CER, it will display a log message to the command shell where the CER Factory Server was run from.

Note that the design of the CER Factory does not support communication of errors to the CER Factory Template Client sending the Template so any errors in producing an EDDY CER from the Template will go unnoticed by the CER Factory Client.

Edit the `../misc/CBPD1CERF.xml` file with a text editor. Change the `<eventInfo.oid>` element from 1005 to 1050. Now run the same CER Factory Template Client command you ran earlier in shell #1:

```
./cerfclientjava.dev -d -f../misc/CBPD1CERF.xml -p1400 -slocalhost
```

Now take a look at the display output from the CER Factory Server in shell #1. It no longer displays the EDDY CER produced, but rather the following error message indicating that it could not produce an EDDY CER from the Template sent to it:

```
[Thu, 30 Aug 2007 14:40:57 -0500, EDDY v0.5.2, EDDYCERFactory, WARN]: Malformed
Template; the <eventInfo.oid> element does not map to an Event Domain.
(localhost.localdomain:127.0.0.1)
```

This error indicates that the OID specified in the `<eventInfo.oid>` element wasn't one that was mapped to an Event Domain (Network, Security, System, Application or Environmental) in the CER Factory Server's `eddyconfig.xml` configuration file. The error message also indicates the FQDN and IP address of the Template Client that sent the malformed Template.

- Now let's illustrate a second example where the Template sent by the CER Factory Client is malformed. Edit the `../misc/CBPD1CERF.xml` once again and change the `<dataPayload.payloadType>` from 2 to 5. Be sure to also change the `<eventInfo.oid>` value back to 1005. Run the CER Factory Template Client as before with:

```
./cerfclientjava.dev -d -f../misc/CBPD1CERF.xml -p1400 -slocalhost
```

The display output from the CER Factory Server display an error message, but one that is different from the first one we saw:

```
[Thu, 30 Aug 2007 14:58:34 -0500, EDDY v0.5.2, EDDYCERFactory, WARN]: Malformed
Template; it could not be marshalled into Java objects.
(localhost.localdomain:127.0.0.1)
```

This error indicates that the CER Factory Template is malformed according to the schema definition for Templates. The reason for the error message is that the `<dataPayload.payloadType>` element is typed in the Template's schema to only hold values for 1, 2 or 3 that represent Raw, Cooked or Analyzed data payloads. A value of 5 is subsequently invalid.

- **C Template Client**

We have been testing the CER Factory SDK with the Java version of the CER Factory Template Client and now we'll run a test with the C version. The C Client has the exact same functionality as the Java Client with a similar set of options as well.

Before executing the C Template Client, edit the `../misc/CBPD1CERF.xml` file and change the `<dataPayload.payloadType>` element back to 2 from 5. Run the C Template Client with the following command:

```
./cerfclientc -d -f../misc/CBPD1CERF.xml -p1400 -slocalhost
```

If all went well, you will see the same output in each of the command shells that you saw at the beginning of this section. If you get an error and the C program doesn't run, you may need to recompile the C Template Client to work with your particular environment. To recompile the C Template Client, issue the following command in the command shell and try running the Client again:

```
./cerfclientc.dev
```

- **CER Factory Client Configuration**

The CER Factory Template Client programs in both Java and C forms support the following command line arguments as seen in the 'usage' text displayed when the command `./cerfclientjava.dev -h` is run:

```
usage: cerfclientjava [-d] -f<template> [-h] [-r<delay>]
      ([-k<keystore> -l<password>] -p<port> -s<host> | -u<socket>)
options: -d          display CER Factory Template file
         -f<template> CER Factory Template file
         -h          display this usage list and exit
         -k<keystore> SSL/TLS keystore
         -l<password> SSL/TLS keystore password
         -p<port>    CER Factory Template Server port
         -r<delay>  send Template file every <delay> milliseconds
         -s<host>   CER Factory Template Server host/address
         -u<socket> CER Factory Template Server domain socket
version: 1.0.7 (4.December.2007)
author:  Jim Gargani (jgargani@cmu.edu)
```

By default, the CER Factory Template Server listens on TCP port 1400 and that is why we use this port in the examples when running the Template Client. The options listed above are pretty straightforward as to their meaning.

The CER Factory Template Client is able to send Templates to CER Factory Servers listening on both TCP ports (unencrypted and SSL/TLS) and UNIX Domain Sockets. These methods are the three transport types supported by the CER Factory SDK. When UNIX Domain Sockets are specified, the CER Factory Client must be running on the same host as the CER Factory Server. Note that either a TCP socket or a UNIX domain socket may be specified, but not both at the same time.

Also note that the CER Factory Client is not always limited to sending Templates to the host that it is running on. The '-s' option allows the Template Client to connect to any CER Factory Template Server by specifying its fully qualified domain name or IP address.

- **CER Factory Server Configuration**

The CER Factory Template Server supports the set of standard configuration options that any EDDY Agent does. Of potential interest to the user reading this Quick Start guide are those options that are specific to the CER Factory Template Server Agent itself. The Agent specific preference options supported are highlighted in the eddyconfig.xml included with CER Factory Server in the CER Factory SDK seen below:

```
<?xml version="1.0" encoding="UTF-8"?>
<eddyConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <host name="localhost">
    <agent name="edu.cmu.acs.eddy.EDDYCERFactory">
      <base>
        <snip...>
      </base>
      <extended>
        <agent>
          <boolOption name="debug" value="false"/>
        </agent>
        <acquire>
          <boolOption name="display" value="true"/>
          <boolOption name="extraneous" value="true"/>
          <boolOption name="status" value="true"/>
          <boolOption name="strictV069" value="false"/>
          <intOption name="queueSize" value="1000"/>
          <intOption name="timeout" value="60"/>

          <snip...>

        <strOption name="oidMap">
          <value>
            <!-- OIDs are mapped, one per line to the      -->
            <!-- corresponding Event Domain with the      -->
            <!-- following                                    -->
            <!-- syntax: OID:Event Domain (Event Class).  -->
            <!--                                           -->
            <!-- The Event Domain values range from 1-5 and -->
            <!-- reflect the following Event Domain types: -->
            <!-- Network = 1, Security = 2, System = 3,    -->
            <!-- Application = 4, Environmental = 5.       -->
            <!--                                           -->
            <!-- The following mapping list defines the    -->
            <!-- currently defined EDDY OIDs as of the    -->
            <!-- development of this agent.               -->
            1001:1
            1002:1
            1003:1
            1004:1
            1005:5
            1006:1
            1007:1
            1008:4
            1009:1
            1020:4
            1021:4
            1022:4
          </value>
        </strOption>
      </extended>
    </agent>
  </host>
</eddyConfig>
```

```
</strOption>

  <snip...>
</acquire>
<consume>
  <intOption name="total" value="10"/>
</consume>
</extended>
</agent>
</host>
</eddyConfig>
```

The ‘display’ configuration preference when set to ‘true’ will display the EDDY CER that is produced from the CER Factory Template if the CER Factory Template Server is able to generate it. If this value is set to false or if it is not included in the eddyconfig.xml file, it defaults to a value of ‘false’ inside the Agent.

The ‘extraneous’ configuration preference when set to ‘true’ will allow the processing of XML elements specified in the CER Factory Template that are not bound to the Template schema definition. For those elements that are included in the Template but are not bound to the schema, they are defined in the full EDDY CER as <userTag> elements. If the ‘extraneous’ option is not defined or set to false, it defaults to a value of ‘false’ inside the Agent. This configuration preference is provided because of the computational and parsing effort required to support this feature and since it is expected to be an infrequently used capability, it is disabled by default.

The ‘status’ configuration preference when set to ‘true’ will display error messages resulting from malformed Templates sent by the CER Factory Template Client. The error message text indicates the nature of the malformed Template as well as including the FQDN and IP address of the Template Client sending the Template. If this value is set to false or if it is not included in the eddyconfig.xml file, it defaults to a value of ‘false’ inside the Agent.

The ‘strictv069’ configuration preference when set to ‘true’ will force the CER Factory Template Server to produce EDDY CERs that conform strictly to the version 0.6.9 standard. The difference between strict and non-strict version 0.6.9 EDDY CERS is the inclusion of XML attribute fields that contain no CER-specific (useless) content in the strict rendering. An incompatibility between JAXB 1.6 and 2.0 required removing these attributes and all EDDY Agents moving forward will be coded to work with JAXB 2.0 and the non-strict version of the EDDY CER. This configuration preference is included to provide compatibility with older JAXB 1.6 EDDY Agents that may be receiving CERs produced by the CER Factory Template Server. If this value is set to false or if it is not included in the eddyconfig.xml file, it defaults to a value of ‘false’ inside the Agent.

The ‘queueSize’ and ‘timeout’ configuration preferences are used to specify parameters specific to the CER Factory Template transport mechanism and are described in the ‘Client & Server Protocol’ section of Chapter 4 below.

The 'oidMap' configuration preference is used to define a mapping of CER OIDs to the Event Domain () that represents that OID. This mapping allows a Template Client to not include Event Domain information in the Template it sends to the CER Factory Server although the Event Domain may be defined in the Template if desired. The OID mapping is a series of lines of OIDs, followed with a colon and then the Event Domain that the OID is associated with. Care must be taken when defining the OID mapping list because the CER Factory Template Server provides little error checking to verify that the syntax and semantics of OID mapping lines are correct. Although the OID mapping list above is in numeric order, there is no strict requirement for it to be defined that way. If the 'oidMap' preference option is not specified or is empty, no default OID mappings are used and no OID map will exist in the CER Factory Template Server. This will require all Templates to have Event Domain elements defined for every Template sent by the CER Factory Template Client.

The 'total' configuration preference value is used to print a notification of total counts of CERs processed by the CER Factory Template Server Agent every time 'total' number of additional CERs have been read. If this value is set to 0 or if the preference option is not included, this status feature is disabled.

- **CBPD Display Agent Configuration**

The CBPD Environmental Event Display Agent in similar fashion to the CER Factory Template Server supports the set of standard configuration options that any EDDY Agent does. The only Agent specific preference options supported (other than one for debugging) are those that control what the Agent displays. They are highlighted in the following eddyconfig.xml file that is included in the EDDY CER Factory distribution below:

```
<?xml version="1.0" encoding="UTF-8"?>
<eddyConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <host name="localhost">
    <agent name="edu.cmu.acs.eddy.EDDYCBPDDisplay">
      <base>
        <snip...>
      </base>
      <extended>
        <agent>
          <boolOption name="debug" value="false"/>
        </agent>
        <consume>
          <boolOption name="dump" value="false"/>
          <boolOption name="parse" value="true"/>
          <intOption name="total" value="10"/>
        </consume>
      </extended>
    </agent>
  </host>
</eddyConfig>
```

The 'dump' configuration preference when set to 'true' will display the EDDY CBPD CER that the Agent reads off of the EDDY Backplane. This is the CER

that is sent from the CER Factory Template Server in our testing examples. If this option is not defined, it defaults to a 'false' value inside the Agent.

The 'parse' configuration preference when set to 'true' parses out the XML elements of the CBPD CER and displays them one to a line. The CBPD Display Agent utilizes JAXB 2.0 for the unmarshalling of XML document into Java objects. If this option is not defined, it defaults to a 'false' value inside the Agent.

The 'total' configuration preference value is used to print a notification of total counts of CERs processed by the CBPD Display Agent every time 'total' number of additional CERs have been read. If this value is set to 0 or if the preference option is not included, this status feature is disabled.

An important configuration aspect to consider is that changes to the eddyconfig.xml file currently requires a restart of the Agent in order for the new changes to take effect. This applies to both the CBPD Display Agent and the CER Factory Server Agent.

Chapter

3

CER Factory Design

Introduction

The EDDY CER Factory design is centered on the idea of making it as easy and straightforward as possible for developers to produce EDDY CERs from event data. It also provides a more efficient and less CPU intensive approach to generating them.

These goals are achieved by reducing the set of elements and structures required in a complete EDDY CER to a bare minimum of just a few. This reduced set of EDDY CER elements is placed inside a CER Factory Template that is sent to a CER Factory Template Server over a transport mechanism, currently either a TCP socket (encrypted or not) or a UNIX domain socket.

The CER Factory Server then builds a full EDDY CER using the data the Template includes while providing defaults for any of the required CER fields the Template does not define. The CER Factory Template Server always attempts a best effort to produce a full EDDY CER from the Template regardless of the data it receives. As a result, the CER Factory is quite robust in managing any possible Template it may be presented with.

Required Template Elements

The CER Factory Template schema definition defines a flattened form of the full EDDY CER schema. The CER Factory Template schema provides a one to one mapping of elements from the full EDDY CER into a less heavily nested structure. CER Factory Template Clients are free to pick and choose which elements of the CER Factory Template schema that want to define and upon receipt of the CER Factory Template from the Client, the CER Factory Template Server will populate the full EDDY CER it builds with these specified elements and make a best guess effort to define those elements that are not specified.

Although the vast majority of elements in the CER Factory Template schema are optional, there are several elements that must be included in any CER Factory Template that is passed from the Template Client to the Template Server. The following elements that may be referenced in Appendix A of this document are required in any CER Factory Template that is constructed:

- *<eventInfo.oid>* - This element defines the OID or unique identifier value for the EDDY CER to be constructed. The CER Factory Template Server cannot provide a default for this element since it is arbitrary. The current list of OIDs defined may

be referenced at <http://www.cmu.edu/computing/eddy/software-dist.htm#OID>. This element is defined as an xs:unsignedLong data type and is required to hold a value between 0 and 18,446,744,073,708,551,615 in order to be considered valid.

- *<dataPayload.payloadType>* - This element defines the Event Order or type of the payload data specified by the *<dataPayload.payload>* element. Event Orders that have defined include Raw, Cooked and Analysis types. This is another arbitrary CER element that does not have a sensible default, so it is required to be defined in CER Factory Templates. This element is defined as a payloadType data type and is required to hold a value between 1-3 (Raw = 1, Cooked = 2, Analysis = 3) in order to be considered valid.
- *<dataPayload.payload>* - This element is the core of any CER Factory Template or EDDY CER; it contains the data of interest for the EDDY Event. Again it is an arbitrary element and must be defined in the CER Factory Template since no reasonable default could be selected for it. This element is defined as an xs:string data type and should hold a well-formed XML payload representation that corresponds to the Event Order of the specified CER OID.

If one or more of the four required CER Factory Template elements are not defined in the XML document sent by the CER Factory Template Client to the Server, the request to build a full EDDY CER will fail silently without the Client being informed of the error condition. The lack of acknowledgement status indicators between CER Factory Template Client and Server components is a design decision meant to increase simplicity in the CER Factory software as well as reduce the complexity of the Template Client that will generate the Template.

Optional Template Elements

At a minimum, the CER Factory Template Server is able to produce a full EDDY CER using only the required Template fields and substituting its own defaults for the rest of the fields needed to craft a CER. The defaults it chooses are reasonable values that convey useful information or default “no-op” data.

Each EDDY CER is comprised of elements in a nested, hierarchical format. The CER Factory Template is a flattened version of the EDDY CER with each element of the full CER represented in it. This means that all of the data contained in a full EDDY CER may be included in a CER Factory Template by defining the full set of required and optional Template elements.

When the CER Factory Server receives a Template sent to it by the Factory Client, it builds a full EDDY CER by populating the CER it is crafting with the required and optional elements specified in the Template. For all other elements the Template Server needs to define in order to produce a complete CER, it chooses the default values and elements described below.

Typically, only the required elements are specified in the CER Factory Template with perhaps a handful of optional elements such as timestamp and warning level data. Some of the more popular and frequently defined optional elements include:

- `<eventInfo.eventClass>` - Event Domain of the event.
- `<eventInfo.startTime.utcTime>` - Time representing the start of the event.
- `<eventInfo.stopTime.utcTime>` - Time representing the end of the event.
- `<eventInfo.eventHostname>` - FQDN of the device/host generating the event.
- `<eventInfo.eventHostAddr>` - IP of the device/host generating the event.
- `<eventInfo.warningLevel>` - Warning level of the event.

Indirect Template Elements

CER Factory Template elements map one-to-one to elements in the full EDDY Common Event Record. Comparing the CER Factory Template schema to that of the full EDDY CER schema it is reasonably clear which elements in one schema map to the other.

For example, the CER Template Factory Template element `<eventInfo.StartTime.utcTime>2007-06-28T15:15:15.000+00:00</eventInfo.startTime.utcTime>` can easily be observed to map to the full EDDY CER element structure:

```
<startTime>
  <utcTime>2007-06-28T15:15:15.000+00:00</utcTime>
</startTime>
```

While most Template elements map in the obvious manner seen above, there are a few that provide a more indirect and more complex mapping from the CER Factory Template to the full EDDY CER. The reason for this indirect mapping is to keep the definition of the CER Factory Template as simple, lightweight and concise as possible.

The following CER Factory Template elements shown are the indirect elements. For each indirect element below, an explanation and snippet is provided that demonstrates how that indirect element is expanded into XML in a full EDDY CER.

- `<eventInfo.eventHostAddrType>`
`<eventInfo.normalizerAddrType>`
`<eventInfo.corrDescriptor.networkEventCorrDescriptor.srcAddrType>`
`<eventInfo.corrDescriptor.networkEventCorrDescriptor.dstAddrType>`
`<dataPayload.analyzedDataPayload.analysisHostAddrType>`
`<dataPayload.analyzedDataPayload.corrDescriptor.networkEventCorrDescriptor.srcAddrType>`
`<dataPayload.analyzedDataPayload.corrDescriptor.networkEventCorrDescriptor.dstAddrType>`

For each of these optional Template elements, a network type identifier is specified. The valid values range from 1-3. A value of 1 indicates that an Ethernet

MAC address is defined, 2 indicates a IP version 4 address and 3 a IP version 6 address. Each of these indirect elements is used in tandem with a corresponding network address element. If a network type identifier is defined without a corresponding network address, the type identifier is ignored. If the network type identifier is a value other than 1-3, the Template is discarded and an EDDY CER is not generated. The following element definition examples illustrate the full EDDY CER XML that is generated for the three network address types:

Template:

```
<eventInfo.eventHostAddrType>1</eventInfo.eventHostAddrType>  
<eventInfo.eventHostAddr>AA:12:CD:34:56:2C</eventInfo.eventHostAddr>
```

EDDY CER:

```
<eventHostAddr>  
  <ethernetAddr>  
    AA:12:CD:34:56:2C  
  </ethernetAddr>  
</eventHostAddr>
```

Template:

```
<eventInfo.eventHostAddrType>2</eventInfo.eventHostAddrType>  
<eventInfo.eventHostAddr>128.2.11.105</eventInfo.eventHostAddr>
```

EDDY CER:

```
<eventHostAddr>  
  <ipv4Addr>  
    128.2.11.105  
  </ipv4Addr>  
</eventHostAddr>
```

Template:

```
<eventInfo.eventHostAddrType>3</eventInfo.eventHostAddrType>  
<eventInfo.eventHostAddr>206.82.54.32</eventInfo.eventHostAddr>
```

EDDY CER:

```
<eventHostAddr>  
  <ipv6Addr>  
    206.82.54.32  
  </ipv6Addr>  
</eventHostAddr>
```

- <dataPayload.payloadType>

This required element is used to define the payload type or Event Order for the full EDDY CER that is being produced from the CER Factory Template. The valid values range from 1 to 3 and represent raw, cooked and analyzed payloads respectfully. The following examples illustrate the full EDDY CER XML snippet that is generated for each of these three Event Order values:

Template:

```
<dataPayload.payloadType>1</dataPayload.payloadType>  
<dataPayload.payload>  
  ASAA7ACACAB/AAABAAAABcAFIAAAAAAF5RDRO6wAGTUVENE7rAAZNRcCoAQHv
```

```
//6EQARNQdsitYAAAAABAAAAAEEAAAFqAAABQAAAAAAAAAAAAAAAAAAgQ  
CAAADYgr27kBAF5//pCIQAATk9USUZZICogSFRUUC8xLjENCkhPU1Q6IDlz  
OS4yNTUuMjU1LjI1MDoxOTAwDQpDQUNIRS1DT05UUK9MOiBtYXgtYWdlPTEy  
MA0KTE9DQVRJT046IGh0dHA6Ly8xOTIuMTY4LjEuMT01Njc4L2lnZC54bWwN  
Ck5UOiB1cm46c2M=  
</dataPayload.payload>
```

EDDY CER:

```
<dataPayload>  
  <rawDataPayload>  
    <rawEvent>  
      ASAA7ACACAB/AAABAAABcAFIAAAAAAF5RDRO6wAGTUVENE7rAAZNRcCoAQHv  
      //6EQARNQdsitYAAAAABAAAAAEEAAAFqAAABQAAAAAAAAAAAAAAAAAAgQ  
      CAAADYgr27kBAF5//pCIQAATk9USUZZICogSFRUUC8xLjENCkhPU1Q6IDlz  
      OS4yNTUuMjU1LjI1MDoxOTAwDQpDQUNIRS1DT05UUK9MOiBtYXgtYWdlPTEy  
      MA0KTE9DQVRJT046IGh0dHA6Ly8xOTIuMTY4LjEuMT01Njc4L2lnZC54bWwN  
      Ck5UOiB1cm46c2M=  
    </rawEvent>  
  </rawDataPayload>  
</dataPayload>
```

Template:

```
<eventInfo.eventClass>5</eventInfo.eventClass>  
<dataPayload.payloadType>2</dataPayload.payloadType>  
<dataPayload.payload>  
  <cbpd>  
    <cbpd-1.0.0>  
      <temperatureSensor>  
        <temperatureSensorType>5</temperatureSensorType>  
        <temperatureSensorSetPoint>50.0</temperatureSensorSetPoint>  
        <temperatureSensorLowerRange>5.0</temperatureSensorLowerRange>  
        <temperatureSensorUpperRange>100.0</temperatureSensorUpperRange>  
        <accuracyOfTemperatureSensor>0.001</accuracyOfTemperatureSensor>  
        <timeConstant>00.1</timeConstant>  
      </temperatureSensor>  
    </cbpd-1.0.0>  
  </cbpd>  
</dataPayload.payload>
```

EDDY CER:

```
<dataPayload>  
  <cookedDataPayload>  
    <environmental>  
      <cbpd>  
        <cbpd-1.0.0>  
          <heatSensor>  
            <coverageArea>0.2</coverageArea>  
            <heatSensorSetPoint>0.111</heatSensorSetPoint>  
            <heatSensorLowerRange>0.003</heatSensorLowerRange>  
            <heatSensorUpperRange>0.280</heatSensorUpperRange>  
            <heatSensorAccuracy>0.001</heatSensorAccuracy>  
            <timeConstant>0.1</timeConstant>  
          </heatSensor>  
        </cbpd-1.0.0>  
      </cbpd>  
    </environmental>  
  </cookedDataPayload>  
</dataPayload>
```

Template:

```
<eventInfo.eventClass>1</eventInfo.eventClass>
<dataPayload.payloadType>3</dataPayload.payloadType>
<dataPayload.payload>
  <diagnosis>Congestion in Core.</diagnosis>
  <analysisData>
    <flowsIn>1234</flowsIn>
    <flowsOut>5678</flowsOut>
    <bytesIn>12345678</bytesIn>
    <bytesOut>87654321</bytesOut>
    <packetsIn>12345</packetsIn>
    <packetsOut>54321</packetsOut>
  </analysisData>
</dataPayload.payload>
```

EDDY CER:

```
<dataPayload>
  <analyzedDataPayload>
    <version>0.69</version>
    <analysisHostname>localhost.localdomain</analysisHostname>
    <analysisHostAddr>127.0.0.1</analysisHostAddr>
    <occurredStamp>
      <startTime>
        <utcTime>2007-07-16T15:15:15.000+00:00</utcTime>
      </startTime>
      <timeMethod>1</timeMethod>
      <agentTime>
        <utcTime>2007-07-16T15:15:15.000+00:00</utcTime>
      </agentTime>
      <agentTimeMethod>1</agentTimeMethod>
    </occurredStamp>
    <corrDescriptor>
      <version>0.69</version>
      <eventClass>1</eventClass>
      <warningLevel>7</warningLevel>
      <eventCorrDescriptor>
        <networkEventCorrDescriptor>
          <version>0.69</version>
          <netProto>0</netProto>
          <srcAddr>
            <ipv4Addr>0.0.0.0</ipv4Addr>
          </srcAddr>
          <dstAddr>
            <ipv4Addr>0.0.0.0</ipv4Addr>
          </dstAddr>
          <srcPort>0</srcPort>
          <dstPort>0</dstPort>
          <direction>bi</direction>
        </networkEventCorrDescriptor>
      </eventCorrDescriptor>
    </corrDescriptor>
    <diagnosis>Congestion in Core.</diagnosis>
    <analysisData>
      <flowsIn>1234</flowsIn>
```

```
<flowsOut>5678</flowsOut>
<bytesIn>12345678</bytesIn>
<bytesOut>87654321</bytesOut>
<packetsIn>12345</packetsIn>
<packetsOut>54321</packetsOut>
</analysisData>
</analyzedDataPayload>
</dataPayload>
```

Default Template Elements

Any optional CER Factory Template elements that are not defined in the Template will be populated with default values in the full EDDY CER created by the CER Factory Template Server. The Template Server makes no attempt to sanity-check in a semantic sense its default values against the optional Template elements that may also be defined. As a result, the burden of defining valid and sensible element values is the responsibility of the CER Factory Template Client creating the CER Factory Template.

The CER Factory Template Server populates verbatim the required and optional elements into the full EDDY CER it is building. It maintains a list of elements that have been defined in the CER Factory Template and applies defaults for any unspecified elements that are required to craft a full CER. The Template Server constructs the smallest possible CER based on the Template elements it receives and does not provide defaults for optional elements of the full CER if they are not needed.

An example of this can be seen when the optional `<eventInfo.occurredStamp.stopTime.subseconds>` element is defined in the CER Factory Template. Without this element, there is no need to build out the `<stopTime>` structure in the `<eventInfo>` structure since `<stopTime>` itself is optional and the Template Server is not given any elements that comprise it. But when the `<eventInfo.occurredStamp.stopTime.subseconds>` element is defined, this necessitates building out the required elements of the `<stopTime>` structure including selecting a default value (current time) for the `<utcTime>` element as well as populating the `<subseconds>` element with the supplied value.

The following XML snippets illustrate optional Template elements and the resulting full CER XML that is generated in response:

This Template element...

```
<eventInfo.occurredStamp.stopTime.subseconds>
  1234.56
</eventInfo.occurredStamp.stopTime.subseconds>
```

...Produces this full CER element:

```
<stopTime>
  <utcTime>
    2007-06-28T19:15:15.000+00:00
  </utcTime>
  <subseconds>
```

```
1234.56
</subseconds>
</stopTime>
```

This Template element...

```
<eventInfo.occurredStamp.stopTime.utcTime>
2007-06-28T19:15:15.000+00:00
</eventInfo.occurredStamp.stopTime.utcTime>
```

...Produces this full CER element:

```
<stopTime>
  <utcTime>
    2007-06-28T19:15:15.000+00:00
  </utcTime>
</stopTime>
```

The following list of CER Factory Template elements and values illustrate the *default* values that are placed in the full CER generated by the Template Server for those elements not specifically defined by the Template Client in its CER Factory Template:

- <header.version> - 0.69
- <header.ttl> - 128
- <eventInfo.version> - 0.69
- <eventInfo.eventID> - 0
- <eventInfo.startTime.utcTime> - Current time.
- <eventInfo.stopTime.utcTime> - Current time, only included as default if <eventInfo.stopTime.subseconds> is defined as an optional element.
- <eventInfo.timeMethod> - 1
- <eventInfo.agentTime.utcTime> - Current time.
- <eventInfo.agentTimeMethod> - 1
- <eventInfo.eventHostname> - FQDN of CER Factory Template Client.
- <eventInfo.eventHostAddrType> - 2
- <eventInfo.eventHostAddr> - IP address of CER Factory Template Client.
- <eventInfo.normalizerHostname> - FQDN of CER Factory Template Server.
- <eventInfo.normalizerAddrType> - 2
- <eventInfo.normalizerAddr> - IP address of CER Factory Template Server.
- <eventInfo.corrDescriptor.version> - 0.69
- <eventInfo.warningLevel> - 7

- <eventInfo.corrDescriptor.eventCorrDescriptor.version> - 0.69
- <eventInfo.corrDescriptor.networkEventCorrDescriptor.netProto"> - 0, only included as default if <eventInfo.eventClass> is defined as a Network Event Domain or if <eventInfo.oid> maps to a Network Event Domain.
- <eventInfo.corrDescriptor.networkEventCorrDescriptor.srcAddrType> - 2, only included as default if <eventInfo.eventClass> is defined as a Network Event Domain or if <eventInfo.oid> maps to a Network Event Domain.
- <eventInfo.corrDescriptor.networkEventCorrDescriptor.srcAddr> - 0.0.0.0, only included as default if <eventInfo.eventClass> is defined as a Network Event Domain or if <eventInfo.oid> maps to a Network Event Domain.
- <eventInfo.corrDescriptor.networkEventCorrDescriptor.dstAddrType> - 2, only included as default if <eventInfo.eventClass> is defined as a Network Event Domain or if <eventInfo.oid> maps to a Network Event Domain.
- <eventInfo.corrDescriptor.networkEventCorrDescriptor.dstAddr> - 0.0.0.0, only included as default if <eventInfo.eventClass> is defined as a Network Event Domain or if <eventInfo.oid> maps to a Network Event Domain.
- <eventInfo.corrDescriptor.networkEventCorrDescriptor.srcPort> - 0, only included as default if <eventInfo.eventClass> is defined as a Network Event Domain or if <eventInfo.oid> maps to a Network Event Domain.
- <eventInfo.corrDescriptor.networkEventCorrDescriptor.dstPort> - 0, only included as default if <eventInfo.eventClass> is defined as a Network Event Domain or if <eventInfo.oid> maps to a Network Event Domain.
- <eventInfo.corrDescriptor.networkEventCorrDescriptor.direction> - bi, only included as default if <eventInfo.eventClass> is defined as a Network Event Domain or if <eventInfo.oid> maps to a Network Event Domain.
- <dataPayload.analyzedDataPayload.version> - 0.69, only included as default if <dataPayload.payloadType> is defined as an Analysis Event Order.
- <dataPayload.analyzedDataPayload.analysisHostname> - FQDN of CER Factory Template Server, only included as default if <dataPayload.payloadType> is defined as an Analysis Event Order.
- <dataPayload.analyzedDataPayload.analysisHostAddrType> - 2, only included as default if <dataPayload.payloadType> is defined as an Analysis Event Order.
- <dataPayload.analyzedDataPayload.analysisHostAddr> - IP address of CER Factory Template Server, only included as default if <dataPayload.payloadType> is defined as an Analysis Event Order.
- <dataPayload.analyzedDataPayload.startTime.utcTime> - Current time, only included as default if <dataPayload.payloadType> is defined as an Analysis Event Order.
- <dataPayload.analyzedDataPayload.stopTime.utcTime> - Current time, only included as default if <dataPayload.payloadType> is defined as an Analysis Event Order and if <dataPayload.analyzedDataPayload.stopTime.subseconds> is defined as an optional element.

- `<dataPayload.analyzedDataPayload.timeMethod>` - 1, only included as default if `<dataPayload.payloadType>` is defined as an Analysis Event Order.
- `<dataPayload.analyzedDataPayload.agentTime.utcTime>` - Current time, only included as default if `<dataPayload.payloadType>` is defined as an Analysis Event Order.
- `<dataPayload.analyzedDataPayload.agentTimeMethod>` - 1, only included as default if `<dataPayload.payloadType>` is defined as an Analysis Event Order.
- `<dataPayload.analyzedDataPayload.corrDescriptor.version>` - 0.69, only included as default if `<dataPayload.payloadType>` is defined as an Analysis Event Order.
- `<dataPayload.analyzedDataPayload.corrDescriptor.eventClass>` - `<eventInfo.eventClass>`, only included as default if `<dataPayload.payloadType>` is defined as an Analysis Event Order.
- `<dataPayload.analyzedDataPayload.corrDescriptor.warningLevel>` - 7, only included as default if `<dataPayload.payloadType>` is defined as an Analysis Event Order.
- `<dataPayload.analyzedDataPayload.corrDescriptor.eventCorrDescriptor.version>` - 0.69, only included as default if `<dataPayload.payloadType>` is defined as an Analysis Event Order.
- `<dataPayload.analyzedDataPayload.corrDescriptor.networkEventCorrDescriptor.netProto>` - 0, only included as default if `<dataPayload.payloadType>` is defined as an Analysis Event Order and if `<eventInfo.eventClass>` is defined as a Network Event Domain or if `<eventInfo.oid>` maps to a Network Event Domain.
- `<dataPayload.analyzedDataPayload.corrDescriptor.networkEventCorrDescriptor.srcAddrType>` - 2, only included as default if `<dataPayload.payloadType>` is defined as an Analysis Event Order and if `<eventInfo.eventClass>` is defined as a Network Event Domain or if `<eventInfo.oid>` maps to a Network Event Domain.
- `<dataPayload.analyzedDataPayload.corrDescriptor.networkEventCorrDescriptor.srcAddr>` - 0.0.0.0, only included as default if `<dataPayload.payloadType>` is defined as an Analysis Event Order and if `<eventInfo.eventClass>` is defined as a Network Event Domain or if `<eventInfo.oid>` maps to a Network Event Domain.
- `<dataPayload.analyzedDataPayload.corrDescriptor.networkEventCorrDescriptor.dstAddrType>` - 2, only included as default if `<dataPayload.payloadType>` is defined as an Analysis Event Order and if `<eventInfo.eventClass>` is defined as a Network Event Domain or if `<eventInfo.oid>` maps to a Network Event Domain.
- `<dataPayload.analyzedDataPayload.corrDescriptor.networkEventCorrDescriptor.dstAddr>` - 0.0.0.0, only included as default if `<dataPayload.payloadType>` is defined as an Analysis Event Order and if `<eventInfo.eventClass>` is defined as a Network Event Domain or if `<eventInfo.oid>` maps to a Network Event Domain.
- `<dataPayload.analyzedDataPayload.corrDescriptor.networkEventCorrDescriptor.srcPort>` - 0, only included as default if `<dataPayload.payloadType>` is defined as an Analysis Event Order and if `<eventInfo.eventClass>` is defined as a Network Event Domain or if `<eventInfo.oid>` maps to a Network Event Domain.
- `<dataPayload.analyzedDataPayload.corrDescriptor.networkEventCorrDescriptor.dstPort>` - 0, only included as default if `<dataPayload.payloadType>` is defined as an Analysis Event Order and if

`<eventInfo.eventClass>` is defined as a Network Event Domain or if `<eventInfo.oid>` maps to a Network Event Domain.

- `<dataPayload.analyzedDataPayload.corrDescriptor.networkEventCorrDescriptor.direction>` - bi, only included as default if `<dataPayload.payloadType>` is defined as an Analysis Event Order and if `<eventInfo.eventClass>` is defined as a Network Event Domain or if `<eventInfo.oid>` maps to a Network Event Domain.

Event Domain Mapping

Each EDDY CER belongs to one of five Event Domains (System, Application, Network, Security or Environmental) and this Event Domain may be specified in the optional Template element `<eventInfo.eventClass>`. The Event Domain is a critical element that needs to be determined in order to construct a full EDDY CER. A question arising is ‘How can the Event Domain element be optional if it is a component of a CER that can’t be inferred as a default value?’ The answer is that CER Factory Template Server provides a service where the Event Domain is made available by a table lookup using the value in the `<eventInfo.oid>` element. This table has knowledge of which Event Domains map to which OIDs.

The table that maps EDDY OIDs to their respective Event Domains is defined in the CER Factory Template Server’s configuration file, `eddyconfig.xml`. The mapping table is defined in the Agent preference element `<oidMap>`. An example OID mapping table is included below for all of the currently defined EDDY CERs. Instances of the CER Factory Template Server may extend this list as needed.

```
<extended>
  <acquire>
    <strOption name="oidMap">
      <value>
        <!-- OIDs are mapped, one per line to the corresponding Event Domain with the following
              syntax: OID:Event Domain (Event Class).
        -->
```

The Event Domain values range from 1-5 and reflect the following Event Domain types:
Network = 1, Security = 2, System = 3, Application = 4, Environmental = 5.

The following mapping list defines the currently defined EDDY OIDs as of the publication of this document. -->

```
1001:1
1002:1
1003:1
1004:1
1005:5
1006:1
1007:1
1008:4
1009:1
1020:4
1021:4
1022:4
      </value>
    </strOption>
```

```
</acquire>  
</extended>
```

The CER Factory Template Server will use the OID mapping table for all Templates that it receives that do not contain a `<eventInfo.eventClass>` element. It first extracts the OID from the Template and then consults the mapping table for a match. If a match is found, the EDDY CER under construction is generated for the matching Event Domain type. If no match is found, the EDDY CER is not generated and the Template is ignored since there are no defaults that can be assumed.

The Event Domain OID mapping method is not used when the `<eventInfo.eventClass>` is defined by the CER Factory Template. In the case when the `<eventInfo.eventClass>` element is defined, that value is used to determine the Event Domain defined in the full EDDY CER being built. The valid values for the `<eventInfo.eventClass>` range from 1-5 for each of the possible Event Domains (1 = Network, 2 = Security, 3 = System, 4 = Application, 5 = Environmental). If a value is specified for the `<eventInfo.eventClass>` element other than 1-5, the Template is ignored and no CER is produced.

Extraneous Template Elements

Ideally, a CER Factory Template would be comprised of the set of required elements as well as one or more optional elements. The CER Factory Template Server is flexible however when it comes to gracefully managing extraneous elements that do not correspond name-wise to required or optional elements in the Template it receives from a CER Factory Template Client.

In these situations, extraneous elements are placed into the EDDY CER as userTag elements in the eventInfo structure. For example, if an extraneous element such as “`<foo>bar</foo>`” were included in the CER Factory Template, it would be defined in the full EDDY CER built by the CER Factory Template Server as:

```
<userTag>  
  <key>foo</key>  
  <value>bar</value>  
</userTag>
```

In order to support extraneous elements defined in the CER Factory Template, the ‘extraneous’ configuration preference in the eddyconfig.xml must be set to true, otherwise extraneous element processing is disabled and a Template containing extraneous elements will be rejected by the CER Factory Template Server. The ‘extraneous’ configuration option is specified in the eddyconfig.xml as follows:

```
<extended>  
  <acquire>  
    <boolOption name="extraneous" value="true"/>  
  </acquire>  
</extended>
```

Extraneous elements in a CER Factory Template include true XML elements such as the “<foo>bar</foo>” example seen above, but also include XML element attributes. If extraneous element processing is enabled, all attributes defined in the Template will be stripped out and they will not be represented in the full EDDY CER as <userTag> elements.

In the case where extraneous elements are nested within one another as the following example illustrates, a <userTag> and corresponding key/value pair will be defined in the full EDDY CER for each nested element:

```
<foo1>
  bar1
  <foo2>
    bar2
  </foo2>
</foo1>

<userTag>
  <key>foo1</key>
  <value>bar1</value>
  <key>foo2</key>
  <value>bar2</value>
</userTag>
```

If extraneous elements are nested inside valid CER Factory Template elements, the extraneous element is stripped out and defined as a <userTag> element by the CER Factory Template Server and the processing of the valid Template element proceeds without the extraneous element present. For example, if the Template contains the following element:

```
<dataPayload.payloadType>2<foo>bar</foo></dataPayload.payloadType>
```

...The following <userTag> element is defined for the extraneous element:

```
<userTag>
  <key>foo</key>
  <value>bar</value>
</userTag>
```

...And the valid <dataPayload.payloadType> Template element is redefined and presented to the CER Factory Template Server for further processing as follows:

```
<dataPayload.payloadType>2</dataPayload.payloadType>
```

Malformed Templates

The CER Factory Template Server requires a well-formed (excluding the payload data) and semantically correct CER Factory Template to be sent to it by the CER Factory Template Client. A well-formed Template implies a properly defined XML document that conforms to

XML document standards. A semantically correct CER Factory template is one that contains at a minimum the required elements described above that are properly typed with one or more optional elements. Optional elements that are improperly typed according the full CER Template schema definition will **not** be considered malformed.

For simplicity and efficiency sake, the CER Factory Template Server will not notify the Template Client via an acknowledgement or other protocol of any malformed Templates that it receives and will not generate any errors that are logged or otherwise communicated. Upon receiving a malformed CER Factory Template, the Template Server will simply ignore the Template and move on to the next one it receives.

From the perspective of the Template Client it will assume that the Template is always processed by the Template Server. As a result, the burden of defining a well-formed XML document which is syntactically and semantically correct is its responsibility.

Template Type Validation

The CER Factory Template schema does not bind its optional elements to the same data types of the full CER schema; it simply defines all optional elements as strings. This is a natural consideration in light of the one-to-one mapping between the heavily typed nested/full CER structure and that of the largely untyped CER Factory Template. In order for a CER Factory Template to be considered correct and valid, it must match the types of the required and optional elements it defines to those in full CER schema definition.

With this in mind, an important aspect of the operation of the CER Factory Template Server is that it will not validate the data types of the optional CER Factory Template elements. This behavior was a design decision in light of the efficiencies gained by not having to validate data types for all Template elements. As a result, the burden for correct data type specification is left to the CER Factory Template Client.

For example, consider the following incorrectly typed Template definition for the `<eventInfo.startTime.utcTime>` element:

```
<eventInfo.startTime.utcTime>The time is two thirty PM.</eventInfo.startTime.utcTime>
```

The data type for this element should be defined as `xs:dateTime` and not as a text string. Regardless of this data typing error, the CER Factory Template Server will populate the full EDDY CER it builds with this improper value and move it along according to its routing rules to downstream EDDY Agents. As such, it is the responsibility of these downstream Agents to handle this malformed element data.

Template Semantic Validation

In order for a CER Factory Template to be accepted by the CER Factory Sever for generation of a full EDDY CER, it must be well-formed XML that coincides with the schema definition

for CER Factory Templates. Data typing of the optional elements that comprise the Template is not performed, but the Template itself must follow the element ordering and structure rules of the schema.

There are cases where data types are specified correctly for Template elements, but because of their values, there are semantic inconsistencies with other elements within the Template. In these cases, the CER Factory Server makes **no** attempt to disallow these semantic issues and forges ahead and produces a full EDDY CER with the inconsistent data. Again, it is the responsibility of the Template Client to populate the Template elements correctly and it is the downstream EDDY Agents receiving the CER to deal with any problems in the data that they receive.

A couple of examples will serve to illustrate how properly typed Template elements introduce semantic inconsistency into the Temple. First, consider the following template for a network event where the stop time of event is defined as coming before the start time:

```
<cerFactory>
  <eventInfo.oid>1005</eventInfo.oid>
  <!-- CBPD -->
  <eventInfo.startTime.utcTime>
    2007-07-05T12:20:00.000+00:00
  </eventInfo.startTime.utcTime>
  <eventInfo.stopTime.utcTime>
    2007-07-05T12:15:00.000+00:00
  </eventInfo.stopTime.utcTime>
  <eventInfo.eventClass>5</eventInfo.eventClass>
  <!-- Environmental -->
  <dataPayload.payloadType>2</dataPayload.payloadType>
  <!-- Cooked -->
  <dataPayload.payload>
    <cbpd>
      <cbpd-1.0.0>
        <heatSensor>
          <coverageArea>0.2</coverageArea>
          <heatSensorSetPoint>0.111</heatSensorSetPoint>
          <heatSensorLowerRange>0.003</heatSensorLowerRange>
          <heatSensorUpperRange>0.280</heatSensorUpperRange>
          <heatSensorAccuracy>0.001</heatSensorAccuracy>
          <timeConstant>0.1</timeConstant>
        </heatSensor>
      </cbpd-1.0.0>
    </cbpd>
  </dataPayload.payload>
</cerFactory>
```

The inclusion of these start and stop times out of order makes no sense and are inconsistent, but the CER Factory Server still produces a full CER EDDY from the Template data regardless.

Next, examine the next example of element semantic inconsistency. Here, the Event Domain is specified as a Network Event although the data payload is clearly an Environmental Event:

```
<cerFactory>
  <eventInfo.oid>1005</eventInfo.oid>
  <!-- CBPD -->
  <eventInfo.eventClass>1</eventInfo.eventClass>
  <!-- Network -->
  <dataPayload.payloadType>2</dataPayload.payloadType>
  <!-- Cooked -->
  <dataPayload.payload>
    <cbpd>
      <cbpd-1.0.0>
        <temperatureSensor>
          <temperatureSensorType>5</temperatureSensorType>
          <!-- Room temperature -->
          <temperatureSensorSetPoint>50.0</temperatureSensorSetPoint>
          <!-- The temperature value to be sensed. -->
          <temperatureSensorLowerRange>5.0</temperatureSensorLowerRange>
          <!-- The lower bound for operation of the temperature sensor. -->
          <temperatureSensorUpperRange>100.0</temperatureSensorUpperRange>
          <!-- The upper bound for operation of the temperature sensor. -->
          <accuracyOfTemperatureSensor>0.001</accuracyOfTemperatureSensor>
          <!-- The accuracy of the sensor. -->
          <timeConstant>00.1</timeConstant>
          <!-- The time constant of the sensor. -->
        </temperatureSensor>
      </cbpd-1.0.0>
    </cbpd>
  </dataPayload.payload>
</cerFactory>
```

Again, the CER Factory Template Server will dutifully produce a full EDDY CER with this semantically inconsistent data. As a result, the Template Server will place the Environmental Event inside the data payload wrapper of a Network Event which is clearly invalid for the data as well as the OID.

The CER Factory Template Server will ignore any optional elements that are defined in the CER Factory Template that are inconsistent with the Event Domain element. For example, if a Network Event Domain is defined and optional elements are included in the Template for an Analyzed event, those elements are not included in the full EDDY CER under construction since doing so would produce a XML document that did not conform to the XML structural layout of a full EDDY CER.

Restricted Template Elements

The schema for the CER Factory Template is a flattened version of the full EDDY CER schema. CER Factory Templates may contain the entire set of data of a full EDDY CER by specifying the complete set of required and optional elements. In all but a few cases, the elements in a CER Factory Template map one-to-one to the corresponding element in the full EDDY CER without restrictions.

The exceptions to the one-to-one Template to full EDDY CER mapping are the `<header.ttl>`, `<eventInfo.eventHostname>`, `<eventInfo.eventHostAddrType>` and `<eventHost.eventHostAddr>` elements. The elements are considered restricted because of their sensitive security nature with respect to spoofing potential and other reasons. While they may be defined in the CER Factory Template, they will not (in the case of the `<header.ttl>` element) and may not (for the `<eventInfo.eventHostName>`, `<eventInfo.eventHostAddrType>` and `<eventInfo.eventHostAddr>` elements) be copied into their respective full EDDY CER element fields.

When one or more of these restricted fields are defined and they are not copied directly into their respective full CER elements, they will be placed in the full EDDY CER as key/value fields with the `<userTag>` element. The following XML snippets illustrate the restricted field inclusion in the CER Factory Template followed by the XML that is placed into the `<userTag>` of the full EDDY CER:

- **Template:**

```
<header.ttl>
  128
</header.ttl>
```

EDDY CER:

```
<userTag>
  <key>header.ttl</key>
  <value>128</value>
</userTag>
```

- **Template:**

```
<eventInfo.eventHostname>
  cricket15.andrew.cmu.edu
</eventInfo.eventHostname>
```

EDDY CER:

```
<userTag>
  <key>eventInfo.eventHostname</key>
  <value>cricket15.andrew.cmu.edu</value>
</userTag>
```

- **Template:**

```
<eventInfo.eventHostAddrType>
  2
</eventInfo.eventHostAddrType>
```

EDDY CER:

```
<userTag>
  <key>eventInfo.eventHostAddrType</key>
  <value>2</value>
</userTag>
```

- **Template:**

```
<eventInfo.eventHostAddr>
  128.2.11.21
</eventInfo.eventHostAddr>
```

EDDY CER:

```
<userTag>  
  <key>eventInfo.eventHostAddr</key>  
  <value>128.2.11.21</value>  
</userTag>
```

The `<eventInfo.eventHostname>`, `<eventInfo.eventHostAddrType>` and `<eventHost.eventHostAddr>` elements defined in the Template will be copied to their respective full EDDY CER elements in the following cases, otherwise they will be defined as key/value `<userTag>` elements:

- `<eventInfo.eventHostname>` - If the defined element is the same FQDN as the CER Factory Template Client sending the Template to the CER Factory Server.
- `<eventInfo.eventHostAddrType>` - If the defined element holds the value of 2, the address type specifier for IP version 4 network addresses.
- `<eventInfo.eventHostAddr>` - If the defined element is the same IP address as the CER Factory Template Client sending the Template to the CER Factory Server.

Template Client Access Lists

The CER Factory Template Server restricts which CER Factory Template Clients can send Templates to it via an access control list. This Access List is defined in the `eddyconfig.xml` configuration file for the Template Server. The `eddyconfig.xml` is the standard method of defining configuration preferences for EDDY Agents built from the EDDY Framework.

The Template Client Access Lists are defined in the `eddyconfig.xml` file as a series of one or more lines with the first section of the line containing an IP address, a fully qualified domain name or an IP subnet followed by a comma separated list of EDDY OIDs of which the Client Template Server will generate full EDDY CERs for. If the Template Server receives a Template for an OID that has not been specified in the Access List, it will not generate an EDDY CER for it and will ignore the Template.

The CER Factory Template Client hosts allowed to send Templates to the Template Servers are specified by individual IP addresses, subnets or as a fully qualified domain name. IP addresses take the normal numerical form of *a.b.c.d* and FQDNs are specified such as *eddy1.andrew.cmu.edu*, etc. Subnets are defined with the IP network prefix following by a slash and then the number of significant network bits. For example, the subnet *128.2.255.255/16* will allow all hosts to connect to the Template Server in the IP range of 128.2.0.0 to 128.2.255.255 for any of the OIDs defined. Note that *128.2.255.255/16* is also equivalent to *128.2.0.0/16*.

The use of the “!” character preceding an Access List configuration line will disallow that host or subnet for the specified OIDs from connecting to the CER Factory Template Server. For

example, to not allow Template Client host *eddy6.andrew.cmu.edu* to send Templates for OID 1010, you would define the following Access List entry:

```
!eddy6.Andrew.cmu.edu:1010
```

The “*” character is used as a wildcard for both hosts and OIDs in the Template Client Access Lists. When used for hosts, the meaning of “*” is to allow **all** hosts for the specified OIDs to connect to the CER Factory Template Server. When specified in the OID field, the meaning is to allow Templates for all OIDs for the defined hosts. To allow all hosts to connect and support any Template OID, the following Access List can be defined:

```
*.*
```

By default, **all** access for hosts and OIDs is disabled, so in order to allow connectivity from any host and templates for any OID, they must be explicitly configured in the Access List lines of the *eddyconfig.xml* file.

The Template Client Access List configuration lines are additive and later entries will supercede earlier entries if there is a conflict. For example, consider the following Access List items:

```
128.2.11.110:*  
!128.2.11.110:1015
```

The first entry allows the host *128.2.11.110* to send CER Factory Templates for all OIDs. The second entry disallows OID 1015 for the same host. The next result of the additive nature of specifying CER Factory Client Access Lists is that these two entries support all OIDs except 1015 from host *128.2.11.110* to be processed by the CER Factory Template Server.

The Template Client Access List is defined in the `<extended><acquire><access>` configuration section in the *eddyconfig.xml* file. The following examples serve to illustrate some possible Access List configurations:

This Access List allows host *eddy1.andrew.cmu.edu* to send Templates containing OIDs 1010, 1011 and 1012 as well as host *eddy3.andrew.cmu.edu* sending Templates for all OIDs to the CER Factory Template Server:

```
<extended>  
  <acquire>  
    <strOption name="access">  
      <value>  
        eddy1.andrew.cmu.edu:1010,1011,1012  
        eddy3.andrew.cmu.edu:*  
      </value>  
    </strOption>  
  </acquire>  
</extended>
```

This Access List allows host *128.2.11.101* to send Templates containing OIDs 1000 and 1001 as well as hosts in the subnet *128.2.255.255/16* to send Templates for OID 1020 to the CER Factory Template Server:

```
<extended>
  <acquire>
    <strOption name="access">
      <value>
        128.2.11.101:1000,1001
        128.2.255.255/16:1020
      </value>
    </strOption>
  </acquire>
</extended>
```

This Access List allows hosts in the subnet *128.237.255.255/16* to send Templates for all OIDs to the CER Factory Template Server but not OID 1030 from host *128.237.34.121* (which is a host inside the *128.237.255.255/16* subnet):

```
<extended>
  <acquire>
    <strOption name="access">
      <value>
        128.237.255.255/16:*
        !128.237.34.121:1030
      </value>
    </strOption>
  </acquire>
</extended>
```

Template Transport Plug-Ins

The CER Factory Template Server supports a plug-in architecture for communication transports. It implements this with the use of loadable Java classes that are specified in the `eddyconfig.xml` configuration class. In order for these classes to be compatible with the CER Factory Template Server plug-in mechanism, they must implement a particular Java interface.

This release of the CER Factory software provides three transport communication plug-ins in the Template Server, two utilizing TCP sockets (unencrypted and SSL/TLS) and the third UNIX domain sockets. The choice of sockets was made to simplify the software requirement of the Template Client as well as reduce the amount of memory needed to communicate with the Template Server. Hence if a Client wishes to use the CER factory, all they need is some type of TCP stack. There is no software for either socket type provided for the CER Factory Template Client other than code that can be acquired from the examples available in the software distribution for Java, C and Perl. Future releases will include other transport plug-ins such as HTTP, HTTPS and SOAP.

The CER Factory Template Server supports a single configurable transport plug-in per executing Template Server. This limitation was a simplification decision in lieu of rapid development of the release of the software. The desired Template transport plug-in is selected

in the eddyconfig.xml configuration file as follows using the following configuration preference example:

```
<extended>
  <acquire>
    <strOption name="transport">
      <value>
        edu.cmu.acs.eddy.TransportPlugInTCP
      </value>
    </strOption>
  </acquire>
</extended>
```

The example configuration shows the sections and elements within the eddyconfig.xml file that are defined in order to specify the Template Server transport plug-in. In this particular example, the TCP unencrypted socket transport plug-in is loaded. For the SSL/TLS transport plug-in, use the class name edu.cmu.acs.eddy.TransportPlugInTLS. In order to load the UNIX domain socket transport plug-in, the edu.cmu.acs.eddy.TransportPlugInUDP class would be specified. If a plug-in transport is not defined in the eddyconfig.xml file, the TCP unencrypted socket transport is used as the default.

The CER Factory Template Server transport plug-ins support multiple simultaneous connections from Template Clients and funnel the transported Templates into a single processing queue inside the Template Server. The Template Server then proceeds to extract the Templates out in the order that they were received and generates full EDDY CERs from the Templates that are then routed out of the Template Server according to the defined routing rules for the Agent.

The TCP unencrypted socket transport plug-in provides only basic and simple socket connectivity. There is a single eddyconfig.xml configuration preference for the TCP socket transport plug-in shown as follows:

```
<extended>
  <acquire>
    <intOption name="sockPort" value="1500"/>
  </acquire>
</extended>
```

The *<sockPort>* configuration element specifies the TCP port that the transport plug-in will listen on to receive incoming connections from CER Factory Template Clients. If this configuration option is not specified, a default port value of 1400 is used as the default. The value range that can be specified for the *<sockPort>* configuration preference is 0 to 65,535.

The SSL/TLS socket transport plug-in provides mutually authenticated and encrypted communication between the CER Factory Template Client and Server. It requires more complex programming on the Template Client side than an unencrypted TCP socket with

the added benefit of being secure. There are three eddyconfig.xml configuration preferences for the SSL/TLS socket transport plug-in that are described as follows:

```
<extended>
  <acquire>
    <intOption name="sockPort" value="1400"/>
    <strOption name="keystore">
      <value>
        ../../eddycertificates/lib/eddyhost.ks
      </value>
    </strOption>
    <strOption name="keystorePwd">
      <value>
        eddypass
      </value>
    </strOption>
  </acquire>
</extended>
```

The `<sockPort>` configuration element specifies the TCP port that the SSL/TLS transport plug-in will listen on to receive incoming connections from CER Factory Template Clients. If this configuration option is not specified, a default port value of 1400 is used as the default. The value range that can be specified for the `<sockPort>` configuration preference is 0 to 65,535.

The `<keystore>` configuration element specifies the Java keystore file that contains the private and EDDY Certificate Authority signed public keys for the host the CER Factory Template Server is running on. If this configuration option is not specified, a default keystore location of “../../eddycertificates/lib/eddyhost.ks” is used as the default.

The `<keystorePwd>` configuration element specifies the password that is associated with the `<keystore>` configuration option. If this configuration option is not specified, a default password value of “eddypass” is used as the default.

The UNIX domain socket transport plug-in provides UNIX domain socket connectivity on UNIX systems such as Linux and Solaris. This transport mechanism is **not** supported on Windows systems and will not work on Windows systems. The domain socket that CER Factory Template Clients write their Templates to and the CER Factory Template Server reads from is specified using the `<sockName>` configuration preference seen below:

```
<extended>
  <acquire>
    <strOption name="sockName">
      <value>
        /tmp/cerfactory
      </value>
    </acquire>
</extended>
```

The `<sockName>` configuration element is the only preference option supported by the UNIX domain socket transport plug-in. If this configuration option is not defined, a default domain socket name of `“./cerFactory”` is used as the default.

Client & Server Protocol

The communication protocol between the CER Factory Template Client and the CER Factory Template Server operates identically regardless of the actual plug-in transport mechanism that is used. The communication protocol itself is very simple. The decision to implement a basic protocol was made in order to simplify the development process and get the CER Factory software up and running in a minimal amount of time and also to provide an efficient protocol that requires little processing overhead on both the Template Server and more importantly the Template Client. The consideration of the processing power utilized to transport CER Factory Templates from the Client is important in lieu of the processing restricted Clients that are typically used in a CER Factory deployment.

The CER Factory transport protocol provides the following features that illustrate both its capabilities and limitations:

- A connection between the CER Factory Template Client and the Template Server is initiated by the Template Client to a listening Template Server. If the Template Client is a host that is allowed to send Templates as validated against the Template Access List, the Template Server will accept the Template sent from the Template Client, otherwise it will close the connection.

Once connected, the Template Client sends a Template to the Template Server. The Template Server will then process the Template it receives if the OID of the Template matches those OIDs that are allowed for the connecting Template Client as validated against the Template Access List.

- One or more Templates may be transmitted between the Template Client and Template Server for each connection that is established between them. The CER Factory design allows for persistent connections where a connection is established and more than one Template is sent between the Template Client and Server.
- There is no framing of the Templates transmitted between CER Factory Template Client and Server. Because of the nature of the Templates being sent such that they are XML documents with clear start and stop delimiters in the opening and closing tags and that reliable transport plug-ins such as TCP and UNIX Domain sockets are used, there is no need for a framing mechanism.
- There are no acknowledgements sent between the CER Factory Template Client and Template Server for either the receipt of a complete Template or for the success or failure of the internal formatting of that Template. The lack of acknowledgements between the Template Client and Template Server was a

design decision meant to increase efficiency and reduce processing overhead in the CER Factory architecture as well as simplify the design in order to get the CER Factory software up and running quickly.

As a result of this lack of acknowledgements, the CER Factory Template Client is not informed when it send malformed Templates to the Template Server. It is therefore the responsibility and burden of the Template Client to make sure the Templates it is sending to the Template Server are properly constructed so that a full EDDY CER can be generated from them.

The lack of acknowledgements is also relevant when there is misconfiguration of the Template Client Access List. Since the Template Server will ignore any Template it receives that contains an unsupported OID, it is important for the Access List to be defined correctly and tested adequately to verify the OIDs specified in the Templates are resulting in full EDDY CERs being produced by the Template Server.

- The CER Factory Template Server maintains a queue of Templates it receives from the Template Clients connected to it. The multi-threaded nature of the Template Server requires a Template queue that can accommodate a number of Templates it has read, but has not yet processed. The construction of full EDDY CERs from Templates by the Template Server is done in a one-by-one manner working from the Template queue.

The `<queueSize>` configuration preference in the eddyconfig.xml file is used to define the size of the Template queue in the CER Factory Template Server. This value defines the maximum size of the queue and the maximum number of Templates that can be queued up waiting for processing by the Template Server. Once the `<queueSize>` limit has been reached, no additional Templates can be added to the Template queue and they are discarded without further processing and no acknowledgement of their discarded state is communicated to the CER Factory Template Client sending the Template.

The `<queueSize>` configuration preference is defined in the eddyconfig.xml configuration file with a Template queue size of 1,500 as demonstrated in the following example:

```
<extended>  
  <acquire>  
    <intOption name="queueSize" value="1500"/>  
  </acquire>  
</extended>
```

The default size of the CER Factory Template Server Template queue is 1,000. This default value is used if the `<queueSize>` configuration preference is not defined in the CER Factory Template Server's eddyconfig.xml file. The value

range that can be specified for the `<queueSize>` configuration preference is 1 to 2,147,483,647.

- The CER Factory Template Server will timeout a connection made to it by the Template Client if it doesn't receive the next byte of the Template within the default time period of 60 seconds from the last byte of the Template received. This timeout functionality is provided in the Template Server to avoid maintaining connections indefinitely for Template Clients that have failed in one way or another.

The `<timeout>` configuration preference in the `eddyconfig.xml` configuration file may be defined to use a timeout value other than 60 seconds as seen below:

```
<extended>  
  <acquire>  
    <intOption name="timeout" value="120"/>  
  </acquire>  
</extended>
```

The example above illustrates a connection timeout period that is configured for 120 seconds, or 2 minutes. If the `<timeout>` element is set to a value of 0, connection timeout functionality is disabled. The value range that can be specified for the `<timeout>` configuration preference is 0 to 2,147,483,647

- Once a Template has been transmitted from the CER Factory Template Client to the Server, the Template Client would typically close the connection it established in order to send the Template to the Server. If the Template Client does not close the connection within 5 seconds of sending the last byte of the Template, the Template Server will take on the responsibility of closing the connection.

Appendix

A

CER Factory Template Examples

Overview

The following CER Factory Templates and corresponding full EDDY CERs produced from those Templates provide a few examples demonstrating the operation of the EDDY CER Factory. The CER Factory Template Server always attempts a 'best effort' to generate a full EDDY CER from the Template it receives and is only not able to do so when the Template does not provide required Template elements or is non-well-formed XML.

Correct Template, Required Elements

The following example demonstrates a properly defined CER Factory Template and the resulting full EDDY CER produced from it. The Template includes all of the required Template elements and no optional elements:

CER Factory Template:

```
<cerFactory>
  <eventInfo.oid>1005</eventInfo.oid>
  <!-- CBPD -->
  <dataPayload.payloadType>2</dataPayload.payloadType>
  <!-- Cooked -->
  <dataPayload.payload>
    <cbpd>
      <cbpd-1.0.0>
        <heatSensor>
          <coverageArea>500.0</coverageArea>
          <heatSensorSetPoint>27.1</heatSensorSetPoint>
          <heatSensorLowerRange>0.0</heatSensorLowerRange>
          <heatSensorUpperRange>100.0</heatSensorUpperRange>
          <heatSensorAccuracy>0.1</heatSensorAccuracy>
          <timeConstant>0.1</timeConstant>
        </heatSensor>
      </cbpd-1.0.0>
    </cbpd>
  </dataPayload.payload>
</cerFactory>
```

Full EDDY CER:

```
<?xml version="1.0" encoding="UTF-8"?>
<commonEventRecord>
  <header>
    <version>0.69</version>
    <ttl>128</ttl>
  </header>
  <eventInfo>
    <version>0.69</version>
    <oid>1005</oid>
```

```
<eventID>0</eventID>
<occurredStamp>
  <startTime>
    <utcTime>2007-06-28T15:15:15.000+00:00</utcTime>
  </startTime>
  <timeMethod>1</timeMethod>
  <agentTime>
    <utcTime>2007-06-28T15:15:15.000+00:00</utcTime>
  </agentTime>
  <agentTimeMethod>1</agentTimeMethod>
</occurredStamp>
<eventHostname>eddy3.andrew.cmu.edu</eventHostname>
<eventHostAddr>
  <ipv4Addr>128.2.11.103</ipv4Addr>
</eventHostAddr>
<normalizerHostname>eddy3.andrew.cmu.edu</normalizerHostname>
<normalizerAddr>
  <ipv4Addr>128.2.11.103</ipv4Addr>
</normalizerAddr>
<normalizerAddrType>
  <ipv4Addr>128.2.11.103</ipv4Addr>
</normalizerAddrType>
<corrDescriptor>
  <version>0.69</version>
  <eventClass>5</eventClass>
  <warningLevel>7</warningLevel>
  <eventCorrDescriptor>
    <environmentalEventCorrelationDescriptor>
      <version>0.69</version>
    </environmentalEventCorrelationDescriptor>
  </eventCorrDescriptor>
</corrDescriptor>
</eventInfo>
<dataPayload>
  <cookedDataPayload>
    <environmental>
      <cbpd>
        <cbpd-1.0.0>
          <heatSensor>
            <coverageArea>500.0</coverageArea>
            <heatSensorSetPoint>27.1</heatSensorSetPoint>
            <heatSensorLowerRange>0.0</heatSensorLowerRange>
            <heatSensorUpperRange>110.0</heatSensorUpperRange>
            <heatSensorAccuracy>0.1</heatSensorAccuracy>
            <timeConstant>0.1</timeConstant>
          </heatSensor>
        </cbpd-1.0.0>
      </cbpd>
    </environmental>
  </cookedDataPayload>
</dataPayload>
</commonEventRecord>
```

Correct Template, Required & Optional Elements

The following example demonstrates a properly defined CER Factory Template and the resulting full EDDY CER produced from it. The Template includes all of the required Template elements as well as the optional *<eventInfo.eventClass>* and *<eventInfo.warningLevel>* elements.

CER Factory Template:

```
<cerFactory>
  <eventInfo.oid>1001</eventInfo.oid>
  <!-- Argus -->
  <eventInfo.eventClass>1</eventInfo.eventClass>
```

```
<!-- Network -->
<eventInfo.warningLevel>6</eventInfo.warningLevel>
<!-- Warning condition -->
<dataPayload.payloadType>1</dataPayload.payloadType>
<!-- Raw -->
<dataPayload.payload>
  ASAA7ACACAB/AAABAAABcAFIAAAAAAF5RDRO6wAGTUVENE7rAAZNRcCoAQHv
  ///6EQARNQdsitYAAAAABAAAAAAAEAAAFqAAABQAAAAAAAGQ
  CAAADYgr27kBAF5///pCIQAATk9USUZZICogSFRUUC8xLjENckhPULQ6IDIz
  OS4yNTUuMjU1LjI1MDoxOTAwDQpDQUNIRS1DT05UUk9MOiBtYXgtYWdlPTEy
  MA0KTE9DQVRJT046IGH0dHA6Ly8xOTIuMTY4LjEuMTU1Njc4L2lnZC54bWwN
  Ck5UOiB1cm46c2M=
</dataPayload.payload>
</cerFactory>
```

Full EDDY CER:

```
<?xml version="1.0" encoding="UTF-8"?>
<commonEventRecord>
  <header>
    <version>0.69</version>
    <ttl>128</ttl>
  </header>
  <eventInfo>
    <version>0.69</version>
    <oid>1001</oid>
    <eventID>0</eventID>
    <occurredStamp>
      <startTime>
        <utcTime>2007-06-28T15:15:15.000+00:00</utcTime>
      </startTime>
      <timeMethod>1</timeMethod>
      <agentTime>
        <utcTime>2007-06-28T15:15:15.000+00:00</utcTime>
      </agentTime>
      <agentTimeMethod>1</agentTimeMethod>
    </occurredStamp>
    <eventHostname>eddy3.andrew.cmu.edu</eventHostname>
    <eventHostAddr>
      <ipv4Addr>128.2.11.103</ipv4Addr>
    </eventHostAddr>
    <normalizerHostname>eddy3.andrew.cmu.edu</normalizerHostname>
    <normalizerAddr>
      <ipv4Addr>128.2.11.103</ipv4Addr>
    </normalizerAddr>
    <normalizerAddrType>
      <ipv4Addr>128.2.11.103</ipv4Addr>
    </normalizerAddrType>
    <corrDescriptor>
      <version>0.69</version>
      <eventClass>1</eventClass>
      <warningLevel>6</warningLevel>
      <eventCorrDescriptor>
        <networkEventCorrDescriptor>
          <version>0.69</version>
          <netProto>0</netProto>
          <srcAddr>
            <ipv4Addr>0.0.0.0</ipv4Addr>
          </srcAddr>
          <dstAddr>
            <ipv4Addr>0.0.0.0</ipv4Addr>
          </dstAddr>
          <srcPort>0</srcPort>
          <dstPort>0</dstPort>
          <direction>bi</direction>
        </networkEventCorrDescriptor>
      </eventCorrDescriptor>
    </corrDescriptor>
  </eventInfo>
  <dataPayload>
```

```
<rawDataPayload>
  <rawEvent>
    ASAA7ACACAB/AAABAAABcAFIAAAAAAF5RDRO6wAGTUVENE7rAAZNRcCoAQHv
    ///6EQARNQdsitYAAAAABAAAAAAAEAAAFqAAABQAAAAAAAAAAAAAAAAAgQ
    CAAADYgr27kBAF5///pCIQAATk9USUZIZICogSFRUUC8xLjENCkhPU1Q6IDIz
    OS4yNTUuMjU1LjI1MDoxOTAwdQpDQUNIRS1DT05UUK9MOiBtYXgtYWdlPTEy
    MA0KTE9DQVRJT046IGh0dHA6Ly8xOTIuMTY4LjEuMTc4LjI1LjI1LjI1LjI1LjI1
    Ck5UOiBlcm46c2M=
  </rawEvent>
</rawDataPayload>
</dataPayload>
</commonEventRecord>
```

Correct Template, Required & Extraneous Elements

The following example demonstrates a properly defined CER Factory Template and the resulting full EDDY CER produced from it. The Template includes all of the required Template elements as well as the extraneous *<bogus>* element. Note that the *<bogus>* element is defined in the full EDDY CER as a *<userTag>* element.

CER Factory Template:

```
<cerFactory>
  <eventInfo.oid>1005</eventInfo.oid>
  <!-- CBPD -->
  <dataPayload.payloadType>2</dataPayload.payloadType>
  <!-- Cooked -->
  <dataPayload.payload>
    <cbpd>
      <cbpd-1.0.0>
        <temperatureSensor>
          <temperatureSensorType>5</temperatureSensorType>
          <temperatureSensorSetPoint>50.0</temperatureSensorSetPoint>
          <temperatureSensorLowerRange>5.0</temperatureSensorLowerRange>
          <temperatureSensorUpperRange>100.0</temperatureSensorUpperRange>
          <accuracyOfTemperatureSensor>0.001</accuracyOfTemperatureSensor>
          <timeConstant>00.1</timeConstant>
        </temperatureSensor>
      </cbpd-1.0.0>
    </cbpd>
  </dataPayload.payload>
  <bogus>123</bogus>
  <!-- Extraneous element -->
</cerFactory>
```

Full EDDY CER:

```
<?xml version="1.0" encoding="UTF-8"?>
<commonEventRecord>
  <header>
    <version>0.69</version>
    <ttl>128</ttl>
  </header>
  <eventInfo>
    <version>0.69</version>
    <oid>1005</oid>
    <eventID>0</eventID>
    <occurredStamp>
      <startTime>
        <utcTime>2007-06-28T15:15:15.000+00:00</utcTime>
      </startTime>
      <timeMethod>1</timeMethod>
      <agentTime>
        <utcTime>2007-06-28T15:15:15.000+00:00</utcTime>
      </agentTime>
      <agentTimeMethod>1</agentTimeMethod>
    </occurredStamp>
  </eventInfo>
</commonEventRecord>
```

```

</occurredStamp>
<eventHostname>eddy3.andrew.cmu.edu</eventHostname>
<eventHostAddr>
  <ipv4Addr>128.2.11.103</ipv4Addr>
</eventHostAddr>
<normalizerHostname>eddy3.andrew.cmu.edu</normalizerHostname>
<normalizerAddr>
  <ipv4Addr>128.2.11.103</ipv4Addr>
</normalizerAddr>
<normalizerAddrType>
  <ipv4Addr>128.2.11.103</ipv4Addr>
</normalizerAddrType>
<corrDescriptor>
  <version>0.69</version>
  <eventClass>5</eventClass>
  <warningLevel>7</warningLevel>
  <eventCorrDescriptor>
    <environmentalEventCorrelationDescriptor>
      <version>0.69</version>
    </environmentalEventCorrelationDescriptor>
  </eventCorrDescriptor>
</corrDescriptor>
<userTag>
  <key>bogus</key>
  <value>123</value>
</userTag>
</eventInfo>
<dataPayload>
  <cookedDataPayload>
    <environmental>
      <cbpd>
        <cbpd-1.0.0>
          <temperatureSensor>
            <temperatureSensorType>5</temperatureSensorType>
            <!-- Room temperature -->
            <temperatureSensorSetPoint>50.0</temperatureSensorSetPoint>
            <!-- The temperature value to be sensed. -->
            <temperatureSensorLowerRange>
              5.0
            </temperatureSensorLowerRange>
            <!-- The lower bound for operation of the temperature sensor. -->
            <!-- -->
            <temperatureSensorUpperRange>
              100.0
            </temperatureSensorUpperRange>
            <!-- The upper bound for operation of the temperature sensor. -->
            <!-- -->
            <accuracyOfTemperatureSensor>
              0.001
            </accuracyOfTemperatureSensor>
            <!-- The accuracy of the sensor. -->
            <timeConstant>00.1</timeConstant>
            <!-- The time constant of the sensor. -->
          </temperatureSensor>
        </cbpd-1.0.0>
      </cbpd>
    </environmental>
  </cookedDataPayload>
</dataPayload>
</commonEventRecord>

```

Correct Template, Required & Restricted Elements

The following example demonstrates a properly defined CER Factory Template and the resulting full EDDY CER produced from it. The Template includes all of the required Template elements as well as the restricted `<eventInfo.eventHostname>` element. Note that

the `<eventInfo.eventHostname>` element is defined in the full EDDY CER as a `<userTag>` element.

CER Factory Template:

```
<cerFactory>
  <eventInfo.oid>1011</eventInfo.oid>
  <!-- SenSCIR -->
  <eventInfo.eventHostname>cricket3.andrew.cmu.edu</eventInfo.eventHostname>
  <dataPayload.payloadType>2</dataPayload.payloadType>
  <!-- Cooked -->
  <dataPayload.payload>
    <senscir>
      <senscir-1.0.0>
        <temperature>75.4</temperature>
      </senscir-1.0.0>
    </senscir>
  </dataPayload.payload>
</cerFactory>
```

Full EDDY CER:

```
<?xml version="1.0" encoding="UTF-8"?>
<commonEventRecord xmlns="http://www.cmu.edu/eddy">
  <header xmlns="">
    <version>0.69</version>
    <ttl>128</ttl>
  </header>
  <eventInfo xmlns="">
    <version>0.69</version>
    <oid>1011</oid>
    <eventID>0</eventID>
    <occurredStamp>
      <startTime>
        <utcTime>2007-07-25T12:15:15.000+00:00</utcTime>
      </startTime>
      <timeMethod>1</timeMethod>
      <agentTime>
        <utcTime>2007-07-25T12:15:15.000+00:00</utcTime>
      </agentTime>
      <agentTimeMethod>1</agentTimeMethod>
    </occurredStamp>
    <eventHostname>censcirgw2.andrew.cmu.edu</eventHostname>
    <eventHostAddr>
      <ipv4Addr>128.2.11.124</ipv4Addr>
    </eventHostAddr>
    <normalizerHostname>eddy6.andrew.cmu.edu</normalizerHostname>
    <normalizerAddr>
      <ipv4Addr>128.2.11.106</ipv4Addr>
    </normalizerAddr>
    <normalizerAddrType>
      <ipv4Addr>128.2.11.106</ipv4Addr>
    </normalizerAddrType>
    <corrDescriptor>
      <version>0.69</version>
      <eventClass>5</eventClass>
      <warningLevel>7</warningLevel>
      <eventCorrDescriptor>
        <environmentalEventCorrelationDescriptor>
          <version>0.69</version>
        </environmentalEventCorrelationDescriptor>
      </eventCorrDescriptor>
    </corrDescriptor>
    <userTag>
      <key>eventInfo.eventHostname</key>
      <value>cricket3.andrew.cmu.edu</value>
    </userTag>
  </eventInfo>
  <dataPayload xmlns="">
```

```
<cookedDataPayload>
  <environmental>
    <senscir>
      <senscir-1.0.0>
        <temperature>75.4</temperature>
      </senscir-1.0.0>
    </senscir>
  </environmental>
</cookedDataPayload>
</dataPayload>
</commonEventRecord>
```

Incorrect Template, Missing Required Elements

The following example demonstrates a improperly defined CER Factory Template that is missing one of the required Template elements. The missing Template element is `<dataPayload.payloadType>`. Since a required Template element is missing, the CER Factory Template Server is unable to produce a full EDDY CER from the Template.

CER Factory Template:

```
<cerFactory>
  <eventInfo.oid>1010</eventInfo.oid>
  <!-- Network Key/Value -->
  <eventInfo.warningLevel>1</eventInfo.warningLevel>
  <!-- Informational-->
  <eventInfo.userTag>
    <!-- SNMP MIBS -->
    <key>interfaces.ifTable.ifEntry.ifInOctets.20</key>
    <value>1520103453</value>
    <key>interfaces.ifTable.ifEntry.ifOperStatus.25</key>
    <value>up(1)</value>
  </eventInfo.userTag>
  <!--<dataPayload.payloadType>1</dataPayload.payloadType-->
  <!-- Raw -->
  <!-- Required element commented -->
  <dataPayload.payload/>
  <!-- No Payload -->
</cerFactory>
```

Incorrect Template, Malformed Template

The following example demonstrates a improperly defined CER Factory Template. It is not a well-formed XML document and as a result, the CER Factory Template Server is unable to produce a full EDDY CER from it.

CER Factory Template:

This is an example of a completely malformed CER Factory template.

Appendix

B

CER Factory Template Client

Overview

Writing Templates to the CER Factory Server is a relatively simple task. A Template Client is required to craft a Template and then send it to the CER Factory Server over a TCP (unencrypted and SSL/TLS) or UNIX Domain socket. The following sample code illustrates how this is achieved for all three currently supported transport mechanisms by sending a Template file stored on disk to the Template Server.

Source Code

```
#include <openssl/ssl.h>
#include <netdb.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/un.h>

char cert[256] = "";
int display = 0;
char file[256] = "";
char keys[256] = "";
int port = -1;
int repeat = -1;
char server[256] = "";
char uds[256] = "";

void usage() {
    printf("usage:  cerfclient [-d] -f<template> [-h] [-r<delay>]\n");
    printf("                ([-c<rootcert> -k<keys>] -p<port> -s<host> | -u<socket>)\n");
    printf("options: -c<rootcert>  CA root certificate file\n");
    printf("          -d            display CER Factory Template file\n");
    printf("          -f<template> CER Factory Template file\n");
    printf("          -h            display this usage list and exit\n");
    printf("          -k<key>      keys and CA signed certificate file\n");
    printf("          -p<port>     CER Factory Template Server port\n");
    printf("          -r<delay>    send Template file every <delay> milliseconds\n");
    printf("          -s<host>     CER Factory Template Server host/address\n");
    printf("          -u<socket>   CER Factory Template Server domain socket\n");
    printf("version: 1.0.6 (4.December.2007)\n");
    printf("author:  Jim Gargani (jgargani@cmu.edu)\n");
}

int checkArgs(int argc, char *argv[]) {
    int argIndex;
    char buffer[128];

    for(argIndex = 1; argIndex < argc; ++argIndex) {
        if(strlen(argv[argIndex]) < 2) {
            usage();
            return(1);
        }
    }
}
```

```
}  
  
if(buffer[2] = '\0', strcmp(strncpy(buffer, argv[argIndex], 2), "-c") == 0) {  
    if(strcmp(cert, "") == 0) {  
        if(strlen(argv[argIndex]) > 2) {  
            strcpy(cert, argv[argIndex] + 2);  
        } else {  
            usage();  
            return(1);  
        }  
    } else {  
        usage();  
        return(1);  
    }  
}  
} else if(buffer[2] = '\0', strcmp(strncpy(buffer, argv[argIndex], 2), "-d") == 0) {  
    if(strlen(argv[argIndex]) != 2) {  
        usage();  
        return(1);  
    } else {  
        if(display == 0) {  
            display = 1;  
        } else {  
            usage();  
            return(1);  
        }  
    }  
}  
} else if(buffer[2] = '\0', strcmp(strncpy(buffer, argv[argIndex], 2), "-f") == 0) {  
    if(strcmp(file, "") == 0) {  
        if(strlen(argv[argIndex]) > 2) {  
            strcpy(file, argv[argIndex] + 2);  
        } else {  
            usage();  
            return(1);  
        }  
    } else {  
        usage();  
        return(1);  
    }  
}  
} else if(buffer[2] = '\0', strcmp(strncpy(buffer, argv[argIndex], 2), "-k") == 0) {  
    if(strcmp(keys, "") == 0) {  
        if(strlen(argv[argIndex]) > 2) {  
            strcpy(keys, argv[argIndex] + 2);  
        } else {  
            usage();  
            return(1);  
        }  
    } else {  
        usage();  
        return(1);  
    }  
}  
} else if(buffer[2] = '\0', strcmp(strncpy(buffer, argv[argIndex], 2), "-p") == 0) {  
    if(port == -1) {  
        if(strlen(argv[argIndex]) > 2) {  
            port = atoi(argv[argIndex] + 2);  
            if((port < 0) || (port > 65535)) {  
                usage();  
                return(1);  
            }  
        } else {  
            usage();  
            return(1);  
        }  
    } else {  
        usage();  
        return(1);  
    }  
}  
} else if(buffer[2] = '\0', strcmp(strncpy(buffer, argv[argIndex], 2), "-r") == 0) {  
    if(repeat == -1) {  
        if(strlen(argv[argIndex]) > 2) {  
            repeat = atoi(argv[argIndex] + 2);  
            if(repeat < 0) {  
                usage();  
                return(1);  
            }  
        } else {  
            usage();  
            return(1);  
        }  
    } else {  
        usage();  
        return(1);  
    }  
}
```

```
    }
} else if(buffer[2] = '\\0', strcmp(strncpy(buffer, argv[argIndex], 2), "-s") == 0) {
    if(strcmp(server, "") == 0) {
        if(strlen(argv[argIndex]) > 2) {
            strcpy(server, argv[argIndex] + 2);
        } else {
            usage();
            return(1);
        }
    } else {
        usage();
        return(1);
    }
} else if(buffer[2] = '\\0', strcmp(strncpy(buffer, argv[argIndex], 2), "-u") == 0) {
    if(strcmp(uds, "") == 0) {
        if(strlen(argv[argIndex]) > 2) {
            strcpy(uds, argv[argIndex] + 2);
        } else {
            usage();
            return(1);
        }
    } else {
        usage();
        return(1);
    }
} else if(buffer[2] = '\\0', strcmp(strncpy(buffer, argv[argIndex], 2), "-h") == 0) {
    usage();
    return(1);
} else {
    usage();
    return(1);
}
}

if(strcmp(file, "") == 0) {
    usage();
    return(1);
}

if((strcmp(server, "") == 0) && (strcmp(uds, "") == 0)) {
    usage();
    return(1);
}

if((strcmp(server, "") != 0) && (strcmp(uds, "") != 0)) {
    usage();
    return(1);
}

if(strcmp(server, "") != 0) {
    if((strcmp(cert, "") != 0) && (strcmp(keys, "") == 0)) {
        usage();
        return(1);
    }

    if((strcmp(cert, "") == 0) && (strcmp(keys, "") != 0)) {
        usage();
        return(1);
    }
}

if(port == -1) {
    usage();
    return(1);
}

if(strcmp(uds, "") != 0) {
    if(strcmp(cert, "") != 0) {
        usage();
        return(1);
    }
}

if(strcmp(keys, "") != 0) {
    usage();
    return(1);
}

if(port != -1) {
    usage();
    return(1);
}
}
```

```
    }
    return(0);
}

int writeTemplate(char *templateString) {
    struct sockaddr_in cerFactoryAddr;
    struct hostent *cerFactoryHost;
    int cerFactorySock;

    if(strcmp(server, "") != 0) {
        cerFactorySock = socket(AF_INET, SOCK_STREAM, 0);

        cerFactoryHost = gethostbyname(server);
        if(cerFactoryHost == '\0') {
            printf("error: The CER Factory Template Server is unreachable.\n");
            return(1);
        }

        bcopy(cerFactoryHost->h_addr, &cerFactoryAddr.sin_addr.s_addr, cerFactoryHost->h_length);
        cerFactoryAddr.sin_family = AF_INET;
        cerFactoryAddr.sin_port = htons(port);

        if(connect(cerFactorySock, (struct sockaddr *)&cerFactoryAddr, sizeof(cerFactoryAddr)) == -1) {
            printf("error: The CER Factory Template Server is unreachable.\n");
            close(cerFactorySock);
            return(1);
        }

        if(strcmp(cert, "") != 0) {
            SSL_library_init();

            SSL_METHOD *sslMethod = SSLv23_method();
            SSL_CTX *sslCtx = SSL_CTX_new(sslMethod);

            if(!SSL_CTX_use_certificate_chain_file(sslCtx, keys)) {
                printf("error: The CER Factory client keys/certificate file is invalid.\n");
                SSL_CTX_free(sslCtx);
                close(cerFactorySock);
                return(1);
            }

            if(!SSL_CTX_use_PrivateKey_file(sslCtx, keys, SSL_FILETYPE_PEM)) {
                printf("error: The CER Factory client keys/certificate file is invalid.\n");
                SSL_CTX_free(sslCtx);
                close(cerFactorySock);
                return(1);
            }

            if(!SSL_CTX_load_verify_locations(sslCtx, cert, 0)) {
                printf("error: The certificate authority root certificate is invalid.\n");
                SSL_CTX_free(sslCtx);
                close(cerFactorySock);
                return(1);
            }

            SSL *ssl = SSL_new(sslCtx);
            BIO *bio = BIO_new_socket(cerFactorySock, BIO_NOCLOSE);
            SSL_set_bio(ssl, bio, bio);
            SSL_connect(ssl);

            if(SSL_get_verify_result(ssl) != X509_V_OK) {
                printf("error: The CER Factory server certificate is invalid.\n");
                SSL_free(ssl);
                SSL_CTX_free(sslCtx);
                close(cerFactorySock);
                return(1);
            }

            SSL_write(ssl, templateString, strlen(templateString));

            // Yes, we need to call it twice to shutdown both sides of the connection.
            SSL_shutdown(ssl);
            SSL_shutdown(ssl);

            SSL_free(ssl);
            SSL_CTX_free(sslCtx);
        } else {
            write(cerFactorySock, templateString, strlen(templateString));
        }
    }
}
```

```
    }

    close(cerFactorySock);

    return(0);
} else {
    cerFactorySock = socket(AF_UNIX, SOCK_STREAM, 0);

    struct sockaddr_un remoteSockAddr;
    remoteSockAddr.sun_family = AF_UNIX;
    strcpy(remoteSockAddr.sun_path, uds);

    if (connect(cerFactorySock, (struct sockaddr *)&remoteSockAddr, strlen(remoteSockAddr.sun_path) +
        sizeof(remoteSockAddr.sun_family)) == -1) {
        printf("error: The CER Factory Template Server is unreachable.\n");
        close(cerFactorySock);
        return(1);
    }

    send(cerFactorySock, templateString, strlen(templateString), 0);

    close(cerFactorySock);

    return(0);
}
}

int main(int argc, char *argv[]) {
    char templateBuffer[256];
    FILE *templateFile;
    char templateString[32768];

    if(argc > 7) {
        usage();
        exit(1);
    } else {
        if(checkArgs(argc, argv) == 1) {
            exit(1);
        } else {
            templateFile = fopen(file, "r");

            if(templateFile == 0) {
                printf("error: The CER Factory Template file does not exist.\n");
                exit(1);
            }

            while(fgets(templateBuffer, sizeof(templateBuffer), templateFile) != 0) {
                strcat(templateString, templateBuffer);
            }

            fclose(templateFile);

            if(display == 1) {
                printf("%s", templateString);
            }

            if(repeat == -1) {
                if(writeTemplate(templateString) == 1) {
                    exit(1);
                }
            }

            exit(0);
        } else {
            while(1) {
                if(writeTemplate(templateString) == 1) {
                    exit(1);
                }

                usleep(repeat * 1000);
            }
        }
    }
}
}
```

Appendix

C

CER Factory Template Schema

Overview

The CER Factory Template XML schema definition is a special schema in the EDDY system that is used to define and bind CER Factory Templates. A valid CER Factory Template is bound to the CER Factory Template schema below as it describes the formatting syntax and semantics Templates must take on.

A CER Factory Template must contain a set of required schema XML elements (*<eventInfo.oid>*, *<dataPayload.payloadType>* and *<dataPayload.payload>*) and may contain one or more optional XML elements. Elements present in the Template that are not part of the schema definition below are considered extraneous elements and will be built as *<userTag>* elements if the 'extraneous' configuration preference is set to 'true' in the *eddyconfig.xml* file.

If extraneous elements processing is enabled, the CER Factory Template Server pre-processes the Template by first stripping out extraneous elements and then presenting the remaining Template to the CER Factory Template schema validation code. This code verifies that the Template is correct as bound against the CER Factory Template Schema. If extraneous element processing is not enabled, any specified Template element that is not part of the schema definition will result in the CER Factory Template Server rejecting the Template.

Template Schema

```
<?xml version="1.0" encoding="UTF-8"?>

<!--
  EDDY Common Event Record Factory XML Schema Definition.
  Version: 1.0.0
  Last Modified: 13.July.2007
  By: Jim Gargani (jgargani@cmu.edu)
-->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="cerFactory" type="cerFactoryStruct" />

  <xs:complexType name="cerFactoryStruct">
    <xs:sequence>
      <xs:element name="comment" type="xs:string" minOccurs="0" />
      <!-- comment [0-1] -->

      <!-- ***** -->
      <!-- BEGIN: header -->
      <!-- ***** -->
      <xs:element name="header.version" type="xs:string" minOccurs="0" />
    
```

CARNEGIE MELLON UNIVERSITY
SOFTWARE RELEASE 0.5.2-4, DOCUMENT VERSION 1.20

```
<!-- header.version [1] -->
<xs:element name="header.ttl" type="xs:string" minOccurs="0"/>
<!-- header.ttl [1] -->
<!-- ***** -->
<!-- END: header -->
<!-- ***** -->

<!-- ***** -->
<!-- BEGIN: eventInfo -->
<!-- ***** -->
<xs:element name="eventInfo.version" type="xs:string" minOccurs="0"/>
<!-- eventInfo.version [1] -->
<xs:element name="eventInfo.oid" type="xs:unsignedLong"/>
<!-- eventInfo.oid [1] ***REQUIRED CER Factory element*** -->
<xs:element name="eventInfo.eventID" type="xs:string" minOccurs="0"/>
<!-- eventInfo.eventID [1] -->

<!-- ***** -->
<!-- BEGIN: eventInfo.occurredStamp -->
<!-- ***** -->
<xs:element name="eventInfo.startTime.utcTime" type="xs:string"
  minOccurs="0"/>
<!-- eventInfo.occurredStamp.startTime.utcTime [1] -->
<xs:element name="eventInfo.startTime.subseconds" type="xs:string"
  minOccurs="0"/>
<!-- eventInfo.occurredStamp.startTime.subseconds [0-1] -->
<!-- In CER schema, eventInfo.occurredStamp.stopTime is [0-1] -->
<xs:element name="eventInfo.stopTime.utcTime" type="xs:string"
  minOccurs="0"/>
<!-- eventInfo.occurredStamp.stopTime.utcTime [1] -->
<xs:element name="eventInfo.stopTime.subseconds" type="xs:string"
  minOccurs="0"/>
<!-- eventInfo.occurredStamp.stopTime.subseconds [0-1] -->
<xs:element name="eventInfo.timeMethod" type="xs:string" minOccurs="0"/>
<!-- eventInfo.occurredStamp.timeMethod [1] -->
<xs:element name="eventInfo.agentTime.utcTime" type="xs:string"
  minOccurs="0"/>
<!-- eventInfo.occurredStamp.agentTime.utcTime [1] -->
<xs:element name="eventInfo.agentTime.subseconds" type="xs:string"
  minOccurs="0"/>
<!-- eventInfo.occurredStamp.agentTime.subseconds [0-1] -->
<xs:element name="eventInfo.agentTimeMethod" type="xs:string" minOccurs="0"/>
<!-- eventInfo.occurredStamp.agentTimeMethod [1] -->
<!-- ***** -->
<!-- END: eventInfo.occurredStamp -->
<!-- ***** -->

<xs:element name="eventInfo.eventHostname" type="xs:string" minOccurs="0"/>
<!-- eventInfo.eventHostname [1] -->
<xs:element name="eventInfo.eventHostAddrType" type="xs:string"
  minOccurs="0"/>
<!-- eth=1, ip4=2, ip6=3 [1] -->
<xs:element name="eventInfo.eventHostAddr" type="xs:string" minOccurs="0"/>
<!-- eventInfo.eventHostAddr [1] -->
<xs:element name="eventInfo.normalizerHostname" type="xs:string"
  minOccurs="0"/>
<!-- eventInfo.normalizerHostname [1] -->
<xs:element name="eventInfo.normalizerAddrType" type="xs:string"
  minOccurs="0"/>
<!-- eth=1, ip4=2, ip6=3 [1] -->
<xs:element name="eventInfo.normalizerAddr" type="xs:string" minOccurs="0"/>
<!-- eventInfo.normalizerAddr [1] -->
<!-- In CER schema, eventInfo.normalizerAddrType is redundant to
  eventInfo.normalizerAddr and not included in this layout. -->

<!-- ***** -->
<!-- BEGIN: eventInfo.corrDescriptor -->
<!-- ***** -->
<xs:element name="eventInfo.corrDescriptor.version" type="xs:string"
  minOccurs="0"/>
```

```
<!-- eventInfo.corrDescriptor.version [1] -->
<!-- eventInfo.corrDescriptor.eventClass is shortened to eventInfo.eventClass
for brevity. -->
<xs:element name="eventInfo.eventClass" type="xs:string" minOccurs="0"/>
<!-- eventInfo.corrDescriptor.eventClass [1] -->
<!-- eventInfo.corrDescriptor.warningLevel is shortened to
eventInfo.warningLevel for brevity. -->
<xs:element name="eventInfo.warningLevel" type="xs:string" minOccurs="0"/>
<!-- eventInfo.corrDescriptor.warningLevel [1] -->
<xs:element name="eventInfo.corrDescriptor.eventCorrDescriptor.version"
type="xs:string" minOccurs="0"/>
<!-- eventInfo.corrDescriptor.eventCorrDescriptor.networkEventCorrDescriptor.
version [1] -->
<!-- eventInfo.corrDescriptor.eventCorrDescriptor.
securityEventCorrDescriptor.version [1] -->
<!-- eventInfo.corrDescriptor.eventCorrDescriptor.systemEventCorrDescriptor.
version [1] -->
<!-- eventInfo.corrDescriptor.eventCorrDescriptor.
applicationEventCorrDescriptor.version [1] -->
<!-- eventInfo.corrDescriptor.eventCorrDescriptor.
environmentalEventCorrDescriptor.version [1] -->
<xs:element name="eventInfo.corrDescriptor.networkEventCorrDescriptor.
netProto" type="xs:string" minOccurs="0"/>
<!-- eventInfo.corrDescriptor.eventCorrDescriptor.networkEventCorrDescriptor.
netProto [1] -->
<xs:element name="eventInfo.corrDescriptor.networkEventCorrDescriptor.
srcAddrType" type="xs:string" minOccurs="0"/>
<!-- eth=1, ip4=2, ip6=3 [1] -->
<xs:element name="eventInfo.corrDescriptor.networkEventCorrDescriptor.
srcAddr" type="xs:string" minOccurs="0"/>
<!-- eventInfo.corrDescriptor.eventCorrDescriptor.networkEventCorrDescriptor.
srcAddr [1] -->
<xs:element name="eventInfo.corrDescriptor.networkEventCorrDescriptor.
dstAddrType" type="xs:string" minOccurs="0"/>
<!-- eth=1, ip4=2, ip6=3 [1] -->
<xs:element name="eventInfo.corrDescriptor.networkEventCorrDescriptor.
dstAddr" type="xs:string" minOccurs="0"/>
<!-- eventInfo.corrDescriptor.eventCorrDescriptor.networkEventCorrDescriptor.
dstAddr [1] -->
<xs:element name="eventInfo.corrDescriptor.networkEventCorrDescriptor.
srcPort" type="xs:string" minOccurs="0"/>
<!-- eventInfo.corrDescriptor.eventCorrDescriptor.networkEventCorrDescriptor.
srcPort [1] -->
<xs:element name="eventInfo.corrDescriptor.networkEventCorrDescriptor.
dstPort" type="xs:string" minOccurs="0"/>
<!-- eventInfo.corrDescriptor.eventCorrDescriptor.networkEventCorrDescriptor.
dstPort [1] -->
<xs:element name="eventInfo.corrDescriptor.networkEventCorrDescriptor.
direction" type="xs:string" minOccurs="0"/>
<!-- eventInfo.corrDescriptor.eventCorrDescriptor.networkEventCorrDescriptor.
direction [1] -->
<!-- ***** -->
<!-- END: eventInfo.corrDescriptor -->
<!-- ***** -->

<xs:element name="eventInfo.userTag" type="keyValue" minOccurs="0"/>
<!-- eventInfo.userTag [0-1] -->
<!-- ***** -->
<!-- END: eventInfo -->
<!-- ***** -->

<!-- ***** -->
<!-- BEGIN: dataPayload -->
<!-- ***** -->
<xs:element name="dataPayload.payloadType" type="payloadType"/>
<!-- raw, cooked, analyzed [1] ***REQUIRED CER Factory element*** -->
<xs:element name="dataPayload.payload" type="xs:string"/>
<!-- raw, cooked, analyzed payload [1] ***REQUIRED CER Factory element*** -->

<!-- ***** -->
```

CARNEGIE MELLON UNIVERSITY
SOFTWARE RELEASE 0.5.2-4, DOCUMENT VERSION 1.20

```
<!-- BEGIN: dataPayload.analyzedDataPayload -->
<!-- ***** -->
<xs:element name="dataPayload.analyzedDataPayload.version" type="xs:string"
  minOccurs="0"/>
<!-- dataPayload.analyzedDataPayload.version [1] -->
<xs:element name="dataPayload.analyzedDataPayload.analysisHostname"
  type="xs:string" minOccurs="0"/>
<!-- dataPayload.analyzedDataPayload.analysisHostname [1] -->
<xs:element name="dataPayload.analyzedDataPayload.analysisHostAddrType"
  type="xs:string" minOccurs="0"/>
<!-- eth=1, ip4=2, ip6=3 [1] -->
<xs:element name="dataPayload.analyzedDataPayload.analysisHostAddr"
  type="xs:string" minOccurs="0"/>
<!-- dataPayload.analyzedDataPayload.analysisHostAddr [1] -->

<!-- ***** -->
<!-- BEGIN: dataPayload.analyzedDataPayload.occurredStamp -->
<!-- ***** -->
<xs:element name="dataPayload.analyzedDataPayload.startTime.utcTime"
  type="xs:string" minOccurs="0"/>
<!-- dataPayload.analyzedDataPayload.occurredStamp.startTime.utcTime [1] -->
<xs:element name="dataPayload.analyzedDataPayload.startTime.subseconds"
  type="xs:string" minOccurs="0"/>
<!-- dataPayload.analyzedDataPayload.occurredStamp.startTime.subseconds [0-1]
-->
<!-- In CER schema, dataPayload.analyzedDataPayload.occurredStamp.stopTime is
[0-1] -->
<xs:element name="dataPayload.analyzedDataPayload.stopTime.utcTime"
  type="xs:string" minOccurs="0"/>
<!-- dataPayload.analyzedDataPayload.occurredStamp.stopTime.utcTime [1] -->
<xs:element name="dataPayload.analyzedDataPayload.stopTime.subseconds"
  type="xs:string" minOccurs="0"/>
<!-- dataPayload.analyzedDataPayload.occurredStamp.stopTime.subseconds [0-1]
-->
<xs:element name="dataPayload.analyzedDataPayload.timeMethod"
  type="xs:string" minOccurs="0"/>
<!-- dataPayload.analyzedDataPayload.occurredStamp.timeMethod [1] -->
<xs:element name="dataPayload.analyzedDataPayload.agentTime.utcTime"
  type="xs:string" minOccurs="0"/>
<!-- dataPayload.analyzedDataPayload.occurredStamp.agentTime.utcTime [1] -->
<xs:element name="dataPayload.analyzedDataPayload.agentTime.subseconds"
  type="xs:string" minOccurs="0"/>
<!-- dataPayload.analyzedDataPayload.occurredStamp.agentTime.subseconds [0-1]
-->
<xs:element name="dataPayload.analyzedDataPayload.agentTimeMethod"
  type="xs:string" minOccurs="0"/>
<!-- dataPayload.analyzedDataPayload.occurredStamp.agentTimeMethod [1] -->
<!-- ***** -->
<!-- END: dataPayload.analyzedDataPayload.occurredStamp -->
<!-- ***** -->

<!-- ***** -->
<!-- BEGIN: dataPayload.analyzedDataPayload.corrDescriptor -->
<!-- ***** -->
<xs:element name="dataPayload.analyzedDataPayload.corrDescriptor.version"
  type="xs:string" minOccurs="0"/>
<!-- dataPayload.analyzedDataPayload.corrDescriptor.version [1] -->
<xs:element name="dataPayload.analyzedDataPayload.corrDescriptor.eventClass"
  type="xs:string" minOccurs="0"/>
<!-- dataPayload.analyzedDataPayload.corrDescriptor.eventClass [1] -->
<xs:element name="dataPayload.analyzedDataPayload.corrDescriptor.
  warningLevel" type="xs:string" minOccurs="0"/>
<!-- dataPayload.analyzedDataPayload.corrDescriptor.warningLevel [1] -->
<xs:element name="dataPayload.analyzedDataPayload.corrDescriptor.
  eventCorrDescriptor.version" type="xs:string" minOccurs="0"/>
<!-- dataPayload.analyzedDataPayload.corrDescriptor.eventCorrDescriptor.
  networkEventCorrDescriptor.version [1] -->
<!-- dataPayload.analyzedDataPayload.corrDescriptor.eventCorrDescriptor.
  securityEventCorrDescriptor.version [1] -->
<!-- dataPayload.analyzedDataPayload.corrDescriptor.eventCorrDescriptor.
```

CARNEGIE MELLON UNIVERSITY
SOFTWARE RELEASE 0.5.2-4, DOCUMENT VERSION 1.20

```
        systemEventCorrDescriptor.version [1] -->
<!-- dataPayload.analyzedDataPayload.corrDescriptor.eventCorrDescriptor.
      applicationEventCorrDescriptor.version [1] -->
<!-- dataPayload.analyzedDataPayload.corrDescriptor.eventCorrDescriptor.
      environmentalEventCorrDescriptor.version [1] -->
<xs:element name="dataPayload.analyzedDataPayload.corrDescriptor.
      networkEventCorrDescriptor.netProto" type="xs:string" minOccurs="0"/>
<!-- dataPayload.analyzedDataPayload.corrDescriptor.eventCorrDescriptor.
      networkEventCorrDescriptor.netProto [1] -->
<xs:element name="dataPayload.analyzedDataPayload.corrDescriptor.
      networkEventCorrDescriptor.srcAddrType" type="xs:string" minOccurs="0"/>
<!-- eth=1, ip4=2, ip6=3 [1] -->
<xs:element name="dataPayload.analyzedDataPayload.corrDescriptor.
      networkEventCorrDescriptor.srcAddr" type="xs:string" minOccurs="0"/>
<!-- dataPayload.analyzedDataPayload.corrDescriptor.eventCorrDescriptor.
      networkEventCorrDescriptor.srcAddr [1] -->
<xs:element name="dataPayload.analyzedDataPayload.corrDescriptor.
      networkEventCorrDescriptor.dstAddrType" type="xs:string" minOccurs="0"/>
<!-- eth=1, ip4=2, ip6=3 [1] -->
<xs:element name="dataPayload.analyzedDataPayload.corrDescriptor.
      networkEventCorrDescriptor.dstAddr" type="xs:string" minOccurs="0"/>
<!-- dataPayload.analyzedDataPayload.corrDescriptor.eventCorrDescriptor.
      networkEventCorrDescriptor.dstAddr [1] -->
<xs:element name="dataPayload.analyzedDataPayload.corrDescriptor.
      networkEventCorrDescriptor.srcPort" type="xs:string" minOccurs="0"/>
<!-- dataPayload.analyzedDataPayload.corrDescriptor.eventCorrDescriptor.
      networkEventCorrDescriptor.srcPort [1] -->
<xs:element name="dataPayload.analyzedDataPayload.corrDescriptor.
      networkEventCorrDescriptor.dstPort" type="xs:string" minOccurs="0"/>
<!-- dataPayload.analyzedDataPayload.corrDescriptor.eventCorrDescriptor.
      networkEventCorrDescriptor.dstPort [1] -->
<xs:element name="dataPayload.analyzedDataPayload.corrDescriptor.
      networkEventCorrDescriptor.direction" type="xs:string" minOccurs="0"/>
<!-- dataPayload.analyzedDataPayload.corrDescriptor.eventCorrDescriptor.
      networkEventCorrDescriptor.direction [1] -->
<!-- ***** -->
<!-- END: dataPayload.analyzedDataPayload.corrDescriptor -->
<!-- ***** -->
<!-- ***** -->
<!-- END: dataPayload.analyzedDataPayload -->
<!-- ***** -->
<!-- ***** -->
<!-- END: dataPayload -->
<!-- ***** -->
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="keyValue">
    <xs:sequence maxOccurs="unbounded">
      <xs:element name="key" type="xs:string"/>
      <!-- Key -->
      <xs:element name="value" type="xs:string"/>
      <!-- Value -->
    </xs:sequence>
  </xs:complexType>

  <xs:simpleType name="payloadType">
    <xs:restriction base="xs:positiveInteger">
      <xs:enumeration value="1"/>
      <!-- Raw -->
      <xs:enumeration value="2"/>
      <!-- Cooked -->
      <xs:enumeration value="3"/>
      <!-- Analyzed -->
    </xs:restriction>
  </xs:simpleType>
</xs:schema>
```

Appendix

E

CER Factory Perl Log Client

Overview

The CER Factory Perl log client is a Perl script that is used to generate CER Factory Templates as logging files are appended to by application daemons. This script is included in the eddycerfactory directory of the CER Factory and EDDY 0.5.2-1 software distributions. The following section describes the functionality of the 'fc' Perl script using UNIX man page conventions.

Man Page

FC(1) fc FC(1)

NAME

fc - The EDDY perl CER Factory client.

SYNOPSIS

fc -n host -t p:port|u:socket -g genmethod -o oid -f logfile [options]

PARAMETERS

options Command-line options.

DESCRIPTION

fc is used to connect to an existing CERFactory. It constantly "tails" any given logfile and uses that to populate and send a template to the CERFactory. Special capabilities are build in for the handling of Shibboleth logs. This may or may not be expanded for other mechanisms in future.

OPTIONS

-h

Display the help menu and exit.

-d

Display the populated template being sent to the factory.

-n HOST

The IP or FQDN of a CER Factory Template Server.

-t p:PORT

-t u:SOCKET

Selects the transport method. Either a port on a remote server or a UNIX domain socket locally.

-g GENMETHOD

A user-defined description tag for the generation method for this specific type of CER. It could be the name of the logging agent or the user's favorite pet.

-o OID

The pre-assigned OID corresponding to this type of CER.
1020 for a generic log file.
1021 for a shibboleth log file.

-f FILE

The log file from which to read

-u TAG

Optional user tags that can be added to the CER template. This command can be repeated as many times as necessary. It may contain any tag.

Appendix

F

Change Errata

Version Updates

The following are the changes to this document from version 1.19 of the EDDY CER Factory Design & Development Guide:

- **Title Page**
Changed software release version from 0.5.2-3 to 0.5.2-4.

The following are the changes to this document from version 1.18 of the EDDY CER Factory Design & Development Guide:

- **Complete Document**
Changed references from a single Template transport connection mechanism to a persistent and multiple Template connection scheme.

The following are the changes to this document from version 1.17 of the EDDY CER Factory Design & Development Guide:

- **Complete Document**
Changed version number to actual point release 0.5.2-2.

Updated copyright notices to 2008 from 2007.

The following are the changes to this document from version 1.16 of the EDDY CER Factory Design & Development Guide:

- **Complete Document**
Added and edited content for the SSL/TLS transport plug-in.

Edited various sentences to clarify their meaning.

Appendix

G

Copyright

Notice & Disclaimer

Copyright © 2004-2008 Carnegie Mellon University. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. The name "Carnegie Mellon University" must not be used to endorse or promote products derived from this software without prior written permission. For permission or any legal details, please contact:

Center for Technology Transfer
Carnegie Mellon University
4615 Forbes Avenue
Pittsburgh, PA 15213-3890
(412) 268-7393, fax: (412) 268-7395

4. Redistributions of any form whatsoever must retain the following acknowledgment: "This product includes software developed at Carnegie Mellon University".

THIS SOFTWARE IS PROVIDED "AS IS" AND ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE EXPRESSLY DECLINED. IN NO EVENT SHALL CARNEGIE MELLON UNIVERSITY BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTES GOOD OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF OR

RELATING TO THE USE (INCLUDING DISTRIBUTION OR PERFORMANCE) OF
THIS SOFTWARE EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.