National Center For Atmospheric Research (NCAR).
High Altitude Observatory (HAO).
Coupling Energetics and Dynamics of Atmospheric Regions (CEDAR).

Summary.

The CEDAR database is located across binary files (from now we will mention those files as cbf files), not a RDBMS like Oracle, just files. Any program trying to extract data from those files needs a set of functions to get that job done. If those functions do notexist a programmer must create them with implications in time, resources, etc.

What is a CEDAR API? It is a set of C++ classes that allow you to transparent connect to cbf files, extract chunks of meaningfuldata, subset the extracted data, validate data based on some criteria, etc.

The CEDAR API on the large scale consists of an interface and an internal implementation. For programmers interested in just using the interface, documentation in CHAPTER 2 is all they need. For a full detailed description of how this API was created refer to CHAPTER 3 of this document.

Is is very important to make clear that a cbf file's physical record structure for historical reasons is based on COS (Cray Operative System) blocking and that the access style of the data is based on tapes. These elements add extra complexity to the API because the library must extract the data as it is packed inside the cbf files dealing with Cray Blocking and sequential access.

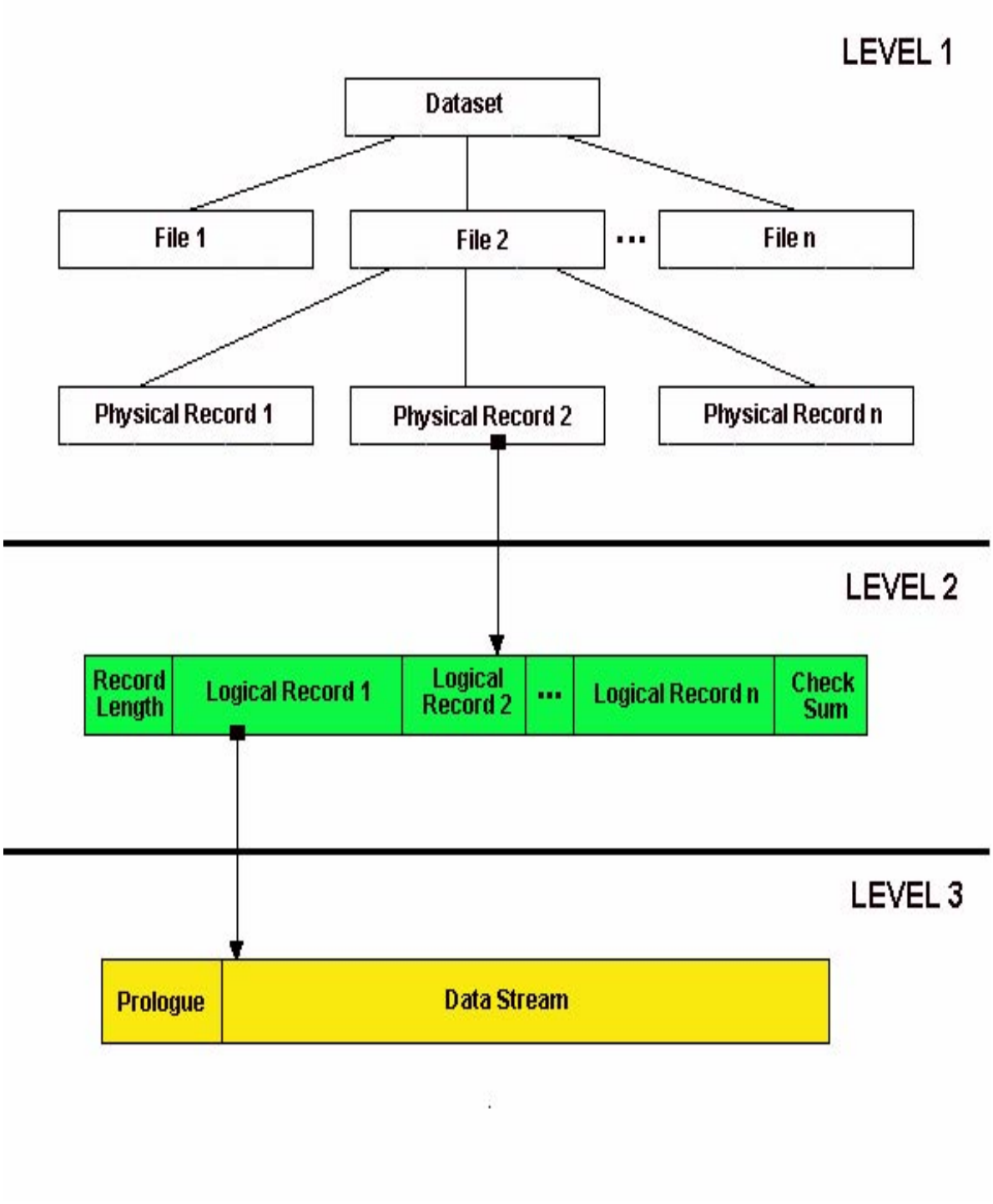
As of this release of the API, the C++ classes do not have write capabilities, so the connection to cbf files is read-only.

An overview of the problem at hand.

The COS blocking system and the CEDAR Format

First we must understand the general idea of the COS blocking system and how it packs pieces of data called Physical Records.

To do that lets analyze the next image:


LEVEL 1
Dataset
File 1
File 2
File n
Physical Record 1
Physical Record 2
Physical Record n
LEVEL 2
Record Length
Logical Record 1
Logical Record 2
Logical Record n
Check Sum
LEVEL 3
Prologue
Data Stream

As you can see there are three levels of packeting.

In LEVEL 1 COS blocking is used to put together Physical Records in files and those files into a dataset. The word file here is very confusing because it refers to a chunk of data between two EOF control words and not to a file as we know it in terms of the Unix File System (UFS).

In LEVEL 2 we can see that the previously mentioned Physical Records are packages themselves (this is the CEDAR Format) as they contain control words and logical records.

In LEVEL 3 the logical records is the information contained in the data area which its corresponding prologue.

For these concepts, lets introduce the basic terminology:

- cbf file: this is the dataset and matches the idea of file in UFS. For the remainder of this document a cbf files is a file in which the physical records are put together using COS blocking.

- group: To avoid the confusion with the word file we use the word group. A group contains multiple records related within a dataset and it is equal to the word file in terms of COS Blocking.

- physical Record: It is the block packing as defined in LEVEL 2.

- logical Record: The information packed as prologue and data areas contained inside the Physical Record as defined in LEVEL 3.

If you need detailed information about COS blocking or the CEDAR Database Format read the COS reference manual or the document titled "CEDAR DATA BASE FORMAT" from May 1990.

Ideas for classes based on the concepts previous explained

The first concept with which we are dealing it is the concept of a cbf files. We call this class:

```
class CedarFile
{
...
};
```

An object of the class CedarFile:

- Establishes file connection to one and only one cbf file

- Retrieves independent physical records from the cbf file in sequence

- Retrieves the physical records in groups indicated by EOF Cray control words.

- Breaks connection to the cbf file.

For the concept of physical record we will create:

```
class CedarBlock
{
...
};
```

An object of the class CedarBlock:

- Holds in memory a complete physical Record extracted by an object CedarFile

- Retrieves independent logical records from the Physical Record in sequence

For the concept of logical Record we have the class:

```
class CedarLogicalRecord
{
...
};
```

An object of the class CedarLogicalRecord:

- Represents the abstract idea of a logical Record (as virtual functions) for later specialization by derived classes

- Using run time polymorphism allows you to control a complete derived logical record object extracted from within a CedarBlock object.

The word specialization was mentioned just in the last paragraph. Why is that? We said that Logical records is an abstract idea that puts together as one thing the different type of records for data and metadata. Those are: Header Records, Catalog Records and Data records using the CEDAR database definitions. By deriving from the abstract class CedarLogicalRecord we can create classes to encapsulate the attributes and functionality of the three types of data and metadata records. Those classes are:

```
class CedarDataRecord : public CedarLogicalRecord
{
...
}

class CedarCatalogRecord : public CedarLogicalRecord
{
...
}
```

```
class CedarHeaderRecord : public CedarLogicalRecord
{
...
}
```

The CEDAR database format uses Universal Time to indicated the time of a logical record. A UT consist of four positive integers to indicate year, month-day, hour-minute and centisecond. Relational operations as well as data encapsulation for dates is managed by the class:

```
class CedarDate
{
...
};
```

Finally it is important to mention a class that has not been fully implemented yet but it has been designed to match the concept of a group. This class is CedarGroup and has the following high level declaration:

```
class CedarGroup
{
...
};
```

An object of the class CedarGroup is designed to retrieve a set of CedarBlock that are related together by two EOF control words as they represent an experiment . This class implementation is pending in a future release.