

RCP Names

Marc Paterno
CD/CPD/APS, FNAL

October 9, 2002

Abstract

This document is a brief explanation of some of the rules concerning RCP names, and how they are handled by the database.

1 The Purpose of RCP Names

Each parameter set (a collection of parameters, which gets presented to a C++ program in the form of an instance of the class *RCP*) carries a unique identifier, its RCPID (an instance of the class *RCPID*). While these identifiers are useful in a program, they aren't the most mnemonic method of identifying parameter sets. For this reason, there also exist *RCP names*. While each parameter set has only one RCPID, and each RCPID corresponds to only one parameter set, RCP names form a many-to-one map with parameter sets. That is, each RCP name is only associated with one parameter set, but each parameter set may be known by many RCP names.

2 What's in a Name

An RCP name consists of several parts:

- a *database name*, which is the name of the database that contains this parameter set,
- a *package name*, which is the name of the CVS package (or “fake package”, for parameter sets not in the “official” database) with which this parameter set is associated,
- an *object name*, which is the name of this parameter set within the package, and
- a *version tag*, which is the string which identifies the version of the object with this name. In the “official” database, this version tag will be the same as the version tag of the library release with which this parameter

set is associated. In a “personal” database, this version tag will be in the form of a timestamp.¹

3 Rules Concerning Names

When one uses the function `edm::RCPManager::extract(std::string& pkg, std::string& obj)` to obtain a parameter set, this actually specifies the *package name* and *object name* parts of the full RCP name. The search that is conducted will eventually result in the *database name* and *version tag* parts of the name being completed by the database in which the parameter set is found, or by the database to which it is added. When the search for a local RCP script finds one, it results in a partial RCP name being formed, one which has only the package name and object name portions completed. This name will “match” any RCP name that has the same package name and object name.

One of the differences between the “official” database and the “personal” databases is that the “official” database is under a strong restriction to which the “personal” databases are not subject.

The “official” database does not issue timestamp-style version tags; when the release managers update this database, new parameter sets are added with names containing the version tag given by the name of the build (*e.g.* t00.65.00). Parameter sets already known to the database do not need to be added, but they are given an additional name, one which contains the new version tag.

When parameter sets are extracted from the “official” database (using `edm::RCPManager::extract(std::string& pkg, std::string& obj)`), only those parameter sets matching the release currently in use (established by the command `setup DORunII <version>`) will be extracted. This is to ensure that a program does not mix parameter sets from different releases.

In contrast, when a new parameter set is added to a “personal” database, it is assigned a version tag that is created by the database itself, in the form of a timestamp (note that the timestamp is created using the current Universal Time). It is rarely necessary to add a second RCP name which differs only in version tag to a “personal” database, since a “personal” database is not under the restriction of returning only parameter sets with a specific version tag. Instead, a “personal database” will return the *most recent* parameter set with the given name, where most recent means the parameter set with the most recently added name.

4 Where the Rules Cause Trouble

The document **RCP Rules for DØ** lists the set of rules which dictate the behavior of the function `edm::RCPManager::extract(std::string& pkg, std::string&`

¹This is actually a slight simplification; the behavior of these databases is controlled by the file pointed to by the environment variable `RCP_DB_NAMES_FILE`. The behavior described here is the behavior established by default at DØ.

obj). In this document, case 2.1.1.1 is the most difficult to handle. This case states the behavior required for the system under the following circumstances.

- A script is found; the an incomplete parameter set specified by this script is formed. The parameter set is incomplete because it is lacking an RCPID, which must be supplied by a database.
- A parameter set that matches² this new parameter set is found in one of the readonly databases.
- None of the RCP names associated with the parameter set found in the database match the partial RCP name formed from the arguments to the `extract` function.

Under these circumstances, the rules require that an exception be thrown, indicating that the parameter set specified by the script can not be associated with the given name. To understand the reasoning that lead to this rule, let us consider the alternatives.

4.1 Alternative 1

Given the case described, the requirement could have been that we add a new RCP name to this parameter set in the readonly database. But in this case, the database is not really readonly; we are modifying it with such a call.

Because the database could be modified by such a call, it would not be feasible to distribute this database in “FileSystemDB” form (or in any other form) to remote institutions. Since every user can modify the “official” database, it would become prohibitively difficult to assure that a program run at a remote institution, using a copy of the “official” database distributed along with a given library release, would give the same result as a program run at Fermilab, where there resides the original copy of the “official” database, which would be continually under modification by this case.

This was considered unacceptable, and so this choice was rejected.

This problem would be eliminated if all programs using the “official” database were always required to contact a single central database server. Not all remote institutions have sufficiently good network connections to make this feasible. It was considered unacceptable to require a network connect to a single central database in order to make use of the “official” RCP database.

²Note that two parameter sets “match” when the pairs of names and values in one are equal to the pairs of names and values in the other. The RCP names, partial or otherwise, associated with the parameter sets are irrelevant in determining the match; only the parameters and their names are considered. Note also that the order in which parameters are defined in an RCP script makes no difference to the collection of names and values in the RCP object that is created, and so that order also has no bearing on whether two parameter sets match.

4.2 Alternative 2

The requirement could have been to add the new RCP name to the currently defined writable database. However, there is still a requirement that the parameter set (regardless of the associated RCP name) have a unique RCPID, so the RCPID contained in the *RCP* object returned to the user must be that of the matching parameter set in the “official” database. This would mean that various “personal” databases would then contain parameter sets which carry an RCPID which says that parameter set comes from the “official” database.

Consider what then happens when two users each run a program that requests an RCP with the code given in Figure 1.

```
edm::RCPManager pman = edm::RCPManager::instance();  
edm::RCP r = pman->extract("packageA", "object1");
```

Figure 1: A simple code snippet that leads to confusion.

Suppose that user **A** runs his program while using the “official” database and his “personal” database, and runs into this case. His program will find the RCP *r*, which will be carrying an RCPID issued by the “official” database. It will look to user **A** just like this parameter set was found in the official database, because in fact it was.

Suppose also that user **B** runs his program (containing the same code) while using the “official” database, but does not have the local script “packageA/rcp/object1.rcp”.³ His call to `extract` will fail, throwing an exception which carries a message stating that no parameter set with the given name could be found.

User **A** and user **B** might have some very confusing times ahead, trying to figure out how they seem to see different things in the same “official” database.

This behavior was considered unacceptably confusing, and so it was rejected.

4.3 Alternative 3

The final alternative is to have the system throw an exception which carries enough information to allow the user to determine how to fix the problem – specifically, what name he should use to extract the parameter set which already has the values that the script indicates he wants.

This behavior was considered to be annoying, but seemed to be the least of the three evils. Accordingly, this is the behavior specified for the system.

5 Final Notes

The behavior of the RCP system in this circumstance has been reviewed several times, most recently during the “DØ Software Infrastructure Week”, in

³It makes no difference to this argument whether or not user **B** is using a writable database.

May 2000. Each time, the reviewing group has decided that the choice of “alternative 3” above is the appropriate choice. Requests for change in this behavior should *not* go to the authors of the RCP package, but rather to the leaders of the DØ software infrastructure group.

The exception thrown by the system in case specified in Section 4 is an instance of the class `edm::XRCPcantAssignNewName`, which inherits from the class `edm::XRCP` (the base class for most exceptions thrown by the RCP system)⁴, which in turn inherits from `ZMexception`, the base class of the ZOOM group’s exception classes.

It is strongly recommended that all use of the RCP system (including calls to gain access to the sole instance of *RCPManager*, calls to extract RCP objects, and calls to get parameters from RCP objects) be wrapped within a `try` block, which is followed by a `catch` block which accepts `edm::XRCP` exceptions, and prints the message all such objects contain to the appropriate place. These exceptions have been designed to carry enough information to help users determine the cause of failure in their programs. Suggestions for improvement in these error messages should go to the czar of the RCP library (paterno@fnal.gov).

Note that the DØ framework automatically catches any exceptions thrown by code in any framework package, and will then print the appropriate error message. Of course, it is impossible for the framework itself to know how to continue in such a case, so after catching the exception, the framework prints the error message and kills the program.

Non-framework executables must provide the `try/catch` blocks themselves.

⁴Some exceptions thrown by the RCP parser subsystem are defined in the header *XParser.hpp* and inherit from `std::exception`. In a future release, these exceptions will also be migrated to inherit from `edm::XRCP`.