

---

# Rivendell Notification Protocol

Fred Gleason

## Table of Contents

Overview .....	1
Notification Messages .....	1
Carts .....	2
Logs .....	2
Pypad Instances .....	2
Dropbox Instances .....	2
RDCatch Event Instances .....	3
RSS Feed Items .....	3
RSS Feeds .....	3
SoundPanel Buttons [ <b>rdairplay</b> (1)] .....	3
Extended SoundPanel Buttons [ <b>rdpanel</b> (1)] .....	4
RDCatch Messages .....	4
Deck Event Processed Operation .....	4
Deck Status Operation .....	4
Reload Decks Operation .....	5
Send Meter Levels Operation .....	6
Set Input Monitor .....	6
Set Input Monitor Response .....	6
Stop Deck Operation .....	7

## Overview

This document defines the IP protocol used by Rivendell for real-time communication between different modules and/or hosts. Messages are sent by means of multicast UDP packets to port 20539. The IPv4 multicast group address used is defined in the SYSTEM.NOTIFICATION\_ADDRESS database field.

Update messages are textual, with individual fields delimited by the ASCII SPACE character. They are formatted as follows:

*keyword arg1 arg2 [ . . . ]*

where *keyword* defines the class of message. The following classes are defined:

Notifications	Signals a change to a Rivendell database object.
Catch Events	RDCatch state changes.

## Notification Messages

Notification messages use the following format:

NOTIFY *type action id*

*type*                      The database object type to which the message pertains.

*action*                    The action being reported. The following actions are defined:

ADD	The referenced object has just been added to the database.
DELETE	The referenced object has just been deleted from the database.
MODIFY	The reference object has just been modified in the database.

*id* Unique id of the object.

The following database object types are defined:

## Carts

**Table 1. Cart Fields**

Field	Value
Database Field	CART.NUMBER
Type	CART
Id Data Type	Unsigned Integer
RDNotification::Type Value	RDNotification::CartType [1]

## Logs

**Table 2. Log Fields**

Field	Value
Database Field	LOGS.NAME
Type	LOG
Id Data Type	String
RDNotification::Type Value	RDNotification::LogType [2]

## Pypad Instances

**Table 3. Pypad Instance Fields**

Field	Value
Database Field	PYPAD_INSTANCES.ID
Type	PYPAD
Id Data Type	Integer
RDNotification::Type Value	RDNotification::PypadType [3]

## Dropbox Instances

**Table 4. Dropbox Instance Fields**

Field	Value
-------	-------

Database Field	STATIONS.NAME
Type	DROPBOX
Id Data Type	String
RDNotification::Type Value	RDNotification::DropboxType [4]

### Note

Dropbox Add/Modify/Delete actions operate at a 'per-host' granularity --i.e. *any* notification will cause *all* dropboxes on the target host to be killed and then restarted.

## RDCatch Event Instances

**Table 5. RDCatch Event Fields**

Field	Value
Database Field	RECORDINGS.ID
Type	CATCH_EVENT
Id Data Type	Integer
RDNotification::Type Value	RDNotification::CatchEventType [5]

## RSS Feed Items

**Table 6. RSS Feed Item Fields**

Field	Value
Database Field	PODCASTS.ID
Type	FEED_ITEM
Id Data Type	Unsigned Integer
RDNotification::Type Value	RDNotification::FeedItemType [6]

## RSS Feeds

**Table 7. RSS Feed Fields**

Field	Value
Database Field	FEEDS.KEY_NAME
Type	FEED
Id Data Type	String
RDNotification::Type Value	RDNotification::FeedType [7]

## SoundPanel Buttons [rdairplay(1)]

**Table 8. SoundPanel Button Fields**

Field	Value
Database Field	PANELS.ID

Type	PANEL_BUTTON
Id Data Type	Integer
RDNotification::Type Value	RDNotification::PanelButtonType [8]

## Extended SoundPanel Buttons [rdpanel(1)]

**Table 9. SoundPanel Button Fields**

Field	Value
Database Field	EXTENDED_PANELS.ID
Type	EXTENDED_PANEL_BUTTON
Id Data Type	Integer
RDNotification::Type Value	RDNotification::ExtendedPanelButtonType [9]

## RDCatch Messages

RDCatch messages use the following format:

*CATCH orig-hostname operation arg1 [...]*

*orig-hostname*                      The name of the host originating the message. From STATIONS.NAME.

*operation*                          The RDCatch operation. See the list below.

## Deck Event Processed Operation

Emitted by **rdcatchd**(8) every time a 'Cut Event' ['CE'] RML is processed.

**Table 10. RDCatch Event Fields**

Field	Offset	Value
Keyword	0	CATCH
Originating Hostname	1	String, from STATIONS.NAME
Operation	2	Integer. 1 [RDCatchEvent::DeckEventProcessedOp]
Deck Channel	3	Integer. Record decks have values in the range 1-127, while play decks have values in the range 128-254.
Event Number	4	Integer. The value of CUT_EVENTS.NUMBER.

## Deck Status Operation

Emitted by **rdcatch**(1) at startup to request the current status of all decks.

**Table 11. RDCatch Event Request Fields**

Field	Offset	Value
-------	--------	-------

Keyword	0	CATCH
Originating Hostname	1	String, from STATIONS.NAME
Operation	2	Integer. 2 [RDCatchEvent::DeckStatusQueryOp]

Emitted by **rdcatchd**(8) whenever the status of a deck changes, or as a set in response to a request. (See Table 11, “RDCatch Event Request Fields”).

**Table 12. RDCatch Event Response Fields**

Field	Offset	Value
Keyword	0	CATCH
Originating Hostname	1	String, from STATIONS.NAME
Operation	2	Integer. 3 [RDCatchEvent::DeckStatusResponseOp]
Deck Channel	3	Integer. Record decks have values in the range 1-127, while play decks have values in the range 128-254.
Status	4	Integer. Current status of the specified deck. See the section called “Deck Status Codes” for code point values.
Event ID	5	Unsigned Integer, from RECORDINGS.ID or 0 if deck is inactive.
Cart Number	6	Unsigned Integer, from CART.NUMBER or 0 if deck status is not 3 [Active].
Cut Number	7	Integer, from cut part of CUTS.CUTNAME or 0 if deck status is not 3 [Active].

## Deck Status Codes

**Table 13. Deck Status Codes**

Code	Meaning	RDDeck::Status Value
0	Offline	RDDeck::Offline
1	Idle	RDDeck::Idle
2	Ready	RDDeck::Ready
3	Active (playing or recording)	RDDeck::Recording
4	Waiting (for a GPI)	RDDeck::Waiting

## Reload Decks Operation

Emitted by **rdadmin**(1) to trigger a reload of the record and play-out deck parameters.

**Table 14. RDCatch Reload Decks Fields**

Field	Offset	Value
-------	--------	-------

Keyword	0	CATCH
Originating Hostname	1	String, from STATIONS.NAME
Operation	2	Integer. 7 [RDCatchEvent::ReloadDecksOp]
Target Hostname	3	String, from STATIONS.NAME

## Send Meter Levels Operation

Emitted by **rdcatchd**(8) to update audio meter levels for active record and play-out decks.

**Table 15. RDCatch Send Meter Levels Fields**

Field	Offset	Value
Keyword	0	CATCH
Originating Hostname	1	String, from STATIONS.NAME
Operation	2	Integer. 8 [RDCatchEvent::SendMeterLevelsOp]
Meter Entry	3	String, <i>chan:left-lvl:right-lvl</i>

## Set Input Monitor

Emitted by **rdcatch**(1) to turn a deck input monitor on or off.

**Table 16. RDCatch Set Input Monitor Fields**

Field	Offset	Value
Keyword	0	CATCH
Originating Hostname	1	String, from STATIONS.NAME
Operation	2	Integer. 5 [RDCatchEvent::SetInputMonitorOp]
Target Hostname	3	String, from STATIONS.NAME
Deck Channel	4	Integer. Record decks have values in the range 1-127, while play decks have values in the range 128-254.
State	5	Boolean. 0=False, 1=True.

## Set Input Monitor Response

Emitted by **rdcatchd**(8) to signal change of a deck input monitor.

**Table 17. RDCatch Set Input Monitor Fields**

Field	Offset	Value
Keyword	0	CATCH
Originating Hostname	1	String, from STATIONS.NAME
Operation	2	Integer. 6 [RDCatchEvent::SetInputMonitorResponseOp]

Deck Channel	3	Integer. Record decks have values in the range 1-127, while play decks have values in the range 128-254.
State	4	Boolean. 0=False, 1=True.

## Stop Deck Operation

Emitted by **rdcatch**(1) to abort a Record or Play-out event.

**Table 18. RDCatch Stop Deck Fields**

Field	Offset	Value
Keyword	0	CATCH
Originating Hostname	1	String, from STATIONS.NAME
Operation	2	Integer. 4 [RDCatchEvent::StopDeckOp]
Target Hostname	3	String, from STATIONS.NAME
Deck Channel	4	Integer. Record decks have values in the range 1-127, while play decks have values in the range 128-254.