Centova Cast

Internals Reference Manual

Published May 04, 2015

Centova Technologies Inc.
www.centova.com

Contents

1 Introduction **10**

2 API Reference **11**

2.1 API Types 12

2.1.1 JSON API 12

2.1.2 XML API 12

2.1.3 Session API 12

2.1.4 Commandline 12

2.2 Class Types 13

2.2.1 System Class Authentication 13

2.2.2 Server Class Authentication 13

2.3 JSON API 14

2.3.1 JSON API Request Structure 14

2.3.2 JSON API Response Structure 14

2.4 XML API 16

2.4.1 XML Request Packet Structure 16

2.4.2 XML Response Packet Structure 17

2.5 Session API 18

2.5.1 Session API Request Structure 18

2.5.2 Session API Response Structure 18

2.5.3 JSON and XML APIs - Asynchronous API Responses 19

2.5.4 Submitting a Request for Asynchronous Processing 19

2.5.5 Handling the Response Callback 21

2.5.6 JSON API Response Callback Example 21

2.5.7 XML Response Callback Example 22

2.5.8 API Errors 23

2.5.9 Postback Failures . . . 23
2.6 Server Class Method Reference . . . 24
2.6.1 Get Account Settings . . . 24
2.6.2 Get Stream Status . . . 24
2.6.3 Copy File . . . 27
2.6.4 Retrieve Logs . . . 28
2.6.5 Get Song History . . . 28
2.6.6 Get Listener List . . . 29
2.6.7 Get Account State . . . 31
2.6.8 Validate Account Credentials . . . 31
2.6.9 Start Stream . . . 32
2.6.10 Reload Stream . . . 33
2.6.11 Restart Stream . . . 33
2.6.12 Stop Stream . . . 34
2.6.13 Activate/Deactivate autoDJ . . . 34
2.6.14 Update Media Library . . . 35
2.6.15 Silently Update Media Library . . . 36
2.6.16 Manage Playlists . . . 36
2.6.17 Advance to Next Song . . . 40
2.6.18 Refresh Disk Usage . . . 40
2.6.19 Reconfigure Account . . . 41
2.6.20 When updating client accounts . . . 41
2.6.21 When updating reseller accounts . . . 44
2.6.22 Common settings that can be provided for either client or reseller accounts . . . 45
2.6.23 Manage DJ accounts . . . 46
2.6.24 When creating or updating a DJ account. . . . 47
2.6.25 When deleting a DJ account . . . 48
2.6.26 When retrieving a DJ account . . . 48
2.6.27 When listing DJ accounts . . . 49
2.6.28 When retrieving a DJ account . . . 49
2.6.29 For all other actions . . . 49
2.6.30 Download Log Archive . . . 49
2.7 System Class Method Reference . . . 51
2.7.1 Sanity Check . . . 51

2.7.2 Image Daemon Interface . . . 51
2.7.3 Rename Account . . . 52
2.7.4 Silently Update Media Library . . . 52
2.7.5 Version and Host Information . . . 53
2.7.6 Provision Account . . . 54
2.7.7 When creating client accounts . . . 54
2.7.8 When creating reseller accounts . . . 58
2.7.9 Common settings that can be provided for either client or reseller accounts . . . 59
2.7.10 Remove Account . . . 61
2.7.11 Reparent Account . . . 61
2.7.12 Set Account Status . . . 62
2.7.13 Check Stream Outages . . . 63
2.7.14 Get Account State . . . 63
2.7.15 Get Resource Utilization . . . 64
2.7.16 Perform Batch Operations . . . 65
2.7.17 Account List . . . 66
2.7.18 Process Logs . . . 67
2.7.19 Database Import/Export . . . 67
2.7.20 Account Backup . . . 68
2.7.21 Account Restore . . . 69
2.7.22 Account Software Change . . . 70
2.7.23 Hosting Server List . . . 70
2.7.24 Region List . . . 72

3 System Accounts 73

3.1 Under Linux . . . 74
3.2 Under Windows . . . 75

4 Advanced Configurations 76

4.1 Dual v3/v2 Deployment . . . 77
4.1.1 Without suPHP . . . 77
4.1.2 With suPHP . . . 77
4.2 Using centovacast.conf . . . 78
4.2.1 Database connection options . . . 78

4.2.2 Locale configuration . . . 78
4.2.3 Feature Configuration . . . 78
4.2.4 Date and Time . . . 79
4.2.5 Track Information Formatting . . . 79
4.2.6 Event Scripts . . . 80
4.2.7 Log Processing . . . 80
4.2.8 Privileges and Policy Enforcement . . . 81
4.2.9 Directories and Path Traversal . . . 82
4.2.10 Streaming Server Hostnames . . . 82
4.2.11 Media Library . . . 83
4.2.12 AutoDJ . . . 84
4.2.13 Optimization . . . 85
4.2.14 Process launching & monitoring . . . 85
4.2.15 Log Processing . . . 86
4.2.16 SMTP options . . . 86
4.2.17 Daemon connectivity . . . 87
4.2.18 Compatibility features . . . 87
4.2.19 Application-assigned values . . . 87

5 Command-line Tools 89

5.1 Controlling Centova Cast . . . 90
5.1.1 Init Script . . . 90
5.1.2 Advanced Process Control . . . 90
5.2 Diagnostic Report Generator . . . 91
5.3 Fixing Problems . . . 92
5.3.1 Permissions Problems . . . 92
5.4 Centova Cast Management Utility . . . 93
5.4.1 Management Utility Invocation . . . 93
5.4.2 Output Formats . . . 93
5.5 Reinstalling Centova Cast . . . 98
5.6 Uninstalling Centova Cast . . . 99
5.7 Update Utility . . . 100
5.7.1 Basic Invocation . . . 100
5.7.2 Updating Individual Components . . . 100

5.7.3 Forcing an Update . . . 100
5.7.4 Adding New Components . . . 101
5.7.5 Performing Custom Actions on Update . . . 101
5.8 Init Script . . . 102
5.9 Menu Customizations . . . 103
5.9.1 Menu Definitions . . . 103
5.9.2 Menu Sections . . . 103
5.9.3 Menu Items . . . 104
5.9.4 Conditions . . . 104
5.9.5 DJ Permissions . . . 105
5.10 Wrapping Server Applications . . . 106
5.10.1 A Sample Wrapper Script . . . 106
5.10.2 Implementing the Wrapper Script . . . 108
5.10.3 Practical Example: Controlling Apache . . . 109
5.10.4 Practical Example: Indirectly Controlling DNAS2 . . . 111
5.10.5 Further development . . . 113
5.11 Event Script Reference . . . 114
5.11.1 Event: playlist_advanced . . . 114
5.11.2 Event: pre-create-reseller . . . 115
5.11.3 Event: pre-create-account . . . 116
5.11.4 Event: post-create-reseller . . . 118
5.11.5 Event: post-create-account . . . 119
5.11.6 Event: pre-terminate-account . . . 120
5.11.7 Event: pre-terminate-reseller . . . 121
5.11.8 Event: post-terminate-account . . . 122
5.11.9 Event: post-terminate-reseller . . . 122
5.11.10 Event: pre-reparent-account . . . 123
5.11.11 Event: post-reparent-account . . . 124
5.11.12 Event: pre-account-status . . . 124
5.11.13 Event: post-account-status . . . 125
5.11.14 Event: pre-start-server . . . 126
5.11.15 Event: post-start-server . . . 126
5.11.16 Event: pre-start-source . . . 127
5.11.17 Event: pre-start-app . . . 128

5.11.18 Event: post-start-source . . . 129
5.11.19 Event: post-start-app . . . 129
5.11.20 Event: pre-reload . . . 130
5.11.21 Event: post-reload . . . 130
5.11.22 Event: pre-stop-source . . . 131
5.11.23 Event: pre-stop-app . . . 132
5.11.24 Event: post-stop-source . . . 133
5.11.25 Event: post-stop-app . . . 133
5.11.26 Event: pre-stop-server . . . 134
5.11.27 Event: post-stop-server . . . 134
5.11.28 Event: pre-reindex . . . 135
5.11.29 Event: post-reindex . . . 136
5.11.30 Event: pre-process-logs . . . 136
5.11.31 Event: post-process-logs . . . 137
5.11.32 Event: pre-rotate-logs . . . 138
5.11.33 Event: post-rotate-logs . . . 138
5.11.34 Event: pre-update-reseller . . . 139
5.11.35 Event: pre-update-account . . . 140
5.11.36 Event: post-update-reseller . . . 142
5.11.37 Event: post-update-account . . . 143
5.11.38 Event: send-email . . . 144
5.11.39 Event: pre-rename-account . . . 145
5.11.40 Event: pre-rename-reseller . . . 146
5.11.41 Event: post-rename-account . . . 147
5.11.42 Event: post-rename-reseller . . . 148
5.11.43 Event: server-outage-restarted . . . 148
5.11.44 Event: server-outage-restart-failed . . . 149
5.11.45 Event: source-outage-restarted . . . 149
5.11.46 Event: source-outage-restart-failed . . . 150
5.11.47 Event: app-outage-restarted . . . 151
5.11.48 Event: app-outage-restart-failed . . . 151
5.11.49 Event: bitrate-exceeded . . . 152
5.12 Event Notification Scripts . . . 153
5.12.1 Disclaimer . . . 153

5.12.2 Getting Started . . . 153
5.12.3 Important Notes . . . 153
5.13 Event Script Structure . . . 155
5.13.1 Overview . . . 155
5.13.2 Filename and Location . . . 155
5.13.3 Implementation . . . 155

6 Files and Paths 156

6.1 Log Files . . . 157
6.2 Client Data (Linux) . . . 158
6.3 Client Data (Windows) . . . 159
6.4 Cron Job . . . 160
6.5 Configuration . . . 161
6.5.1 Web Interface . . . 161
6.5.2 FTP Server . . . 162
6.5.3 Control Daemon (Linux) . . . 162
6.5.4 Control Daemon (Windows) . . . 162
6.5.5 Image Daemon . . . 162
6.5.6 Comet Daemon . . . 163
6.5.7 General Configuration . . . 163
6.5.8 Other Files . . . 163
6.6 Account Files and Paths . . . 164
6.6.1 Log Files . . . 164
6.6.2 Configuration . . . 164
6.7 Core Files and Paths . . . 166
6.7.1 Log Files . . . 166
6.7.2 Client Data (Linux) . . . 166
6.7.3 Client Data (Windows) . . . 167
6.7.4 Cron Job . . . 167
6.7.5 Configuration . . . 167
6.7.6 nextsong.log File Format . . . 169
6.8 Event Script Reference . . . 171
6.8.1 Event: playlist_advanced . . . 171
6.8.2 Event: pre-create-reseller . . . 172

6.8.3 Event: pre-create-account . . . 174
6.8.4 Event: post-create-reseller . . . 176
6.8.5 Event: post-create-account . . . 177
6.8.6 Event: pre-terminate-account . . . 179
6.8.7 Event: pre-terminate-reseller . . . 180
6.8.8 Event: post-terminate-account . . . 181
6.8.9 Event: post-terminate-reseller . . . 182
6.8.10 Event: pre-reparent-account . . . 183
6.8.11 Event: post-reparent-account . . . 185
6.8.12 Event: pre-account-status . . . 186
6.8.13 Event: post-account-status . . . 187
6.8.14 Event: pre-start-server . . . 188
6.8.15 Event: post-start-server . . . 189
6.8.16 Event: pre-start-source . . . 190
6.8.17 Event: pre-start-app . . . 191
6.8.18 Event: post-start-source . . . 192
6.8.19 Event: post-start-app . . . 193
6.8.20 Event: pre-reload . . . 195
6.8.21 Event: post-reload . . . 196
6.8.22 Event: pre-stop-source . . . 197
6.8.23 Event: pre-stop-app . . . 198
6.8.24 Event: post-stop-source . . . 199
6.8.25 Event: post-stop-app . . . 200
6.8.26 Event: pre-stop-server . . . 201
6.8.27 Event: post-stop-server . . . 202
6.8.28 Event: pre-reindex . . . 203
6.8.29 Event: post-reindex . . . 204
6.8.30 Event: pre-process-logs . . . 205
6.8.31 Event: post-process-logs . . . 206
6.8.32 Event: pre-rotate-logs . . . 207
6.8.33 Event: post-rotate-logs . . . 209
6.8.34 Event: pre-update-reseller . . . 210
6.8.35 Event: pre-update-account . . . 211
6.8.36 Event: post-update-reseller . . . 213

6.8.37 Event: post-update-account . . . 215
6.8.38 Event: send-email . . . 217
6.8.39 Event: pre-rename-account . . . 218
6.8.40 Event: pre-rename-reseller . . . 219
6.8.41 Event: post-rename-account . . . 221
6.8.42 Event: post-rename-reseller . . . 222
6.8.43 Event: server-outage-restarted . . . 223
6.8.44 Event: server-outage-restart-failed . . . 224
6.8.45 Event: source-outage-restarted . . . 225
6.8.46 Event: source-outage-restart-failed . . . 226
6.8.47 Event: app-outage-restarted . . . 227
6.8.48 Event: app-outage-restart-failed . . . 228
6.8.49 Event: bitrate-exceeded . . . 229
6.9 Example Plugin . . . 231
6.10 Plugins API . . . 232
6.10.1 Introduction . . . 232
6.10.2 Overview . . . 232
6.10.3 Environment . . . 232
6.11 Plugin Structure . . . 233
6.11.1 Directory Layout . . . 233
6.11.2 The hooks.php File . . . 233
6.11.3 The install_hooks Method . . . 233
6.11.4 Return Values . . . 234

Chapter 1

Introduction

This is the internals reference manual for Centova Cast, the leading Internet radio stream hosting solution.

This document provides systems administrators and integrators with information about the internals of Centova Cast, including file locations and purposes, commandline tools, automation and integration details, and other information that may be useful to advanced users.

Chapter 2

API Reference

To facilitate interaction between Centova Cast and third-party software applications, Centova Cast provides a simple automation API. This manual documents the calling conventions and functionality provided the API.

This document is provided as a courtesy to Centova Cast clients who wish to integrate Centova Cast with their existing business operations. It is intended for qualified software developers and assumes that the reader has some level of experience with HTTP, XML, and software application development in general. We regret that we cannot provide support or assistance with the Centova Cast API or Centova Cast integration as part of our Centova Cast support services.

2.1 API Types

For developer convenience, Centova Cast provides multiple application programming interfaces (APIs) to access its internal management functionality. All of the interfaces are functionally identical, and simply serve as alternate front-ends to the same back-end functionality.

Centova Cast itself uses an internal, native interface to the same management functionality in its own web interface, thereby ensuring that the results of any operation performed via the API will be identical to the results of performing the same operation via the web interface.

2.1.1 JSON API

The Centova Cast JSON API provides a simple HTTP GET/POST-based interface to the API which returns JSON-encoded responses. Simple JSON manipulation libraries are available for almost all programming languages, making this a convenient and easy-to-use integration option.

2.1.2 XML API

The Centova Cast XML API provides a simple XML-over-HTTP interface to the API which accepts and returns XML-encoded requests and responses. Much like JSON, XML is almost universally supported across all programming languages for convenient integration with third-party applications.

2.1.3 Session API

The Centova Cast Session API provides an interface similar to that of the JSON API, except that it works with the current user's login session and is designed to be used to implement custom functionality in the Centova Cast web interface via AJAX requests.

2.1.4 Commandline

The Centova Cast Commandline Management Utility provides a basic interface to the Centova Cast automation API that can be used from the UNIX shell prompt or shell scripts.

2.2 Class Types

API methods are broken down into two classes: `system` and `server`.

System methods generally correspond to tasks that would be performed by the administrator, and which would pertain to the overall management and administration of Centova Cast.

Server methods correspond to management tasks for a single streaming server account.

2.2.1 System Class Authentication

Because they operate on a global (server-wide) basis, all methods of the `system` class (except where otherwise noted) require a `password` argument specifying the Centova Cast administrator password in addition to the arguments noted for each `system` class method. Without the administrator password, all methods of the `system` class will return an authentication failure error.

The methods available under the system class are provided under System Class Method Reference.

2.2.2 Server Class Authentication

Because they operate on a per-account basis, all methods of the `server` class (except where otherwise noted) require a `username` argument specifying the username of the streaming server account to be manipulated, and a `password` argument specifying the password for the account, in addition to the arguments noted for each `server` class method.

If desired, the `password` argument may be replaced with the word `admin`, followed by a pipe character (`|`) followed by the Centova Cast administrator password. For example, if the Centova Cast administrator password is `secret`, it is acceptable to specify `admin|secret` as the password for any streaming server account. This allows the administrator to manage any account without knowing the password for each individual account.

The methods available under the server class are provided under Server Class Method Reference.

2.3 JSON API

The Centova Cast JSON API provides a simple JSON-based interface to Centova Cast's automation API. API methods are called via standard HTTP GET or POST requests to the api.php script in the Centova Cast web root, in which the request parameters are passed as simple GET or POST variables.

Centova Cast responds to JSON API requests by returning JSON-encoded response data as the payload of the HTTP response.

2.3.1 JSON API Request Structure

Requests are passed to the JSON API server via GET or POST requests including the following query variables:

- `xm` - Specifies the class name and method to invoke, separated by a period.
  Example: `xm=system.info`
- `f` - Always set to `json`, to request a JSON-formatted response.
  Example: `f=json`
- `a` - Encapsulates an array of parameters for the API method, in the format `a[name]=value`.
  Example: `a[password]=secret`

Example

A typical JSON request might look something like the following:

```
http://example.com:2199/api.php?xm=server.getstatus&f=json&a[username]=jdoe&a[password]=secret
```

The URL above would indicate a request to the `getstatus` method of the `server` class. Two arguments, `username` and `password`, are provided with values `jdoe` and `secret`, respectively.

While the JSON API supports both GET and POST requests, it is typically more secure to use POST requests since GET request parameters (which may contain passwords) are logged to the web server's access log.

2.3.2 JSON API Response Structure

A JSON response object always contains two top-level properties:

- a `type` property indicating the type of result: `success` (corresponding to `CSuccess` in the API method reference sections) if the request was successful, or `error` (corresponding to `CError` in the API method reference sections) if an error occurred, and
- a `response` property whose value is an object providing the details of the response

Response Content

Within the `response` object, a `message` property always contains the a textual description of the result of the request.

The `response` object may also include a `data` property containing result data generated by the request. For information about the structure of the data within the `data` property, consult the *Return Value* section of the specific API method you wish to call.

Example

A typical JSON response might look something like the following:

```
{
    "type":"success",
    "response":{
    "data":[
        {
        "username":"jdoe",
        "state":"up",
        "expected":"up",
        "sourcestate":"up",
        "sourceexpected":"up"
        }
    ],
    "message":"Check complete"
    }
}
```

The response packet above would indicate that a request was successfully processed by Centova Cast. One result row was generated by the request, which contained fields entitled username, state, expected, sourcestate, and sourceexpected, whose values were jdoe, up, up, up, and up, respectively.

Had the response included additional result rows, these would have been represented by additional elements within the `data` array.

2.4 XML API

The Centova Cast XML API provides a simple XML-over-HTTP interface to Centova Cast's automation API. API methods are called via a standard HTTP POST request to the api.php script in the Centova Cast web root, in which an XML request packet is provided as POST data. Centova Cast responds to API requests by returning an XML response packet as the payload of the HTTP response.

2.4.1 XML Request Packet Structure

An XML request packet always begins with a standard XML header, followed by a single `<centovacast>` element.

Request Stanza

Inside the `<centovacast>` element, a single `<request>` element identifies the packet as a request packet. The `<request>` element must include both a class attribute indicating the API method class (described under the *API Method Classes* sections below), as well as a method attribute indicating the API method to be called. All API methods available are described in detail under their respective headings below under the Server Class Method Reference and System Class Method Reference sections below.

Parameters

Within the `<request>` element, one or more additional elements may be provided as arguments to the API method. The name of each argument element must correspond to the name of an argument accepted by the requested API method, and the contents of each argument element must specify the value of each argument.

Example

A typical XML request packet might look something like the following:

```
<?xml version="1.0" encoding="UTF-8"?>
<centovacast>
    <request class="system" method="info">
        <password>secret</password>
        <username>jdoe</username>
    </request>
</centovacast>
```

The packet above would indicate a request to the `info` method of the `system` class. Two arguments, `username` and `password`, are provided with values `jdoe` and `secret`, respectively.

Note that while the character encoding is specified by convention in the XML preamble, Centova Cast always expects UTF-8 character encoding in request packets regardless of the encoding specified in the XML preamble.

2.4.2 XML Response Packet Structure

An XML response packet always begins with a standard XML header, followed by a single `<centovacast>` element with a `version` attribute indicating the Centova Cast version, and a `host` attribute indicating the hostname of the Centova Cast web interface serving the request.

Response Type

Inside the `<centovacast>` element, a single `<response>` element identifies the packet as a response packet. The `<response>` element always includes a `type` attribute indicating the type of result: `success` (corresponding to `CSuccess` in the API method reference sections) if the request was successful, or `error` (corresponding to `CError` in the API method reference sections) if an error occurred.

Response Content

Within the `<response>` element, a single `<message>` attribute always contains a textual description of the result of the request.

The element may also include a `<data>` element containing result data generated by the request. For information about the structure of the data within the `<data>` element, consult the *Return Value* section of the specific API method you wish to call.

Example

A typical XML response packet might look something like the following:

```
<?xml version="1.0" encoding="UTF-8"?>
<centovacast version="3.0.0" host="example.com:2199">
    <response type="success">
        <message>Check complete</message>
        <data>
            <row>
                <username>jdoe</username>
                <state>up</state>
                <expected>up</expected>
                <sourcestate>up</sourcestate>
                <sourceexpected>up</sourceexpected>
            </row>
        </data>
    </response>
</centovacast>
```

The response packet above would indicate that a request was successfully processed by Centova Cast version 3.0.0. One result row was generated by the request, which contained fields entitled username, state, expected, sourcestate, and sourceexpected, whose values were jdoe, up, up, up, and up, respectively.

Had the response included additional result rows, these would have been represented by additional `<row>` elements within the `<data>` elements.

2.5 Session API

The Centova Cast Session API is very similar to the JSON API, with the notable exception that its method calls do not accept `username` or `password` parameters. Instead, it operates in the context of the current Centova Cast user session.

The Session API is primarily useful for adding functionality directly to the Centova Cast interface. This can be accomplished (by experienced developers) by modifying the Centova Cast web interface templates and adding JavaScript code which performs custom AJAX requests to the Session API.

API methods are called via standard HTTP POST requests to the `sessionapi.php` script in the Centova Cast web root. Note that unlike the JSON API, HTTP GET requests to the Session API are *not* permitted, as these would make the API vulnerable to CSRF (Cross-Site Request Forgery) attacks.

Just like the JSON API, Centova Cast responds to Session API requests by returning JSON-encoded response data as the payload of the HTTP response.

2.5.1 Session API Request Structure

Requests are passed to the Session API server via POST requests including the following query variables:

- `m` - Specifies the class name and method to invoke, separated by a period.
  Example: `m=system.info`
- `a` - Encapsulates an array of parameters for the API method, in the format `a[name]=value`.
  Example: `a[password]=secret`

Example

A typical HTTP request to the Session API might look something like the following:

```
POST /sessionapi.php HTTP/1.0
Content-Type: application/x-www-form-urlencoded
Content-Length: 33

m=server.switchsource&a[state]=up
```

The request above would indicate a request to the `switchsource` method of the `server` class. One argument, `state`, is provided with the value of `up`. When invoked via an AJAX request from an HTML template in the Centova Cast user interface, this particular example would start the autoDJ for the currently logged-in user.

2.5.2 Session API Response Structure

The Session API returns responses identical to those of the JSON API. A JSON response object always contains two top-level properties:

- a `type` property indicating the type of result: `success` (corresponding to `CSuccess` in the API method reference sections) if the request was successful, or `error` (corresponding to `CError` in the API method reference sections) if an error occurred, and
- a `response` property whose value is an object providing the details of the response

Response Content

Within the `response` object, a `message` property always contains the a textual description of the result of the request.

The `response` object may also include a `data` property containing result data generated by the request. For information about the structure of the data within the `data` property, consult the *Return Value* section of the specific API method you wish to call.

Example

A typical JSON response might look something like the following:

```
{
    "type":"success",
    "response":{
        "data":{
            "size":"1048576",
            "files":"16"
        },
        "message":"Updated"
    }
}
```

The response packet above would indicate that a request was successfully processed by Centova Cast. A data object with properties `size` and `files` was generated by the request.

2.5.3 JSON and XML APIs - Asynchronous API Responses

Some API requests, such as media library updates and log archivals, may take an inordinately long time to complete. These requests are normally processed synchronously – that is, the API client must remain connected until the request completes in order to receive the response – which may be undesirable in interactive environments where a user must wait for the operation to complete.

To better handle such cases, Centova Cast provides an asychronous response mechanism through which the API client can submit the request, disconnect, and then receive the results of the API call via a postback to a client-supplied URL.

2.5.4 Submitting a Request for Asynchronous Processing

Asynchronous requests are mostly identical to synchronous requests, so it is important to understand how to submit standard API requests before implementing asynchronous processing in your API client.

Asynchronous requests are handled in a process of three steps:

1. **Request**

   The API client initiates a normal request to the XML or JSON API, but provides a callback URL via the `callbackurl` parameter.

   For example, a request to following JSON API endpoint would normally request the status for the user account named `jdoe`:

```
http://example.com:2199/api.php?xm=server.getstatus&f=json&a[username]=jdoe&a[password]=s
```

To submit this request for asynchronous processing, simply append a ‘callbackurl‘ parameter:

```
http://example.com:2199/api.php?xm=server.getstatus&f=json&a[username]=jdoe&a[password]=s
```

2. **Initial Response**

   The API server will immediately respond to the API request to confirm that it was submitted successfully:

```
{
    "type":"success",
    "response":{
        "data":[
            {
                "callbackurl"=>"http://example.com/myhandler.php"
            }
        ],
        "message":"Asynchronous request started"
    }
}
```

   Alternately, the API server may return an error if there was a problem authenticating the request, or if the requested API method was not valid.

   Unlike a response to a synchronous request, this response does *not* indicate the results of the call to the API method itself; this response only indicates whether or not the request was successfully submitted for processing.

   After sending this initial response, Centova Cast closes the connection to the API client.

3. **Callback**

   The API server continues to process the request asynchronously. The results of the call to the API method will be posted to the callback URL (in this example, `http://example.com/myhandler.php`) when processing is completed.

While asynchronous processing is typically only useful for long-running API calls, it is possible to submit any JSON or XML API method call for asynchronous processing via the callback mechanism.

2.5.5 Handling the Response Callback

Centova Cast submits the response to the callback URL as an HTTP POST request wherein the API response content is provided as POST data.

The API response content provided to the callback is identical to that which would have been received through a synchronous API call. When making an asynchronous request to the JSON API, the callback will receive a JSON-encoded response; when accessing the XML API, the callback will receive an XML-encoded response.

The HTTP response code and/or output from the callback script is ignored by Centova Cast, except for the purpose of logging non-2xx response codes to the Centova Cast event log.

2.5.6 JSON API Response Callback Example

When submitting the following request to the JSON API:

```
http://example.com:2199/api.php?xm=server.getstatus&f=json&a[username]=jdoe&a[password]=secret&
```

The handler at `http://example.com/myhandler.php` will receive the following postback from Centova Cast:

```
POST /myhandler.php HTTP/1.0
Host: example.com
User-Agent: Centova Cast/3.x.x
Connection: close
Content-Length: 312
Content-Type: application/json

{
    "type":"success",
    "response":{
        "data":[
            {
                "username":"jdoe",
                "state":"up",
                "expected":"up",
                "sourcestate":"up",
                "sourceexpected":"up"
            }
        ],
        "message":"Check complete"
    }
}
```

Your custom `myhandler.php` script might handle the callback as follows:

```php
<?php
$json = file_get_contents('php://input');
$result = json_decode($json);
record_response(
    'Received a response of type ' . $result->type . ' containing data: ' .
    print_r($result->response->data,true)
);

function record_response(/* ... */) {
    // do something appropriate with the response
}
```

2.5.7 XML Response Callback Example

When submitting the following request to the XML API:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<centovacast>
    <request class="system" method="info">
        <password>secret</password>
        <username>jdoe</username>
        <callbackurl>http://example.com/myhandler.php</callbackurl>
    </request>
</centovacast>
```

The handler at `http://example.com/myhandler.php` will receive the following postback from Centova Cast:

```
POST /myhandler.php HTTP/1.0
Host: example.com
User-Agent: Centova Cast/3.x.x
Connection: close
Content-Length: 478
Content-Type: text/xml

<?xml version="1.0" encoding="UTF-8"?>
<centovacast version="3.x.x" host="example.com:2199">
    <response type="success">
        <message>Check complete</message>
        <data>
            <row>
                <username>jdoe</username>
                <state>up</state>
                <expected>up</expected>
                <sourcestate>up</sourcestate>
                <sourceexpected>up</sourceexpected>
```

```
            </row>
        </data>
    </response>
</centovacast>
```

Your custom `myhandler.php` script might handle the callback as follows:

```
<?php
$xml = file_get_contents('php://input');
$result = simplexml_load_string($xml);
record_response(
    'Received a response of type ' . $result->response['type'] . ' containing data: ' .
    print_r($result->response->data,true)
);

function record_response(/* ... */) {
    // do something appropriate with the response
}
```

2.5.8 API Errors

The results of an unsuccessful API call are posted to the callback URL in exactly the same manner as a successful API result, and may be handled in exactly the same manner as an error received from a synchronous API call.

2.5.9 Postback Failures

Centova Cast will attempt to submit the API result to the callback URL precisely once. It will neither queue nor retry a failed postback; in the event of a failure, the result data will be lost.

Failed postbacks are, however, logged to the Centova Cast event log which may be accessed by the administrator of the Centova Cast server.

2.6 Server Class Method Reference

The sections below describe the methods available under the server class.

2.6.1 Get Account Settings

Retrieves the configuration for a CentovaCast client account. If autoDJ support is enabled, the configuration for the autoDJ is returned as well.

Method

getaccount

Arguments

None.

Return Value

A result of type `CSuccess` is returned if the account could be accessed successfully, otherwise `CError` is returned.

An array is returned with the following structure:

- **account** `(array)`
  The details for the account, whose elements correspond to the arguments passed to the `reconfigure` method.

Additional fields may be returned depending on the server and autoDJ application being used for the stream. These fields will specify additional configuration information for the corresponding server or autoDJ application.

2.6.2 Get Stream Status

Retrieves status information from the streaming server for a CentovaCast client account. When using a streaming server that supports multiple mount points (ShoutCast v2, IceCast) only the “default” mount point is checked, but additional mount points can also be queried by passing them as arguments

Method

getstatus

Arguments

The following arguments are accepted:

- **mountpoints** (string)
  A comma-delimited list of mount points whose status should also be checked.

  Example:

  /mounttocheck - Return status information for /mounttocheck in addition to the default mount point.
  /mounttocheck,/anothermount - Return status information for two mount points in addition to the default.
  all - Return status information for all of the server's mount points.

Return Value

A result of type CSuccess is returned if the account could be accessed successfully, otherwise CError is returned.

An array of data is returned with the following structure:

- **status** (array)
  An array of status information for the stream and default mount point.
  - **mount** (string)
    Indicates the mount point on which the stream is broadcasting.
  - **sid** (int)
    Indicates the UVOX stream ID for this mount point (ShoutCast DNAS v2 only)
  - **listenercount** (int)
    Indicates the number of listeners currently tuned in to the stream.
  - **genre** (string)
    Indicates the genre of the stream, as provided by the streaming source.
  - **url** (string)
    Indicates the URL for the stream, as provided by the streaming source.
  - **title** (string)
    Indicates the title for the stream, as provided by the streaming source.
  - **currentsong** (string)
    Indicates the title (and possibly artist) of the current track being played by the streaming server.
  - **bitrate** (int)
    Indicates the bit rate at which the current stream is being broadcasted, as provided by the streaming source.
  - **sourceconnected** (int)
    Indicates whether a source is connected to the server.

    Possible values include:

        * `1` - source is connected
        * `0` - source is not connected
    - **serverstate** `(int)`
      1 if the server is up, otherwise 0.
    - **sourcestate** `(int)`
      1 if the autoDJ is up, otherwise 0.
    - **appstate** `(array)`
      A list of states for each application associated with this account, keyed by application type.
    - **state** `(int)`
      The state of the given application.
    - **reseller** `(int)`
      1 if this account is a reseller, otherwise 0.
    - **ipaddress** `(string)`
      The IP address for this stream.
    - **port** `(int)`
      The port number for this stream.
    - **proxy** `(int)`
      1 if this account is permitted a web proxy, otherwise 0.
    - **servertype** `(string)`
      The server type for this stream.
    - **sourcetype** `(string)`
      The autoDJ type for this stream.

- **mountpoints** `(array)`
  A list containing rows for each requested mount point.

    - **row** `(array)`
      Status details for the mount point (one row for each mount point)
        * **mount** `(string)`
          Indicates the mount point on which the stream is broadcasting.
        * **sid** `(int)`
          Indicates the UVOX stream ID for this mount point (ShoutCast DNAS v2 only).
        * **listenercount** `(int)`
          Indicates the number of listeners currently tuned in to the stream.
        * **genre** `(string)`
          Indicates the genre of the stream, as provided by the streaming source.
        * **url** `(string)`
          Indicates the URL for the stream, as provided by the streaming source.
        * **title** `(string)`
          Indicates the title for the stream, as provided by the streaming source.
        * **currentsong** `(string)`
          Indicates the title (and possibly artist) of the current track being played by the streaming server.

    * **bitrate** (int)
      Indicates the bit rate at which the current stream is being broadcasted, as provided by the streaming source.
    * **sourceconnected** (int)
      Indicates whether a source is connected to the mount point.
      Possible values include:
        - 1 - source is connected
        - 0 - source is not connected

2.6.3 Copy File

Copies a file into a subdirectory of the var/spool/ directory for a CentovaCast client account. This is typically used to install media files (such as the stream introduction or fallback files, or media for use by the server-side streaming source) to a given account with the appropriate ownership and privileges.

Method

copyfile

Arguments

The following arguments are accepted:

- **sourcefile** (string)
  Specifies the *absolute* path and filename to a file on the server to be copied. This file must reside on the same server on which the Centova Cast web interface is running, and must be readable by the 'centovacast' user account.

  Example:

  /tmp/myfile.mp3

- **destfile** (string)
  Specifies the *relative* path and filename (relative to the client account's var/spool/ directory) to which the source file should be copied.

  Example:

  media/myfile.mp3 - This example would copy the file to var/spool/media/myfile.mp3 under the client account's root directory.

Return Value

A result of type CSuccess is returned if the file was copied successfully, otherwise CError is returned.

One result row is returned containing the following element(s):

- **filename** (string)
  the filename of the destination file

2.6.4 Retrieve Logs

Retrieves the streaming server or autoDJ logs for a stream.

Method

getlogs

Arguments

The following arguments are accepted:

- **type** `(string)`
  The log type to retrieve.

  Possible values include:

  - `error` - Retrieves the stream's error log.
  - `access` - Retrieves the stream's access log.
  - `source` - Retrieves the stream's source (autoDJ) log.

- **page** `(int)`
  The page number to retrieve from the log; a "page" is an arbitrary measurement which currently corresponds to a 10KB block of log data.

Return Value

A result of type `CSuccess` is returned if the log was retrieved successfully, otherwise `CError` is returned.

One result row is returned containing the following element(s):

- **thispage** `(int)`
  The page number of the retrieved page.
- **totalpages** `(int)`
  The total number of pages available in this log.
- **log** `(string)`
  The log data for the requested page.

2.6.5 Get Song History

Retrieves a list of tracks that were recently broadcasted on a stream.

Method

getsongs

Arguments

The following arguments are accepted:

- **mountpoints** `(string)`
  a comma-delimited list of mount points whose song lists should also be retrieved

  Example:

  `/mounttocheck` - Return song list for `/mounttocheck` in addition to the default mount point
  `/mounttocheck,/anothermount` - Return song lists for two mount points in addition to the default
  `all` - Return song lists for all of the server's mount points

Return Value

A result of type `CSuccess` is returned if the song list was retrieved successfully, otherwise `CError` is returned.

Two result rows are returned with the following structure:

- **songs** `(array)`
  an array of song information for the default mount point
  - **title** `(string)`
    Indicates the title (and possibly artist) of the track.
  - **royaltytrackid** `(int)`
    If royalty mode is enabled, returns the track ID for the track.
  - **time** `(int)`
    Indicates the time at which the track was played (if available) as a UNIX timestamp, or 0 if unavailable
- **mountpoints** `(array)`
  a list containing rows for each requested mount point
  - **row** `(array)`
    songs list for the mount point (one row for each mount point)
    - **title** `(string)`
      Indicates the title (and possibly artist) of the track.
    - **royaltytrackid** `(int)`
      If royalty mode is enabled, returns the track ID for the track.
    - **time** `(int)`
      Indicates the time at which the track was played (if available) as a UNIX timestamp, or 0 if unavailable

2.6.6 Get Listener List

Retrieves the list of listeners which are currently tuned in to a stream.

Method

getlisteners

Arguments

The following arguments are accepted:

- **mountpoints** (string)
  a comma-delimited list of mount points whose listener lists should also be retrieved

  Example:

  /mounttocheck - Return listener list for /mounttocheck in addition to the default mount point
  /mounttocheck,/anothermount - Return listener lists for two mount points in addition to the default
  all - Return listener lists for all of the server's mount points

Return Value

A result of type CSuccess is returned if the listener list was retrieved successfully, otherwise CError is returned.

Two result rows are returned with the following structure:

- **listeners** (array)
  an array of listener information for the default mount point
  - **hostname** (string)
    the hostname of the listener
  - **useragent** (string)
    the user agent of the listener
  - **elapsed** (int)
    the number of seconds for which the listener has been connected
- **mountpoints** (array)
  a list containing rows for each requested mount point
  - **row** (array)
    listener list for the mount point (one row for each mount point)
    * **hostname** (string)
      the hostname of the listener
    * **useragent** (string)
      the user agent of the listener
    * **elapsed** (int)
      the number of seconds for which the listener has been connected

2.6.7 Get Account State

Retrieves basic account state information for a given account.

Method

enabled

Arguments

None.

Return Value

A result of type `CSuccess` is returned if the account was retrieved successfully, otherwise `CError` is returned.

One data row is returned containing the following elements:

- **enabled** `(int)`
  Indicates whether the account is enabled

  Possible values include:

  - `0` - indicates that the account is disabled
  - `1` - indicates that the account is enabled

- **username** `(string)`
  Indicates the username for the account

- **reseller** `(int)`
  Indicates whether this is a reseller account

  Possible values include:

  - `0` - indicates that the account is not a reseller
  - `1` - indicates that the account is a reseller

2.6.8 Validate Account Credentials

Tests whether or not a username/password is valid and able to log in, and if it is, indicates the type of account.

Method

authenticate

Arguments

None.

Return Value

A result of type `CSuccess` is returned if the username/password matches a valid account (client, reseller, or DJ), otherwise `CError` is returned. If the username/password is valid for a DJ, but the DJ is not authorized to log in (due to time restrictions, etc.), `CError` is returned.

One data row is returned containing zero or more of the following elements:

- **username** `(string)`
  Indicates the username for the account; if this is a DJ account, the parent account's username is provided here
- **dj** `(int)`
  Indicates whether the account is a DJ account
  Possible values include:
  - `1` - indicates that the account is a DJ
- **reseller** `(int)`
  Indicates whether this is a reseller account
  Possible values include:
  - `1` - indicates that the account is a reseller

2.6.9 Start Stream

Starts a streaming server for a Centova Cast account. If autoDJ support is enabled, the autoDJ is started as well.

Method

start

Arguments

The following arguments are accepted:

- **noapps** `(int|string)`
  Specifies whether dependent application startup should be suppressed
  Possible values include:
  - `0` - start the dependent applications (i.e., the autoDJ) if any exist and are enabled
  - `1` - do not start the dependent applications, even if they exist and are enabled
  - `appname1,appname2,...` - do not start the applications specified in the comma-separated list

Return Value

A result of type `CSuccess` is returned if the stream was started successfully, otherwise `CError` is returned. No data is returned by this method.

2.6.10 Reload Stream

Reloads the streaming server configuration for a Centova Cast account. If autoDJ support is enabled, the configuration and playlist for the autoDJ is reloaded as well. This will not interrupt any listener sessions.

Method

reload

Arguments

None.

Return Value

A result of type `CSuccess` is returned if the stream was reloaded successfully, otherwise `CError` is returned. No data is returned by this method.

2.6.11 Restart Stream

Restarts a streaming server for a Centova Cast account. If autoDJ support is enabled, the autoDJ is restarted as well. Note that this is functionally equivalent to stopping and then starting the stream, so it will disconnect all listeners from the streaming server.

Method

restart

Arguments

None.

Return Value

A result of type `CSuccess` is returned if the stream was restarted successfully, otherwise `CError` is returned. No data is returned by this method.

2.6.12 Stop Stream

Stops a streaming server for a Centova Cast account. If autoDJ support is enabled, the autoDJ is stopped as well.

Method

stop

Arguments

None.

Return Value

A result of type `CSuccess` is returned if the stopped was started successfully, otherwise `CError` is returned. No data is returned by this method.

2.6.13 Activate/Deactivate autoDJ

Starts or stops the autoDJ for a Centova Cast account. This can be used to stop the autoDJ before a live stream commences, and start it afterward.

Method

switchsource

Arguments

The following arguments are accepted:

- **state** `(string)`
  Specifies the new state for the streaming source.

  Possible values include:

  - `up` - Activates the streaming source
  - `down` - Deactivates the streaming source

Return Value

A result of type `CSuccess` is returned if the autoDJ state was changed successfully, otherwise `CError` is returned. No data is returned by this method.

2.6.14 Update Media Library

Updates the media library for a Centova Cast account's autoDJ. A client's media library must be re-indexed every time the client uploads media files to (or removes media files from) his autoDJ's media directory on the server, otherwise the disk changes will not be reflected in Centova Cast. Note that changes made via Centova Cast's web-based file manager (including media uploaded via the file manager's file-upload interface) are automatically applied to the media library and do not require a media library update. Similarly changes made via Centova Cast's internal FTP server are automatically applied to the media library as well. This method exists primarily for use when media will be added to an account's `var/spool/media/` directory on the server via some external process.

Method

reindex

Arguments

The following arguments are accepted:

- **intoplaylist** `(int)`
  the ID of a playlist to which all imported tracks should be added
- **intoplaylistname** `(string)`
  similar to `intoplaylist`, but specifies the name (or name fragment) of a playlist to which all imported tracks should be added
- **ignoremissingplaylist** `(int)`
  specifies whether an error should be generated if the playlist ID specified by `intoplaylist` or `intoplaylistname` does not exist

  Possible values include:

  - `0` - abort with an error if the `intoplaylist` or `intoplaylistname` argument is used but the specified playlist ID does not exist
  - `1` - silently ignore errors and continue processing as if `intoplaylist` or `intoplaylistname` was not specified if the specified playlist ID does not exist
- **updateall** `(int)`
  specifies whether a quick or full update should be performed

  Possible values include:

  - `0` - perform a quick update, only indexing newly-added tracks and removing tracks that no longer exist on disk
  - `1` - perform a full update, checking for new and removed tracks AND re-scanning all existing (previously imported) tracks for metadata changes
- **clearcache** `(int)`
  specifies whether the album data cache should be cleared before reindexing

  Possible values include:

  - `0` - do not clear the cache
  - `1` - clear the cache

- **progress** (`string`)
  the progress identifier to use (otherwise autogenerate)

Return Value

A result of type `CSuccess` is returned if the media library update completed successfully, otherwise `CError` is returned. No data is returned by this method.

2.6.15 Silently Update Media Library

Updates the media library for a Centova Cast account in a manner similar to the *Update Media Library* method's quick update, but with the following differences: 1. Authentication is not required; this method may be called anonymously. 2. No arguments of any kind are accepted, to ensure that playlists cannot be manipulated, etc. 3. No output is generated (even upon an error condition) which could reveal any information about the client's account or the media it contains. This method is used internally by Centova Cast's built-in FTP server to request a media library update after a client FTP session, but may be used for other purposes as needed.

Method

autoreindex

Arguments

None.

Return Value

A result of type `CSuccess` is returned if the media library was updated successfully. Upon error, a result of type `CError` is returned with an intentionally-vague "An error occurred" error message. No data is returned by this method.

2.6.16 Manage Playlists

Method

playlist

Arguments

The following arguments are accepted:

- **action** (string)
  Specifies the playlist action to perform

  Possible values include:

  - activate - Activates the selected playlist. Either playlist or playlistname must also be specified as arguments indicating the playlist to activate.
  - deactivate - Deactivates the selected playlist. Either playlist or playlistname must also be specified as arguments indicating the playlist to deactivate.
  - add - Adds tracks to the selected playlist. Either playlist or playlistname must be specified as arguments indicating the playlist to receive the tracks, and one of trackname, trackpath, albumname, or artistname must be specified indicating the tracks to add.
  - remove - Removes tracks to the selected playlist. Either playlist or playlistname must be specified as arguments indicating the playlist from which the tracks should be removed, and one of trackname, trackpath, albumname, or artistname must be specified indicating the tracks to remove.
  - list - Lists the playlists available for this account. If playlist or playlistname is specified, only the matching playlist(s) are returned, otherwise all playlists are returned.

- **playlist** (string)
  Specifies the ID number of the playlist on which action should be performed.

- **playlistname** (string)
  Specifies a keyword matching the name of the playlist on which action should be performed. This is a more convenient alternative to specifying the playlist ID number in playlist.

- **trackname** (string)
  Specifies a keyword matching the name of the track(s) which should be added or removed.

- **trackpath** (string)
  Specifies a keyword matching the filesystem path of the track(s) which should be added or removed. Note that these tracks must already exist in the media library.

- **albumname** (string)
  Specifies a keyword matching the name of the album(s) whose tracks should be added or removed.

- **artistname** (string)
  Specifies a keyword matching the name of the artists(s) whose tracks should be added or removed.

Return Value

A result of type CSuccess is returned if the playlist action was performed successfully, otherwise CError is returned. For all action types except list, no data is returned.

For the `action` type `list`, zero or more rows of result data are returned, each corresponding to a playlist with the following elements:

- **id** (`int`)
  The ID number for the playlist
- **title** (`string`)
  The title for the playlist
- **type** (`string`)
  The type of playlist

  Possible values include:

  - `general` - Indicates a general rotation playlist
  - `scheduled` - Indicates a scheduled playlist
  - `interval` - Indicates an interval playlist
  - `now` - Indicates an immediate playlist
- **scheduled_datetime** (`string`)
  Indicates the start date for a `scheduled` playlist
- **scheduled_repeat** (`string`)
  Indicates the repeat interval for a `scheduled` playlist

  Possible values include:

  - `never` - one-time only
  - `daily` - Playlist repeats on a daily basis
  - `weekly` - Playlist repeats on a weekly basis
  - `monthly` - Playlist repeats on a monthly basis
  - `yearly` - Playlist repeats on an annual basis
- **scheduled_weekdays** (`string`)
  indicates the days of the week on which the `scheduled` playlist repeats, if `scheduled_repeat` is set to `weekly`

  Example:

  `wed` - Playlist repeats on Wednesdays.
  `mon,wed,fri` - Playlist repeats on Mondays, Wednesdays, and Fridays.
- **scheduled_monthdays** (`string`)
  indicates when a monthly `scheduled` playlist repeats

  Possible values include:

  - `date` - Playlist repeats on the anniversary of `scheduled_datetime`
  - `first` - Playlist repeats on the first day of the month
  - `last` - Playlist repeats on the last day of the month

- **interval_type** (string)
  Indicates the type of interval for an interval playlist

  Possible values include:

  - songs - Playlist repeats after x songs have played
  - minutes - Playlist repeats after x minutes have elapsed

- **interval_length** (int)
  Indicates how often an interval playlist repeats

- **general_weight** (int)
  Indicates the weight of a general playlist

- **status** (string)
  Indicates the status of the playlist

  Possible values include:

  - enabled - the playlist is enabled
  - disabled - the playlist is disabled and ignored

- **Indicates** (general_order)
  the playback order of a general playlist

  Possible values include:

  - random - Tracks are chosen in random order
  - sequential - Tracks are chosen sequentially

- **interval_style** (string)
  Indicates the playback style for an interval playlist

  Possible values include:

  - onerandom - Indicates that a single random track plays from the playlist upon invocation
  - playall - Indicates that the playlist plays from beginning to end, sequentially, upon invocation

- **scheduled_interruptible** (int)
  Indicates whether a scheduled playlist is interruptible

  Possible values include:

  - 0 - Playlist is not interruptible
  - 1 - Playlist is interruptible

- **general_starttime** (string)
  Indicates the time of day in which a general playlist is considered to be valid; if general_starttime and general_endtime are both 00:00:00 then the playlist is always considered to be valid

- **general_endtime** (string)
  Indicates the time of day in which a general playlist is no longer considered to be valid; if general_starttime and general_endtime are both 00:00:00 then the playlist is always considered to be valid
- **track_interruptible** (int)
  Indicates whether a track in the current playlist may be interrupted when another playlist is due to begin

2.6.17 Advance to Next Song

Skips to the next song in the playlist for a Centova Cast account's autoDJ. This method will only work with accounts for which autoDJ support is enabled.

Method

nextsong

Arguments

None.

Return Value

A result of type CSuccess is returned if the song was skipped successfully, otherwise CError is returned.

2.6.18 Refresh Disk Usage

Updates the disk usage for an account.

Method

refreshdiskusage

Arguments

None.

Return Value

A result of type CSuccess is returned if the disk usage was updated successfully, otherwise CError is returned.

One result row is returned containing the following element(s):

- **size** (int)
  the total disk usage in bytes
- **files** (int)
  the total number of files
- **quota** (int)
  the disk quota in bytes

2.6.19 Reconfigure Account

Updates the settings for an existing account in Centova Cast. Note that depending on whom is submitting the request (i.e., the admin, a reseller, or a client) access to certain fields may be restricted. For example, a reseller or client cannot change the stream's port number or IP address. If a value is specified for a restricted field it will be silently ignored. Also note that other fields may be supported depending on the server or source types configured for the server. For a canonical list of the fields supported by a particular account, call the `getaccount` method for the account.

Method

reconfigure

Arguments

The following arguments are accepted:

2.6.20 When updating client accounts ...

- **hostname** (string)
  Specifies the hostname for the stream. This hostname should resolve to the IP address specified by the `ipaddress` argument.

  Example:

  `radio.example.com`
- **ipaddress** (string)
  Specifies the IP address on which the streaming server should listen. This IP address must of course be configured on the server on which CentovaCast will be running.

  Example:

  `10.42.128.3`
- **port** (int)
  Specifies the port number on which the streaming server should listen. This port must not already be in use by other CentovaCast streaming servers or other applications running on the server. Use `auto` to have Centova Cast select the next available port automatically.

  Example:

  `8000`

- **maxclients** (int)
  Specifies the maximum number of listeners that may simultaneously tune in to this stream at any given time.

  Example:

  10

- **adminpassword** (string)
  Specifies the password for this stream. This will be used both to administer the streaming server itself, and to allow the client to login to CentovaCast.

  Example:

  secret

- **sourcepassword** (string)
  Specifies the source password for this stream. This will be used to allow streaming sources to connect to the streaming server and begin broadcasting.

  Example:

  secret

- **maxbitrate** (int)
  Specifies the maximum bit rate for this stream, in kilobits per second (kbps). Note that some streaming servers (notably IceCast) do not enforce this setting, but it must still be specified.

  Example:

  128

- **transferlimit** (int)
  Specifies the maximum monthly data transfer for this stream, in megabytes (MB). If you do not wish to impose a limit, specify unlimited.

  Example:

  1000

- **diskquota** (int)
  Specifies the maximum disk space for this stream, in megabytes (MB). If you do not wish to impose a limit, specify unlimited.

  Example:

  100

- **title** (string)
  Specifies the title for the stream. This will be displayed by listeners' media players when they tune into the stream.

  Example:

  XYZ Widgets Streaming Radio

- **genre** (string)
  Specifies the genre of the stream.

  Example:

`Rock`

v2.0.1+

- **url** (`string`)
  Specifies the URL to the web site associated with this stream (if any).

  Example:

  `http:*xyzwidgets.example.com`

- **introfile** (`string`)
  Specifies the path and filename to the introduction audio file for this stream, relative to the streaming host's base directory. If an absolute filename is provided, the file is assumed to exist on the web interface server and will be copied into the account. A single dash (`-`) may be used to remove any existing introduction audio file.

  Example:

  `var/spool/sounds/introduction.mp3`

- **fallbackfile** (`string`)
  Specifies the path and filename to the fallback audio file for this stream, relative to the streaming host's base directory. If an absolute filename is provided, the file is assumed to exist on the web interface server and will be copied into the account. A single dash (`-`) may be used to remove any existing fallback audio file.

  Example:

  `var/spool/sounds/fallback.mp3`

- **autorebuildlist** (`string`)
  Specifies whether or not the playlist should be rebuilt from the stream's server-side media library every time the stream is started or restarted. This has no effect if the `usesource` argument is set to `0`.

  Possible values include:

  - `0` - Do not automatically rebuild the playlist (unless no playlist exists)
  - `1` - Automatically rebuild the playlist

- **charset** (`string`)
  Specifies the character set for the account.

  Example:

  `ISO-8859-1` - Use the Latin-1 character set

  v2.1.4+

- **mountpoints** (`array`)
  A list containing rows for each of the stream's mount points (if supported); the exact per-mountpoint values are server- and source-dependent and are not documented here.

2.6.21 When updating reseller accounts …

- **maxclients** `(int)`
  Specifies the maximum total number of listener slots that the reseller can allocate. If you do not want to limit the listener slots, specify `unlimited`.

  Example:

  `10` - specifies that the reseller can divide up 10 listener slots between his client accounts; i.e., two 5-listener client accounts, one 10-listener client account, etc.

- **resellerusers** `(int)`
  Specifies the maximum total number of client accounts that the reseller can create. If you do not want to limit the client accounts, specify `unlimited`.

- **transferlimit** `(int)`
  Specifies the maximum total monthly data transfer that the reseller can allocate, in megabytes (MB). If you do not wish to impose a limit, specify `unlimited`.

  Example:

  `10000` - specifies that the reseller can divide up 10000MB of monthly data transfer between his client accounts; i.e., two 5000MB client accounts, four 2500MB accounts, etc.

- **diskquota** `(int)`
  Specifies the maximum disk space that the reseller can allocate, in megabytes (MB). If you do not wish to impose a limit, specify `unlimited`.

  Example:

  `10000` - specifies that the reseller can divide up 10000MB of disk space between his client accounts; i.e., two 5000MB client accounts, four 2500MB accounts, etc.

- **resellerbandwidth** `(int)`
  Specifies the total amount of bandwidth that the reseller can allocate, in kilobits-per-second (kbps). If you do not want to limit the bandwidth, specify `unlimited`. If you do not understand the difference between *bandwidth* and *data transfer*, specify `unlimited` here and enter a value for `transferlimit` instead.

  Example:

  `512` - specifies that the reseller can divide up 512kbps of bandwidth between his client accounts; i.e., two 256kbps client accounts, four 128kbps accounts, etc.

- **maxbitrate** `(int)`
  Specifies the maximum bit rate that a client can allocate to a SINGLE CLIENT ACCOUNT. Unlike the other limits this is a per-stream value, and is NOT divided up between the client's accounts.

  Example:

  `128` - specifies that regardless of any other limits, the reseller can *never* create a stream with a bit rate limit higher than 128kbps (i.e., a 192kbps stream would be forbidden).

- **adminpassword** `(string)`
  Specifies the password for the reseller account.

  Example:

  `secret`

- **resellersenderemail** (string)
  Specifies an alternate E-mail address which will be used as the sender or From: address for any notification messages Centova Cast sends to the reseller's clients. If not specified, the address specified in the email field is used instead.

- **resellerdefcharset** (string)
  Specifies the default character set for new accounts created by the reseller.

  Example:

  ISO-8859-1 - Use the Latin-1 character set

  v2.1.4+

2.6.22 Common settings that can be provided for either client or reseller accounts …

- **organization** (string)
  Specifies the company/organization to whom this account belongs.

  Example:

  XYZ Widgets Inc.

- **email** (string)
  Specifies the account's E-mail address. Centova Cast will automatically send notifications to this address when necessary.

  Example:

  xyzwidgets@example.com

- **timezone** (string)
  Specifies the local time zone for the account, as a valid time zone string, used to ensure that the playlist scheduler uses times that make sense to the client. Leave blank or specify auto to use the server's time zone. For reseller accounts, this is the default time zone used for all of the reseller's clients.

  Example:

  America/Los_Angeles - US/Canada Pacific Time (GMT-08:00)
  UTC - Universal Coordinated Time (UTC)

  v3.0.0+

- **allowproxy** (int)
  Specifies whether or not the stream account be permitted to use the port-80 web proxy. For reseller accounts, this indicates whether the client can create proxy-enabled client accounts.

  Possible values include:

  - 0 - Disallow access to the port-80 proxy.
  - 1 - Allow access to the port-80 proxy.

  v2.1.4+

- **locale** (`string`)
  Specifies the locale (language) for the account. For reseller accounts, this is the default language used for all of the reseller's clients.

  Example:

  `en_US` - Use the US English locale.
  `de_DE` - Use the German locale.

  v3.0.0+

- **usesource** (`int`)
  Specifies whether or not the stream uses autoDJ capabilities. For reseller accounts, this is the default setting used for all of the reseller's clients, and if set to 0 the reseller cannot create autoDJ-enabled accounts at all.

  Possible values include:

  - `0` - Use of autoDJ is permitted, but disabled by default
  - `1` - Use of autoDJ is permitted, and enabled by default
  - `2` - Use of autoDJ is prohibited (live source *must* be used)

Return Value

A result of type `CSuccess` is returned if the account was updated successfully, otherwise `CError` is returned. No data is returned by this method.

2.6.23 Manage DJ accounts

Creates, updates, or removes a DJ account for a streaming account.

Method

managedj

Arguments

The following arguments are accepted:

- **action** (`string`)
  Specifies the DJ account management action to perform.

  Possible values include:

  - `provision` - Create a new DJ account.
  - `reconfigure` - Update an existing DJ account.
  - `terminate` - Delete an existing DJ account.
  - `list` - Lists all DJ accounts. This action takes no arguments.
  - `get` - Retrieves the details for a DJ account.

2.6.24 When creating or updating a DJ account…

- **djusername** (string)
  When creating a DJ account, specifies the username for the DJ account. When updating a DJ account, specifies the username of the DJ account to update.

  Example:

  djsample

- **djpassword** (string)
  Specifies the password for this DJ account.

  Example:

  secret

- **realname** (string)
  Specifies the name for the DJ account.

  Example:

  DJ - Sample

- **status** (string)
  Specifies the status for this DJ account.

  Possible values include:

  - enabled - Enable the DJ account.
  - disabled - Disable the DJ account.

- **diskquota** (int)
  Specifies the maximum disk space for this DJ account, in megabytes (MB). To allow the DJ to utilize the full stream account quota, specify 0. Ignored if ftpprivate permission is not assigned.

  Example:

  100

- **permissions** (string)
  Specifies a comma-delimited list of permissions for the DJ account. Including a permission in the list grants it; omitting a permission from the list revokes it.

  Possible values include:

  - controlserver - Allows the DJ to start or stop the streaming server.
  - controlautodj - Allows the DJ to start or stop the autoDJ only. For SHOUTcast v1 servers, this is required in order for a DJ to begin a live broadcast.
  - manageplaylists - Allows the DJ to modify the autoDJ's playlist settings.
  - medialibrary - Allows the DJ to access the media library and add/remove tracks to/from playlists.
  - managefiles - Allows the DJ to access the file manager and upload, move, rename, and delete files.
  - ftpglobal - Allows the DJ to use his username and password to log in via FTP and access all files for the stream. Also grants managefiles permission.

  - `ftpprivate` - Provides a private folder under `media/dj/djusername/` in which the DJ can upload and manage his own private set of media files via FTP or via the file manager.
  - `viewstatistics` - Allows the DJ to view the statistics for the stream.
  - `viewlisteners` - Allows the DJ to view the current listeners for the stream.
  - `viewlogs` - Allows the DJ to view the log files for the stream.

  Example:

  `controlautodj,manageplaylists,medialibrary`

- **login_weekdays** (`string`)
  Specifies a comma-delimited list of days of the week on which the DJ is allowed to log in. If no days are selected, the DJ will not be permitted to log in.

  Example:

  `mon,wed,fri`

- **login_starttime** (`string`)
  Specifies the earliest time of day (relative to UTC) at which the DJ is allowed to log in, in 24-hour time.

  Example:

  `17:00`

- **login_endtime** (`string`)
  Specifies the latest time of day (relative to UTC) at which the DJ is allowed to be logged in, in 24-hour time.

  Example:

  `23:00`

2.6.25 When deleting a DJ account …

- **djusername** (`string`)
  Specifies the username of the DJ account to delete.

  Example:

  `djsample`

2.6.26 When retrieving a DJ account …

- **djusername** (`string`)
  Specifies the username of the DJ account to retrieve.

  Example:

  `djsample`

Return Value

A result of type `CSuccess` is returned if the action was successfully performed, otherwise `CError` is returned.

2.6.27 When listing DJ accounts …

Zero or more rows of data are returned, each representing a DJ account. While the actual list of values returned for each account may vary from version to version, you can typically rely on the presence of the values passed to the `provision` action (with the exception of the `djpassword` value) plus the additional elements listed below. Any other elements should be considered nonstandard, unsupported, and subject to change without notice.

- **id** `(int)`
  the internal ID number for the DJ account

2.6.28 When retrieving a DJ account …

An array is returned with the following structure:

- **account** `(array)`
  The details for the DJ account, whose elements correspond to the arguments passed to the `provision` action (except the `djpassword` value).

2.6.29 For all other actions …

No data is returned by this method.

2.6.30 Download Log Archive

Archives the logs for the account and provides the compressed file for external use.

Method

archivelogs

Arguments

The following arguments are accepted:

- **byurl** `(int)`
  `1` to provide a temporary URL at which the archive can be downloaded for 4 hours, `0` to provide a filename on the web interface server
- **ipaddress** `(string)`
  if `byurl == 1`, specifies the client IP address that will be authorized to download the log archive file; the IP address of the API client will be used (when available) if not provided

Return Value

A result of type `CSuccess` is returned if the logs were compressed successfully, otherwise `CError` is returned.

An array is returned with the following structure:

- **url** (`string`)
  a complete URL to download the archive file containing the logs (if `byurl` == `1`)
- **archivefile** (`string`)
  the absolute filename (on the Centova Cast web interface server, normally under `/tmp`) of the archive file containing the logs (if `byurl` == `0`)
- **origname** (`string`)
  unused (identical to `archivefile`; kept for backwards compatibility only)

2.7 System Class Method Reference

The sections below describe the methods available under the system class.

2.7.1 Sanity Check

Tests communication with Centova Cast.

Method

sanitycheck

Arguments

None.

Return Value

A result of type `CSuccess` is returned if the request was received successfully, otherwise `CError` is returned. No data is returned by this method.

2.7.2 Image Daemon Interface

Manipulates an image via Centova Cast's Image Daemon.

Method

imaged

Arguments

The following arguments are accepted:

- **in** (`string`)
  The full pathname for the input file to read.
- **out** (`string`)
  The full pathname for the output file to create.
- **action** (`string`)
  The action to perform. Currently only `resize` is supported by the API.
- **width** (`int`)
  The width to which the image should be resized
- **height** (`int`)
  The height to which the image should be resized

Return Value

A result of type `CSuccess` is returned if the operation was successful, otherwise `CError` is returned. No data is returned by this method.

2.7.3 Rename Account

Changes the username of an existing account. Note that this feature makes extensive changes to an account and cannot be used while the stream is online.

Method

rename

Arguments

The following arguments are accepted:

- **newusername** `(string)`
  the new username for the account

Return Value

A result of type `CSuccess` is returned if the operation was successful, otherwise `CError` is returned. No data is returned by this method.

2.7.4 Silently Update Media Library

Requests that the web control panel update the media library for the specified account(s). This request may be made anonymously (no authentication required) and to preserve privacy returns no information about the account or the tracks processed.

Method

autoreindex

Arguments

The following arguments are accepted:

- **accounts** `(string)`
  A comma-separated list of usernames specifying the accounts to be reindexed.

  Example:

  ```
  bob
  bob,john,harry
  ```

Return Value

A result of type `CSuccess` is returned if the media library was updated successfully, otherwise `CError` is returned. No data is returned by this method.

2.7.5 Version and Host Information

Obtains Centova Cast version information and general information about the host server.

Method

version

Arguments

None.

Return Value

A result of type `CSuccess` is returned if the request was received successfully, otherwise `CError` is returned.

An array of data is returned with the following structure:

- **web** `(array)`
  an array of details for the web interface node, with the following structure:
  - **version** `(string)`
    the Centova Cast version number
  - **loadavg1** `(float)`
    the server's 1-minute load average
  - **loadavg5** `(float)`
    the server's 5-minute load average
  - **loadavg15** `(float)`
    the server's 15-minute load average
  - **uptime** `(int)`
    the server's uptime in seconds
  - **os** `(string)`
    the server's OS name
  - **osversion** `(string)`
    the server's OS version, if available
  - **accounts** `(int)`
    the total number of Centova Cast accounts
  - **activeaccounts** `(int)`
    the total number of active (non-suspended/disabled) Centova Cast accounts

- **osrelease** (string)
  the OS release identifier (kernel version on Linux)
- **osmachine** (string)
  the OS machine identifier (kernel architecture on Linux)
- **totalram** (int)
  the total amount of RAM on the server, in KB
- **freeram** (int)
  the amount of free RAM on the server, in KB
- **sharedram** (int)
  the amount of shared RAM on the server, in KB
- **bufferram** (int)
  the amount of RAM allocated to buffers on the server, in KB
- **totalswap** (int)
  the total amount of swap space on the server, in KB
- **freeswap** (int)
  the amount of free swap space on the server, in KB
- **procs** (in)
  the number of processes currently running on the server

2.7.6 Provision Account

Provisions a new client streaming server account in Centova Cast. Note that as of Centova Cast v2.2, account templates can and should be used for detailed configuration of new accounts. While the use of account templates is technically optional for user accounts, account templates *must* be used if a reseller account is being created as no other mechanism is provided to distinguish the account type.

Method

provision

Arguments

The following arguments are accepted:

2.7.7 When creating client accounts ...

- **hostname** (string)
  Specifies the hostname for the stream. This hostname should resolve to the IP address specified by the `ipaddress` argument.

  Example:

  `radio.example.com`

- **ipaddress** (string)
  Specifies the IP address on which the streaming server should listen. This IP address must of course be configured on the server on which Centova Cast will be running.

  Example:

  10.42.128.3

- **port** (int)
  Specifies the port number on which the streaming server should listen. This port must not already be in use by other Centova Cast streaming servers or other applications running on the server. Use auto to have Centova Cast select the next available port automatically.

  Example:

  8000

- **rpchostid** (int)
  Specifies the ID number of the hosting server on which this account should be created.

- **maxclients** (int)
  Specifies the maximum number of listeners that may simultaneously tune in to this stream at any given time.

  Example:

  10

- **adminpassword** (string)
  Specifies the password for this stream. This will be used both to administer the streaming server itself, and to allow the client to login to Centova Cast.

  Example:

  secret

- **sourcepassword** (string)
  Specifies the source password for this stream. This will be used to allow streaming sources to connect to the streaming server and begin broadcasting.

  Example:

  secret

- **maxbitrate** (int)
  Specifies the maximum bit rate for this stream, in kilobits per second (kbps). Note that some streaming servers (notably IceCast) do not enforce this setting, but it must still be specified.

  Example:

  128

- **transferlimit** (int)
  Specifies the maximum monthly data transfer for this stream, in megabytes (MB). If you do not wish to impose a limit, specify unlimited.

  Example:

  1000

- **diskquota** (int)
  Specifies the maximum disk space for this stream, in megabytes (MB). If you do not wish to impose a limit, specify unlimited.

  Example:

  100

- **title** (string)
  Specifies the title for the stream. This will be displayed by listeners' media players when they tune into the stream.

  Example:

  XYZ Widgets Streaming Radio

- **genre** (string)
  Specifies the genre of the stream.

  Example:

  Rock

  v2.0.1+

- **url** (string)
  Specifies the URL to the web site associated with this stream (if any).

  Example:

  http://xyzwidgets.example.com

- **introfile** (string)
  Specifies the path and filename to the introduction audio file for this stream, relative to the streaming host's base directory. If an absolute filename is provided, the file is assumed to exist on the web interface server and will be copied into the account. This may be left blank to specify no introduction audio file.

  Example:

  var/spool/sounds/introduction.mp3

- **fallbackfile** (string)
  Specifies the path and filename to the fallback audio file for this stream, relative to the streaming host's base directory. If an absolute filename is provided, the file is assumed to exist on the web interface server and will be copied into the account. This may be left blank to specify no fallback audio file.

  Example:

  var/spool/sounds/fallback.mp3

- **autorebuildlist** (string)
  Specifies whether or not the playlist should be rebuilt from the stream's server-side media library every time the stream is started or restarted. This has no effect if the usesource argument is set to 0.

  Possible values include:

  - 0 - Do not automatically rebuild the playlist (unless no playlist exists)

  - `1` - Automatically rebuild the playlist

- **autostart** `(int)`
  Specifies whether or not the stream should automatically be started after provisioning. Note that this option will *only* be used if the `usesource` option is set to `0` or `2`. (If autoDJ support is enabled, the stream cannot be started until the client has uploaded some media.)

  Possible values include:

  - `0` - Do not automatically start the stream after provisioning. This is the default.
  - `1` - Automatically start the stream after provisioning if `usesource` is set to `0` or `2`.

  v2.0.1+

- **charset** `(string)`
  Specifies the character set for the account.

  Example:

  `ISO-8859-1` - Use the Latin-1 character set

  v2.1.4+

- **servertype** `(string)`
  Specifies the streaming server type for the stream. The selected server type must be installed on the server and enabled in Centova Cast.

  Example:

  `IceCast` - Use IceCast
  `ShoutCast2` - Use ShoutCast DNAS v2
  `ShoutCast` - Use ShoutCast DNAS v1

  v2.2.0+

- **apptypes** `(string)`
  Specifies the supporting application types for the stream, possibly including a streaming source application for autoDJ support. The selected applications must be installed on the server and enabled in Centova Cast.

  Example:

  `icescc` - Use the “ices-cc” application
  `sctrans,foo` - Use the “sc_trans” and “foo” applications

  v3.0.0+

- **sourcetype** `(string)`
  DEPRECATED - use apptypes instead. Specifies the streaming source type for the stream. The selected source type must be installed on the server and enabled in Centova Cast.

  Example:

  `icescc` - Use ices-cc
  `sctrans` - Use sc_trans

  v2.2.0+

- **template** `(string)`
  Specifies the name of the account template to use for this account. The account template must exist in Centova Cast.

  Example:

  `mytemplate` - (Use the template named “mytemplate”.)

  v2.2.0+

2.7.8 When creating reseller accounts …

- **maxclients** `(int)`
  Specifies the maximum total number of listener slots that the reseller can allocate. If you do not want to limit the listener slots, specify `unlimited`.

  Example:

  `10` - specifies that the reseller can divide up 10 listener slots between his client accounts; i.e., two 5-listener client accounts, one 10-listener client account, etc.

- **resellerusers** `(int)`
  Specifies the maximum total number of client accounts that the reseller can create. If you do not want to limit the client accounts, specify `unlimited`.

- **transferlimit** `(int)`
  Specifies the maximum total monthly data transfer that the reseller can allocate, in megabytes (MB). If you do not wish to impose a limit, specify `unlimited`.

  Example:

  `10000` - specifies that the reseller can divide up 10000MB of monthly data transfer between his client accounts; i.e., two 5000MB client accounts, four 2500MB accounts, etc.

- **diskquota** `(int)`
  Specifies the maximum disk space that the reseller can allocate, in megabytes (MB). If you do not wish to impose a limit, specify `unlimited`.

  Example:

  `10000` - specifies that the reseller can divide up 10000MB of disk space between his client accounts; i.e., two 5000MB client accounts, four 2500MB accounts, etc.

- **resellerbandwidth** `(int)`
  Specifies the total amount of bandwidth that the reseller can allocate, in kilobits-per-second (kbps). If you do not want to limit the bandwidth, specify `unlimited`. If you do not understand the difference between *bandwidth* and *data transfer*, specify `unlimited` here and enter a value for `transferlimit` instead.

  Example:

  `512` - specifies that the reseller can divide up 512kbps of bandwidth between his client accounts; i.e., two 256kbps client accounts, four 128kbps accounts, etc.

- **maxbitrate** `(int)`
  Specifies the maximum bit rate that a client can allocate to a SINGLE CLIENT ACCOUNT. Unlike the other limits this is a per-stream value, and is NOT divided up between the client’s accounts.

Example:

`128` - specifies that regardless of any other limits, the reseller can *never* create a stream with a bit rate limit higher than 128kbps (i.e., a 192kbps stream would be forbidden).

- **adminpassword** `(string)`
  Specifies the password for the reseller account.

  Example:

  `secret`

- **resellersenderemail** `(string)`
  Specifies an alternate E-mail address which will be used as the sender or `From:` address for any notification messages Centova Cast sends to the reseller's clients. If not specified, the address specified in the `email` field is used instead.

- **resellerdefcharset** `(string)`
  Specifies the default character set for new accounts created by the reseller.

  Example:

  `ISO-8859-1` - Use the Latin-1 character set

  v2.1.4+

2.7.9 Common settings that can be provided for either client or reseller accounts …

- **username** `(string)`
  Specifies the username for this account. This will be used to log in to Centova Cast.

  Example:

  `jdoe`

- **organization** `(string)`
  Specifies the company/organization to whom this account belongs.

  Example:

  `XYZ Widgets Inc.`

- **email** `(string)`
  Specifies the account's E-mail address. Centova Cast will automatically send notifications to this address when necessary.

  Example:

  `xyzwidgets@example.com`

- **timezone** `(string)`
  Specifies the local time zone for the account, in hours relative to UTC (GMT), used to ensure that the playlist scheduler uses times that make sense to the client. Leave blank or specify `auto` to use the server's time zone. For reseller accounts, this is the default time zone used for all of the reseller's clients.

  Example:

`-8` - UTC -08:00 - Vancouver
`0` - UTC - London
`3` - GMT +03:00 - Moscow

v2.0.1+

- **allowproxy** `(int)`
  Specifies whether or not the stream account be permitted to use the port-80 web proxy. For reseller accounts, this indicates whether the client can create proxy-enabled client accounts.

  Possible values include:

  - `0` - Disallow access to the port-80 proxy.
  - `1` - Allow access to the port-80 proxy.

  v2.1.4+

- **locale** `(string)`
  Specifies the locale (language) for the account. For reseller accounts, this is the default language used for all of the reseller's clients.

  Example:

  `en_US` - Use the US English locale.
  `de_DE` - Use the German locale.

  v3.0.0+

- **usesource** `(int)`
  Specifies whether or not the stream uses autoDJ capabilities. For reseller accounts, this is the default setting used for all of the reseller's clients, and if set to 0 the reseller cannot create autoDJ-enabled accounts at all.

  Possible values include:

  - `0` - Use of autoDJ is permitted, but disabled by default
  - `1` - Use of autoDJ is permitted, and enabled by default
  - `2` - Use of autoDJ is prohibited (live source *must* be used)

Return Value

A result of type `CSuccess` is returned if the account was created successfully, otherwise `CError` is returned. Upon success an array of data is returned with the following structure:

- **account** `(array)`
  an array of details for the new account, with the following structure (and possibly including additional values)

  - **username** `(string)`
    the username for the new account
  - **ipaddress** `(string)`
    the IP address for the new account

  - **port** (int)
    the port number for the new account
  - **servertype** (string)
    the server type for this account
  - **sourcetype** (string)
    the source type for this account

2.7.10 Remove Account

Removes an existing client streaming server account from Centova Cast. The complete account (including all settings, logs, and any source media) will be permanently deleted.

Method

terminate

Arguments

The following arguments are accepted:

- **username** (string)
  Specifies the username of the stream to remove.

  Example:

  jdoe

- **clientaction** (string)
  Specifies how to handle client accounts if removing a reseller account.

  Possible values include:

  - delete - Delete the reseller's client accounts.
  - reparent - Move the reseller's client accounts to another reseller account

- **targetreseller** (string|int)
  Specifies the username or account ID of the reseller account to receive the deleted reseller's client accounts, if clientaction=reparent.

Return Value

A result of type CSuccess is returned if the streaming server account was removed successfully, otherwise CError is returned. No data is returned by this method.

2.7.11 Reparent Account

Moves a client account from one reseller (or the admin account) to another reseller (or the admin account).

Method

reparent

Arguments

The following arguments are accepted:

- **username** (`string`)
  Specifies the username of the stream to reparent.

  Example:

  `jdoe`

- **newreseller** (`string`)
  Specifies the username of the new reseller account to own the account, or `admin` to reparent to the admin account.

  Example:

  `jsmith`

Return Value

A result of type `CSuccess` is returned if the account was moved successfully, otherwise `CError` is returned. No data is returned by this method.

2.7.12 Set Account Status

Changes the status of an existing client account in Centova Cast.

Method

setstatus

Arguments

The following arguments are accepted:

- **status** (`string`)
  Specifies the new status for the account.

  Possible values include:

  - `disabled` - Disables the account. While disabled, the account will not be permitted to log in to Centova Cast, and if it is a client account, its streaming server be shut down (if necessary) and will remain offline until the account is re-enabled. If the account is a reseller account, all of the reseller's client accounts will also be disabled.

- `enabled` - Enables the account. If the account is a client account, its streaming server will not be automatically be started even if it was up prior to being disabled. If the account is a reseller account, all of the reseller's client accounts will be restored to their original state prior to the reseller account being disabled.

Return Value

A result of type `CSuccess` is returned if the account was updated successfully, otherwise `CError` is returned. No data is returned by this method.

2.7.13 Check Stream Outages

Checks the specified account(s) for outages and restarts processes as necessary. Normally this is done automatically by the cron job, but this method allows processes to be checked on-demand as well.

Method

check

Arguments

None.

Return Value

A result of type `CSuccess` is returned on success, otherwise `CError` is returned. No data is returned by this method.

2.7.14 Get Account State

Returns the state (up or down) of one or more Centova Cast streaming server accounts. This can be used to monitor streams to see if any have crashed. (Note that Centova Cast's cron job automatically monitors and restarts crashed streaming servers as well.)

Method

info

Arguments

None.

Return Value

A result of type `CSuccess` is returned if the account information could be retrieved successfully, otherwise `CError` is returned.

Zero or more rows of data are returned, each with the following structure:

- **username** `(string)`
  Indicates the username of the streaming server account that was tested.
- **state** `(string)`
  Indicates the actual state of the streaming server for the account.

  Possible values include:

  - `up` - The streaming server is online.
  - `down` - The streaming server is offline.
- **expected** `(string)`
  Indicates the expected state of the streaming server for the account.

  Possible values include:

  - `up` - The streaming server should be online.
  - `down` - The streaming server should be offline.
- **sourcestate** `(string)`
  Indicates the actual state of the autoDJ for the account.

  Possible values include:

  - `up` - The autoDJ is online.
  - `down` - The autoDJ is offline.
- **sourceexpected** `(string)`
  Indicates the expected state of the autoDJ for the account.

  Possible values include:

  - `up` - The autoDJ should be online.
  - `down` - The autoDJ should be offline.

2.7.15 Get Resource Utilization

Returns the resource utilization (data transfer, disk usage) of one or more Centova Cast streaming server accounts.

Method

usage

Arguments

None.

Return Value

A result of type `CSuccess` is returned if the account information could be retrieved successfully, otherwise `CError` is returned.

Zero or more rows of data are returned, each with the following structure:

- **username** `(string)`
  Indicates the username of the streaming server account.
- **diskquota** `(int)`
  Indicates the disk quota for the account, or -1 for unlimited.
- **transferlimit** `(int)`
  Indicates the data transfer limit for the account, or -1 for unlimited.
- **diskusage** `(int)`
  Indicates the current month's disk usage for the account.
- **transferusage** `(int)`
  Indicates the current month's data transfer usage for the account.

2.7.16 Perform Batch Operations

Runs a ServerControl method on one or more accounts.

Method

batch

Arguments

The following arguments are accepted:

- **method** `(string)`
  The ServerControl method to invoke for each account.
- **username** `(string)`
  A comma-separated list of usernames for which to execute the method, or `all` for all accounts.

Return Value

A result of type `CSuccess` is returned if all arguments are valid, otherwise `CError` is returned.

Zero or more rows of data are returned, each with the following structure:

- **username** `(string)`
  Indicates the username of the streaming server account.
- **result** `(array)`
  Indicates the result returned by the ServerControl method for this account
  - **status** `(string)`
    Indicates the status (`success` or `error`) returned by the method.
  - **message** `(string)`
    Indicates the result message returned by the method.
  - **data** `(mixed)`
    Provides the data returned by the method; data type will match the data returned by the method.

2.7.17 Account List

Returns a list of all accounts; if invoked by admin, all accounts (including resellers) are included, whereas if invoked by a reseller, only the reseller's accounts are included.

Method

listaccounts

Arguments

The following arguments are accepted:

- **start** `(int)`
  the offset of the first account to retrieve
- **limit** `(int)`
  the maximum number of accounts to return
- **filter** `(string)`
  a keyword by which to filter the results

Return Value

A result of type `CSuccess` is returned if the account list could be loaded successfully, otherwise `CError` is returned.

Zero or more rows of data are returned, each representing an account. While the actual list of values returned for each account may vary from version to version, you can typically rely on the presence of the values passed to the `provision` API call, plus the additional elements listed below. Any other elements should be considered nonstandard, unsupported, and subject to change without notice.

- **id** `(int)`
  the internal ID number for the account
- **username** `(string)`
  the username for the account
- **password** `(string)`
  the password hash for the account (UNIX crypt() format)

2.7.18 Process Logs

Processes (and rotates, if required) the log files for all accounts. Normally this is done automatically by the cron job, but this method allows logs to be processed on-demand as well.

Method

processlogs

Arguments

None.

Return Value

A result of type `CSuccess` is returned on success, otherwise `CError` is returned. No data is returned by this method.

2.7.19 Database Import/Export

Imports or exports a database backup for an account. (Database only, no files.)

Method

database

Arguments

The following arguments are accepted:

- **action** `(string)`
  The database action to perform; either `import` or `export`

  Example:

  `import`

- **filename** `(string)`
  The filename for the file to import from or export to.

  Example:

  `/tmp/foo.ccsql`

- **dryrun** `(int)`
  (`action=import` only.) `1` to perform a test run of the import without actually importing anything, `0` to actually import.

- **nointegrity** `(int)`
  (`action=import` only.) `1` to skip the integrity check, `0` to verify the integrity of the database dump.

- **novalidate** `(int)`
  (`action=import` only.) `0` to require a valid signature indicating that this database backup was generated by the same server it's being restored to, `1` to import backups generated on any server.

Return Value

A result of type `CSuccess` is returned if the import or export could be performed successfully, otherwise `CError` is returned. No data is returned by this method.

2.7.20 Account Backup

Creates a complete backup of an account.

Method

backup

Arguments

The following arguments are accepted:

- **username** `(string)`
  The username of the account to back up.

- **nocontent** (int)
  1 to skip user content (media files, cover images, etc.), 0 to include user content.
- **nologs** (int)
  1 to skip log files, 0 to include log files.

Return Value

A result of type CSuccess is returned if the backup could be created successfully, otherwise CError is returned.

One result row is returned containing the following element(s):

- **filename** (string)
  the complete pathname to the backup file that was created on the hosting server

2.7.21 Account Restore

Restores a backup of an account.

Method

restore

Arguments

The following arguments are accepted:

- **username** (string)
  The new username to assign to the restored account.
- **filename** (string)
  The filename of the file containing the backup to import.

  Example:

  /tmp/foo_user_backup.zip
- **rpchostid** (int)
  Specifies the ID number of the hosting server onto which this account should be restored.
- **reseller** (string)
  Specifies the username of the reseller to whom the restored account should be assigned.
- **dryrun** (int)
  1 to perform a test run of the import without actually importing anything, 0 to actually import.
- **nointegrity** (int)
  1 to skip the integrity check, 0 to verify the integrity of the database dump.

- **novalidate** (int)
  0 to require a valid signature indicating that this database backup was generated by the same server it's being restored to, 1 to import backups generated on any server.

- **overwrite** (int)
  1 to overwrite an existing account with the same username if it exists, 0 to fail if the specified username already exists.

Return Value

A result of type CSuccess is returned if the backup could be restored successfully, otherwise CError is returned. No data is returned by this method.

2.7.22 Account Software Change

HIGHLY EXPERIMENTAL. Changes the software applications used by an account. Likely to be buggy. Will probably break your accounts. Don't even bother trying it unless you're a masochist daredevil with a penchant for wanton destruction of client account data.

Method

appchange

Arguments

The following arguments are accepted:

- **newapp** (string)
  The identifier for the new application to replace the existing application of the same class. (eg: if newapp refers to a server application, the server will be changed; if newapp refers to an autoDJ application, the autoDJ will be changed.)

Return Value

A result of type CSuccess is returned if the account was updated successfully, otherwise CError is returned. No data is returned by this method. Note that while Centova Cast will apply as much of the configuration data as possible from the current application to the new application, the new application will still likely require some manual reconfiguration before it will behave as expected due to substantial differences in the options available for each application. Also note that when switching to a software application that requires additional ports, the entire account may need to be moved to a different port range.

2.7.23 Hosting Server List

Returns a list of all hosting servers managed by this web interface.

Method

listhosts

Arguments

None.

Return Value

A result of type `CSuccess` is returned if the host list could be loaded successfully, otherwise `CError` is returned.

Zero or more rows of data are returned, each representing a host.

- **id** `(int)`
  the internal ID number for the host
- **parameters** `(array)`
  the parameters for the host
  - **ipaddress** `(string)`
    the IP address for the host
  - **port** `(int)`
    the port for the host
  - **title** `(string)`
    the title for the host
  - **defaultip** `(string)`
    the default IP address for new accounts created on this host
  - **hostname** `(string)`
    the hostname used to address this host
  - **proxyipaddress** `(string)`
    the IP address for the port 80 proxy on this server
  - **regionid** `(int)`
    the ID number for the region to which this host is assigned
- **status** `(array)`
  information about the status of host
  - **online** `(int)`
    1 if the host is online, 0 if the host is offline
- **accounts** `(array)`
  information about the accounts hosted on this host
  - **licensed** `(int)`
    the number of accounts for which this server is licensed

  - **active** `(int)`
    the number of active accounts on this server
  - **inactive** `(int)`
    the number of inactive accounts on this server

2.7.24 Region List

Returns a list of all regions managed by this web interface.

Method

listregions

Arguments

None.

Return Value

A result of type `CSuccess` is returned if the region list could be loaded successfully, otherwise `CError` is returned.

Zero or more rows of data are returned, each representing a region.

- **id** `(int)`
  the internal ID number for the region
- **title** `(string)`
  the title of the region
- **name** `(string)`
  the short name of the region
- **target** `(string)`
  an identifier indicating the selection method used to choose a host in this region when provisioning a new account
- **targetid** `(int)`
  the ID number of the target host if `target` is set to `serverid`

Chapter 3

System Accounts

3.1 Under Linux

Centova Cast requires two UNIX user accounts for correct operation on a Linux server. The accounts are:

- **centovacast**
  The Centova Cast web interface and related daemons operate under this account.
- **ccuser**
  The Centova Cast control daemon runs under this account. All client data is owned by this account.

These accounts are created automatically during installation and should not be removed.

3.2 Under Windows

Centova Cast requires one unprivileged Windows system account, named `centovacast`, for correct operation of the Windows daemon. All client data is owned by this account. This account is created automatically during installation and should not be removed.

Additionally, Windows Media Services requires that each independent publishing point be owned by a separate Windows user account. As such, Centova Cast will create a new user named `cast_USERNAME` for each Centova Cast user account, where `USERNAME` is the account's Centova Cast username. These accounts are created with as few privileges as possible, and are of course denied interactive logon rights (so that they can't obtain remote desktop sessions).

Chapter 4

Advanced Configurations

This section provides instructions for configuring advanced features and scenarios for Centova Cast.

4.1 Dual v3/v2 Deployment

Centova Cast v3.x can be deployed on the same server as an existing v2.x installation. The procedure for doing so depends on whether your Apache server uses suPHP.

4.1.1 Without suPHP

If you are not using suPHP with your Centova Cast v2.x installation, you can simply install v3 as usual. You should encounter no problems.

4.1.2 With suPHP

If you are using suPHP on your Apache server, the following three steps need to be performed on your Centova Cast v2.x installation prior to installing v3:

- Edit `/home/centovacast/system/runascc/castd.c` and find the line that says:

  ```
  #define CAST_PORT  2199
  ```

  Change that to:

  ```
  #define CAST_PORT  2200
  ```

- Edit `/home/centovacast/system/config.php` and find the line that says:

  ```
  define('DAEMON_PORT',2199);
  ```

  Change that to:

  ```
  define('DAEMON_PORT',2200);
  ```

- Finally, run these commands via SSH (just copy/paste):

  ```
  cd /home/centovacast/system/runascc
  rm -f castd
  make
  killall -9 castd
  chmod a+x /home/centovacast/scripts/castdctl.sh
  /home/centovacast/scripts/castdctl.sh start
  ```

At this point it should be safe to install Centova Cast v3.x.

4.2 Using centovacast.conf

The main Centova Cast configuration file, `/usr/local/centovacast/etc/centovacast.conf`, contains many options for modifying the behavior and appearance of Centova Cast's web interface.

A non-exhaustive list of the options available in `centovacast.conf` is provided below.

4.2.1 Database connection options

DB_NAME

Specifies your MySQL database name.

DB_USER

Specifies your MySQL database username.

DB_PASS

Specifies your MySQL database password.

DB_HOST

Specifies your MySQL database hostname. Typically this is `localhost` unless your MySQL server is running on a separate machine.

Default: `localhost`

4.2.2 Locale configuration

LOCALE

Set the default locale to use for Centova Cast's internationalization features. To see if your locale is supported, look in the `/usr/local/centovacast/system/locale/` directory.

Note that each user can also change his own locale in his account settings, thereby overriding this setting for his account only.

Default: `en_US`

4.2.3 Feature Configuration

FETCH_NEWS

Controls whether Centova Cast will periodically retrieve news from centova.com to display in the admin area.

Default: `true`

RECENT_TRACK_LIMIT

Sets an upper limit on the number of tracks that may be returned by the Recent Tracks lists in Centova Cast. Note that this may be further limited by the streaming server itself, and by the user in the Recent Tracks widget configuration.

Default: `20`

4.2.4 Date and Time

NOTE: The date and time format is updated automatically when you change your locale, and typically does not need to be set here. Uncomment and set these options only if you do not wish to use the default date/time formats for your locale. Note that this will override the formatting for ALL locales on your system, even if your users have chosen a different locale than the server default.

See http://php.net/manual/en/function.date.php for a list of formatting codes that can be used in these settings.

OVERRIDE_TIMEFORMAT

Time format.

Default: `h:i A`

OVERRIDE_DATEFORMAT

Full date format.

Default: `M d, Y`

OVERRIDE_NODAYFORMAT

Date format, no day.

Default: `M, Y`

OVERRIDE_NOYEARFORMAT

Date format, no year.

Default: `M, d`

4.2.5 Track Information Formatting

NOW_PLAYING_FORMAT

Controls the formatting of the “now playing” text displayed in the Centova Cast client area and stream info widgets.

%artist% = artist name
%title% = track title
%album% = album name
%playlist% = playlist name (if on autoDJ)

Set this to an empty value (do NOT comment out) to use the text from the streaming server verbatim.

Default: %artist% - %title%

POSTPROCESS_TRACK_INFO

Specifies whether extraneous data is removed from the artist, album, and song title before display in Centova Cast's client area, stream info widgets, recent tracks widgets, etc. Extraneous data includes text in parentheses at the end of the artist/album/title, track numbers embedded in the title, filename extensions, and other often-undesirable cruft.

Default: true

4.2.6 Event Scripts

EVENT_SCRIPTS

If enabled, the event scripts under /usr/local/centovacast/var/events will be called in response to various events within Centova Cast.

Default: false

EVENT_SCRIPT_TIMEOUT

Specifies the maximum amount of time (in seconds) for which an event script may run before being terminated by Centova Cast. This should be set high enough to allow your event scripts to complete normally, but low enough to not cause unnecessarily long delays in the web interface. Note that in most cases, event scripts are called while a user waits for the web interface to perform a task, so this directly affects the user's experience.

Default: 30

4.2.7 Log Processing

MAX_ARCHIVED_LOGS

Sets the maximum number of archived log files to keep (i.e., error.log.1, error.log.2, etc.) after the logs are rotated.

Default: 4

LOCALIPADDRESS

Specifies one or more IP addresses, separated by commas, to completely exclude from log file processing.

LOGPROCESS_MIN_DURATION

Specifies the minimum duration (in seconds) for a listener session to be recorded in the statistics; sessions shorter than this value will not be recorded. Set to `0` to record ALL sessions regardless of duration.

Default: `1`

4.2.8 Privileges and Policy Enforcement

CLIENT_ENCODER_SETTINGS

If enabled, clients will be permitted to edit their own encoder settings (including sample rate, channels, and so-on).

Default: `false`

CLIENT_BITRATE_MODIFY

If enabled, clients will be permitted to edit their own bit rate limit.

Default: `false`

BITRATE_LIMIT_ENFORCE

If enabled, Centova Cast will enforce bit rate limits normally per the settings configured by the admin on the Centova Cast settings page.

If disabled, Centova Cast will only trigger the bitrate-exceeded event script when a client exceeds his bit rate limit, allowing the admin to perform custom handling of the violation. The stream will not be shut down, the account will not be suspended, and the Bit Rate Limit Exceeded email will not be sent.

Default: `true`

BITRATE_LIMIT_IGNORE

If enabled, Centova Cast will completely skip the bit rate check. Clients will be permitted to broadcast with a live source at any bit rate.

Default: `false`

REQUIRE_SSL

If enabled, Centova Cast will automatically redirect non-secure (`http://`) requests to the secure (`https://`) web interface, thereby requiring the use of SSL.

Default: `false`

4.2.9 Directories and Path Traversal

MEDIA_PARTITION

By default, the disk usage graph at the top of the accounts page in the admin area is based on the free space on whichever partition holds your Centova Cast "vhosts" directory.

If you need to check the space on another partition instead (e.g. if you use symlinks to reference media on another partition), specify the path to the alternate partition here.

QUOTA_SPOOL_ONLY

If enabled, the user's disk quota will only apply to files uploaded to the `/usr/local/centovacast/var/vhosts/USERNA` directory and its subdirectories; any configuration files, log files, etc. will be excluded from the quota. Note that cc-ftpd does not honor this setting (due to technical limitations) and will always apply the quota ONLY to the spool directory, regardless of this setting.

Default: `true`

4.2.10 Streaming Server Hostnames

SELFREF_HOSTNAME

Set this to TRUE if self-referencing links (such as the link to the ShoutCast administration page, and the links to tune into the server) should use the hostname configured for the stream or server instead of its IP address.

Note that enabling this WILL BREAK YOUR STREAMS if you have not correctly configured DNS for the hostnames you've specified. DO NOT ENABLE THIS UNLESS YOU FULLY UNDERSTAND THE CONSEQUENCES OF DOING SO.

Default: `false`

SELFREF_OVERRIDE

Set this to an IP address or hostname to override the hostname portion of all self-referencing links. Note that before enabling this, you should ensure that:

1. All of your streams are configured to listen on the same IP address, and
2. The IP address or hostname below corresponds to the IP address on which your streams are configured to listen.

This may be useful for NAT configurations or to force a specific hostname for other purposes. Be aware, however, that enabling this completely eliminates the ability to control multiple hosting servers with a single Centova Cast web interface.

In a nutshell, you should probably never enable this.

Example: `SELFREF_OVERRIDE=foo.example.com`

HOSTNAME_FROM_HOST

Set this to true if new accounts whose hostnames are set to “auto” (such as those provisioned through billing modules) should receive the same hostname the host server upon which they are provisioned. Set to false to use the stream’s IP address as its hostname.

Default: `true`

4.2.11 Media Library

ALBUM_COVER_WIDTH / ALBUM_COVER_HEIGHT

Specifies the dimensions to which album cover images will be resized. Note that further width/height restrictions may be imposed on the display of these images by cascading stylesheets.

Default: `60`

ALLOW_IMPORT_M3U

If enabled, users will be permitted to import M3U playlist files into Centova Cast playlists. This option is disabled by default as it was added as a courtesy for a small group of advanced users who requested it, and it is **NOT SUPPORTED IN ANY CAPACITY by Centova Technologies** due to the complexities of getting the relative paths correct. Be sure to read this article before enabling.

Default: `false`

DISABLE_FOLDER_VIEW

If set to false, the user will be able to browse his media library by folder. Note that this is a different feature than the File Manager, and will likely be removed in a future release as it largely duplicates the File Manager’s functionality.

Default: `false`

UPLOAD_EXTENSIONS

Specifies the file types (file extensions) that users will be permitted to upload via the web-based File Manager.

This list is intentionally restrictive and in most cases shouldn’t be modified. Don’t do something silly like allowing users to upload scripts.

Also note that this option is commented out by default to allow Centova Cast to automatically generate its own accepted-file-types list, which may include additional file formats in future as we add support for new audio/video streaming formats. If you uncomment this and set your own file types, you will need to manually add new file formats to it in future. Default: `mp3,ogg,aac,aacp,wma,wmv,jpg,jpeg,gif,png,txt,m3u,xspf,pls`

UPLOAD_SIZE_LIMIT

Specifies the maximum file size that will be accepted by the web-based File Manager's file uploader. This is a per-file limit. Be conservative; web-based uploads aren't very robust (on the server nor client side) and tend to have problems with large files. Encourage users to use FTP for very large files (greater than 100MB or so).

Default: `104857600`

FILEMODE_SPOOL

Specifies the default file mode (permissions, in octal notation) for files uploaded via the web interface to the user's `spool/` directory and its subdirectories, including the `media/` directory. (Note that this does not apply to files uploaded via FTP.)

Default: `0660`

FILEMODE_ONDEMAND

Specifies the default file mode (permissions, in octal notation) for files uploaded via the web interface to the user's `spool/ondemand/` directory. (Note that this does not apply to files uploaded via FTP.)

Default: `0664`

4.2.12 AutoDJ

AUTODJ_FAILSAFE

If a stream's autoDJ is set to "Permitted and enabled", and Centova Cast detects that an live DJ has used the "Deactivate source" option to temporarily turn off the autoDJ, but no source is connected, this option will cause Centova Cast to automatically restart the autoDJ. This can be useful in case a live DJ disconnects and forgets to reactivate the autoDJ.

Note that this will have NO EFFECT if the autoDJ is set to "Disabled", and is only checked when the cron job runs so the reactivation may be delayed.

Default: `true`

RANDOM_UNIQUE_PERIOD

Limits identical song playbacks within a given period (in minutes); refer to this article for details.

Default: `360`

ICES_METADATA_FORMAT

Enter the metadata format for use with ices-cc realtime track selection. This will display in the listener's media player, the recent tracks list, and so-on.

Example: `ICES_METADATA_FORMAT='%artist% - %album% (%releaseyear%) - %title% (%bitrate%kbps, %length% seconds)'`

Default: `%artist% - %title%`

4.2.13 Optimization

OPTIMIZE_HTML

If enabled, Centova Cast will optimize its HTML output, removing all unnecessary whitespace, comments, and other content unnecessary for page rendering. Leaving this enabled is strongly recommended for production use as it reduces page load time and bandwidth usage, however if you are modifying Centova Cast's templates and need to use your browser's View Source feature you may want to temporarily disable this for testing.

Default: `true`

4.2.14 Process launching & monitoring

SKIP_PROCESS_CHECK

If enabled, Centova Cast will stop monitoring all server/autoDJ processes. This means if a server/autoDJ crashes/etc. it will remain offline until manually restarted.

Default: `false`

PID_ZOMBIE_DOWN

If a process goes zombie, should it be considered as "down"?

Default: `true`

PID_LOGGING

If enabled, Centova Cast will log detailed process control and monitoring information to its event log to aid in debugging process control problems.

Default: `false`

THROTTLE_RESTART_ATTEMPTS

Configures the maximum number of times Centova Cast will attempt to restart a failed server/source application within a 30-minute period before throttling the restart attempts. Throttling failed restarts reduces server load and avoids mailbombing users/admins with restart notification emails.

Default: `3`

APP_STARTUP_WAIT

Configures the number of microseconds to wait after starting a server/source application before checking to see if it exited unexpectedly.

This should be set high enough to give the application a chance to bail if something is wrong (such as a configuration file error or port conflict) but not so high as to cause unnecessarily long delays in the web interface.

Default: `500000`

4.2.15 Log Processing

LOGPROCESS_MAX_RESUME

At the end of a log processing job, Centova Cast makes note of any listener sessions which appear to still be in progress, and saves them to be reloaded at the beginning of the next log processing job. Due to the way ShoutCast logs its sessions, in some very rare cases the end of a session may never be detected. This setting caps the number of sessions that may be resumed, ensuring that "endless sessions" do not remain endless forever.

Default: `2000`

LOG_SLEEP_INTERVAL

Specifies the number of log lines after which Centova Cast will pause log processing to give the system load a chance to subside.

Log processing can be a CPU- and disk-intensive task, and on heavily-loaded servers you may want to reduce this value to limit the speed at which logs are processed and thereby reduce the system load.

Default: `500000`

LOG_SLEEP_DURATION

Specifies the duration (in microseconds) for which Centova Cast will pause the log processing job.

Log processing can be a CPU- and disk-intensive task, and on heavily-loaded servers you may want to increase this value to limit the speed at which logs are processed and thereby reduce the system load.

Default: `100000`

4.2.16 SMTP options

SMTP_CONNECT_TIMEOUT

Specifies the maximum amount of time (in seconds) to wait for a successful connection to the configured SMTP server. If a connection is not established in this time period, the email is discarded.

Default: `10`

4.2.17 Daemon connectivity

RPC_CONNECT_TIMEOUT

Configures the timeout (in seconds) for the web interface to connect to cc-control. This should be kept as low as possible to avoid unnecessary delays in the event of a cc-control outage.

Default: `10`

RPC_READ_TIMEOUT

Configures how long (in seconds) the web interface should wait for cc-control to process most commands. If cc-control takes longer than this to return a result, the web interface will assume something went wrong and throw a socket error.

Default: `60`

RPC_LONG_READ_TIMEOUT

Configures how long (in seconds) the web interface should wait for cc-control to process disk-intensive commands, such as recursive filesystem operations, which may take awhile to complete. If your disks are slow, or your users have extremely large media libraries, you may need to increase this.

Default: `300`

4.2.18 Compatibility features

FIX_SCTRANS_TITLES

sc_trans v1.x does not support ID3 tags; instead, it displays a mangled version of the MP3's filename as the track title. This results in the mangled filename being displayed in the Recent Tracks lists, the Now Playing widget,and other areas of Centova Cast.

Enabling this option causes Centova Cast to try to determine which song the filename corresponds to, and replace it with the actual artist/track name in the Recent Tracks list, Now Playing widget, and other areas of Centova Cast.

A better solution is to simply not use sc_trans v1.x.

Default: `true`

4.2.19 Application-assigned values

CRYPTO_KEY

Assigns a key for internal cryptography in Centova Cast. This value should be set once at installation and then never changed; typically it is set automatically by the Centova Cast installer. Changing this value may lead to loss of account/configuration data.

Default: (randomly generated at installation time)

USE_WEB_PROXY

Indicates to Centova Cast whether or not the port 80 proxy has been enabled on this server. Do not change this setting directly, as doing so will not enable or disable the proxy in itself. Instead, run:

```
/usr/local/centovacast/sbin/setproxy [on|off]
```

Default: `false`

Chapter 5

Command-line Tools

Centova Cast includes a variety of commandline applications which can be invoked from a terminal session to access advanced functionality in Centova Cast or automate common tasks. These utilities are described in the subsections that follow.

Please note that except where otherwise noted, the commandline tools are provided for the convenience of experienced administrators only, and are not officially supported by Centova Technologies in any capacity.

5.1 Controlling Centova Cast

5.1.1 Init Script

Centova Cast includes an LSB-compliant init script in `/etc/init.d/centovacast` which functions similarly to other standard init scripts, i.e.:

- `/etc/init.d/centovacast start` – starts Centova Cast
- `/etc/init.d/centovacast stop` – stops Centova Cast
- `/etc/init.d/centovacast restart` – restarts Centova Cast
- `/etc/init.d/centovacast reload` – reloads configuration files where possible
- `/etc/init.d/centovacast status` – shows process statuses

This is the approved and recommended way to control Centova Cast. During installation Centova Cast is automatically configured to start when your server boots up.

5.1.2 Advanced Process Control

Centova Cast's init script also provides functionality for controlling individual Centova Cast processes.

- `/etc/init.d/centovacast start-web` – starts the web interface (cc-web)
- `/etc/init.d/centovacast start-app` – starts the application server (cc-app)
- `/etc/init.d/centovacast start-ccd` – starts the control daemon (cc-control)
- `/etc/init.d/centovacast start-ftp` – starts the FTP server (cc-ftpd)
- `/etc/init.d/centovacast start-img` – starts the image daemon (cc-imaged)

Replace `start` with `stop` in each of the above commands to stop the associated process.

5.2 Diagnostic Report Generator

To assist in troubleshooting Centova Cast-related problems, Centova Cast includes a utility which can create a detailed report about your system for review by an administrator or the Centova Technologies staff. Providing such a report to the helpdesk can greatly reduce the time required to diagnose a problem with your server.

There are three ways to invoke the report script:

- **/usr/local/centovacast/sbin/generatereport**
  With no parameters, this script generates a brief report about your overall system status/integrity.
- **/usr/local/centovacast/sbin/generatereport full**
  With the 'full' parameter, this script generates an overall system report, plus includes your core Centova Cast configuration files, Centova Cast master log files, and information about the integrity of your Centova Cast application files and database tables, all of which are invaluable for diagnosing system issues.

  All highly-sensitive information (such as your crypto key) is omitted from the report for your security.
- **/usr/local/centovacast/sbin/generatereport user1 [user2 user3 . . .]**
  When one or more usernames are specified as parameters, this script generates an overall system report, plus detailed information about each of the specified user accounts, including their complete account settings, server/auto DJ configuration files, and log files.

  Highly-sensitive information (such as passwords) is omitted from the report to the best of the script's ability, but the report file should still be kept confidential as it may still contain enough information to be of use to an attacker.

5.3 Fixing Problems

5.3.1 Permissions Problems

Centova Cast includes a script which will completely re-set all file permissions if they are damaged by a systems administrator or another application:

```
/usr/local/centovacast/sbin/fixperms
```

If unusual problems such as “Permission denied” errors are encountered, running this script may in some cases resolve the problem.

5.4 Centova Cast Management Utility

Centova Cast includes a commandline interface to automate many common tasks that would normally be performed via the web interface or via the XML API.

Note that the management utility is provided for the convenience of experienced administrators only, and is not supported by Centova Technologies.

5.4.1 Management Utility Invocation

Basic Invocation

The commandline management utility is invoked using the *ccmanage* utility:

```
/usr/local/centovacast/bin/ccmanage
```

Running `ccmanage` without parameters will provide a list of the most comment options available. See the detailed command reference for a complete list of available commands.

Passwordless Mode

For automation purposes it may be convenient to skip the password prompt presented by *ccmanage* when invoked as root. This capability is provided by the *sumanage* utility:

```
/usr/local/centovacast/sbin/sumanage
```

The *sumanage* utility is identical to *ccmanage* with the exceptions that it can only be invoked by root, and that it operates without prompting for a password.

For security reasons, *sumanage* will not operate until you create a special password file with which it can authenticate against Centova Cast's user database. To create the file, you can run the following commands as root:

```
echo 'admin|adminpassword' > /usr/local/centovacast/etc/.ccshadow
chmod 0600 /usr/local/centovacast/etc/.ccshadow
```

In the first command above, replace *adminpassword* with your actual Centova Cast administrator password. You will need to re-execute the above command every time you change your Centova Cast administrator password.

5.4.2 Output Formats

For ease of integration with other scripts and utilities, the management utility supports a number of output modes to control how data generated by the management utility is returned.

The following output modes are available:

Text Mode

Selected using `--outputmode=text`, this mode generates output as formatted, human-readable text. This is the default output mode if no `--outputmode` parameter is used.

Example output from `ccmanage version --outputmode=text`:

```
web:
|  version: "3.0.0"
|  accounts: "81"
|  other:
|  |  Load (1m):
|  |  |  0: "f"
|  |  |  1: "0.2"
Result: OK Centova Cast v3.0.0
```

BASH script

Selected using `--outputmode=bash`, this mode generates output as a series of variable assignments suitable for use in BASH scripts.

Example output from `ccmanage version --outputmode=bash`:

```
WEB_VERSION="3.0.0"
WEB_ACCOUNTS="81"
WEB_OTHER_LOAD1M_0="f"
WEB_OTHER_LOAD1M_1="0.18"
SUCCESSFUL="1"
MESSAGE="Centova Cast v3.0.0"
```

Example implementation in a BASH script:

```
#!/bin/sh
eval $(/usr/local/centovacast/sbin/sumanage version --outputmode=bash)
[ "$SUCCESSFUL" -eq 0 ] && echo "Error: $MESSAGE" && exit 1
echo "Centova Cast version is $WEB_VERSION"
```

PHP Serialized

Selected using `--outputmode=serialize`, this mode enerates output as a PHP serialized string, usable via PHP's unserialize() function.

Example output from `ccmanage version --outputmode=serialize`:

```
a:3:{s:10:"successful";i:1;s:7:"message";s:21:"Centova Cast v3.0.0";s:4:"data";a:1:{s:3:"web";
    a:3:{s:7:"version";s:7:"3.0.0";s:8:"accounts";s:2:"81";s:5:"other";a:1:{s:9:"Load (1m)";
    a:2:{i:0;s:1:"f";i:1;d:0.12;}}}}}
```

Example implementation in a PHP script:

```
<?php
$output = `/usr/local/centovacast/sbin/sumanage version --outputmode=serialize`;
if (!is_array($result = unserialize($output))) die("Bad output\n");
if (!$result['successful']) die('Error: '.$result['message']."\n");
echo 'Centova Cast version is ' . $result['data']['web']['version']."\n";
```

PHP Statement

Selected using `--outputmode=php`, this mode generates output as a PHP array() statement which may be directly included in a PHP script.

Example output from `ccmanage version --outputmode=php`:

```
array (
    'successful' => 1,
    'message' => 'Centova Cast v3.0.0',
    'data' =>
        array (
            'web' =>
                    array (
                    'version' => '3.0.0',
                    'accounts' => '81',
                    'other' =>
                        array (
                            'Load (1m)' =>
                                array (
                                    0 => 'f',
                                    1 => 0.08,
                                ),
                        ),
                ),
        ),
)
```

Example implementation in PHP a script (not recommended):

```
<?php
$output = `/usr/local/centovacast/sbin/sumanage version --outputmode=php`;
$result = eval('return '.$output.';');
if (!$result['successful']) die('Error: '.$result['message']."\n");
echo 'Centova Cast version is ' . $result['data']['web']['version']."\n";
```

JSON

Selected using `--outputmode=json`, this mode generates output using JavaScript Object Notation (JSON).

Example output from `ccmanage version --outputmode=json`:

```
{
        "successful":1,
        "message":"Centova Cast v3.0.0",
        "data":{
                "web":{
                        "version":"3.0.0",
                        "accounts":81,
                        "other":{
                                "Load (1m)":[
                                        "f",
                                        0.1700
                                ]
                        }
                }
        }
}
```

Example implementation in a Python script:

```
import sys, subprocess, json

p = subprocess.Popen(["/usr/local/centovacast/sbin/sumanage","version","--outputmode=json"], st
output, error = p.communicate()
result = json.loads(output)

if result['successful'] != 1:
    print u'Error:', result['message']
    sys.exit(1)

print u'Centova Cast version is', result['data']['web']['version']
```

XML

Selected using `--outputmode=xml`, this mode generates output in XML format.

Example output from `ccmanage version --outputmode=xml`:

```
<?xml version="1.0" encoding="UTF-8" ?>
<result>
  <successful>1</successful>
  <message>Centova Cast v3.0.0</message>
  <data>
    <web>
      <version>3.0.0</version>
      <accounts>81</accounts>
```

```
      <other>
        <Load1m>
          <node>f</node>
          <node>0.14</node>
        </Load1m>
      </other>
    </web>
  </data>
</result>
```

Example implementation in a Ruby script:

```
require 'rexml/document'

output = `/usr/local/centovacast/sbin/sumanage version --outputmode=xml`
doc = REXML::Document.new(output)

if doc.root.elements["successful"].text != "1"
        puts "Error: " + doc.root.elements["message"].text
        exit 1
end

puts "Centova Cast version is " + doc.root.elements["data/web/version"].text
```

CSV Spreadsheet

Selected using `--outputmode=csv`, this mode generates output in CSV (comma-separated value) format. Note that due to the limitations of this format, it is only usable with commands that return a flat list of values; attempting to CSV mode with commands that return nested data structures will result in an error message.

Example output from `ccmanage usage all --outputmode=csv`:

```
username,diskquota,transferlimit,diskusage,transferusage
rockstream,5000,100000,3875,75325
popstream,7500,150000,4925,107350
~result,OK,"Check complete"
```

Example of generating a plain-English report from the CSV data using the `awk` utility:

```
/usr/local/centovacast/sbin/sumanage usage all --outputmode=csv \
    | awk -F ',' 'NR>1 && $1 !~ /^~/ {print "User " $1 " has used "
        $4 "/" $2 "MB of disk space and " $5 "/" $3 "MB of data transfer."}'
```

5.5 Reinstalling Centova Cast

During testing, it may be useful to perform frequent, automated reinstallations of Centova Cast to provide a "clean" environment for further tests.

The following script is an example of how this process can be completely automated and run in an unattended manner. Note that it requires an initial, functioning installation of Centova Cast in order to operate.

```
#!/usr/bin/env bash
# pull in database configuration values
. /usr/local/centovacast/etc/centovacast.conf

# uninstall
echo 'UNINSTALL' | /usr/local/centovacast/sbin/uninstall --i-want-to-delete-all-my-data

# clear the database
echo "DROP DATABASE $DB_NAME; CREATE DATABASE $DB_NAME;" \
    | mysql -h$DB_HOST -u$DB_NAME -p$DB_PASS

# clean up any remaining data from the old installation (but just in case)
rm -rf /usr/local/centovacast /etc/centovacast.conf

# automate the reinstallation; tweak the parameters to your tastes
./install.sh --icecast-all --channel=stable --admin-email=example@example.com \
    --admin-pass=example --dbname=$DB_NAME --dbuser=$DB_USER --dbpass=$DB_PASS \
    --dbhost=$DB_HOST
```

After saving this to your server as `reinstall.sh` and making it executable using `chmod a+x reinstall.sh`, simply place it in the same directory as your original `install.sh` file (from your initial Centova Cast installation) and invoke it as needed to perform a fully automated reinstallation.

5.6 Uninstalling Centova Cast

If for some reason you need to remove Centova Cast from your server, you may use the `uininstall` utility to perform the uninstallation:

```
/usr/local/centovacast/sbin/uninstall --i-want-to-delete-all-my-data
```

Note that you *must* pass the `--i-want-to-delete-all-my-data` parameter to confirm that you really do want to delete your entire Centova Cast installation and all of your client data.

Further, before performing the uninstallation, the uninstaller will prompt you to type the word `UNINSTALL` in all caps to verify that you did indeed mean to invoke the uninstaller.

5.7 Update Utility

Centova Cast includes a fully automated update utility to upgrade Centova Cast when a full update or new components are available.

5.7.1 Basic Invocation

The basic invocation for the update utility is:

```
/usr/local/centovacast/sbin/update
```

This performs a complete update, and is documented in the Upgrading Version 3 section of the Installation Manual.

5.7.2 Updating Individual Components

If for some reason you only want to update some particular component of Centova Cast instead of performing a complete upgrade, you can pass an application identifier as the first parameter to the update script. For example:

```
/usr/local/centovacast/sbin/update web
```

The above would update the Centova Cast web server only. If you need a list of all available, upgradeable application identifiers on your server, you can use this command:

```
for f in /usr/local/centovacast/etc/update.d/*.update; do
    . $f; echo "${DATADIR/cc-/} - $TITLE"
done
```

5.7.3 Forcing an Update

Centova Cast automatically detects whether a newer version of each package is available, and will not attempt to update the package if a newer version is not available.

If you wish to force an update (for example, if you suspect the files from the package have been corrupted on your server), you may pass the `--force` parameter to force the installation, eg:

```
/usr/local/centovacast/sbin/update --force
```

This can also be used with individual components as explained in the previous section.

5.7.4 Adding New Components

The update utility can also be used to install new packages into Centova Cast, using the –add parameter.

This is documented in detail in the Adding Additional Software section of the Installation Manual.

5.7.5 Performing Custom Actions on Update

You may wish to have Centova Cast run a custom script before or after an update, for example to preserve and restore modifications you've made to the Centova Cast theme.

Before performing an update, Centova Cast checks for the existence of a `preupdate` shell script in:

```
/usr/local/centovacast/sbin/preupdate
```

If it exists and is executable, Centova Cast will invoke it prior to performing the update.

Similarly, after performing an update, Centova Cast checks for the existence of a `postupdate` shell script in:

```
/usr/local/centovacast/sbin/postupdate
```

If it exists and is executable, Centova Cast will invoke it after the update.

Both `preupdate` and `postupdate` are standard shell scripts; they should begin with a shebang line (eg: `#!/bin/bash` or similar) and may be written in any language for which an interpreter is installed on your server.

The following simple example would preserve any modifications you've made to your Centova Cast logo on the login page:

preupdate:

```
#!/bin/sh
mkdir -p /usr/local/centovacast/var/preupdate_backup
cp -f /usr/local/centovacast/web/theme/images/login-logo.png \
    /usr/local/centovacast/var/preupdate_backup/
```

postupdate:

```
#!/bin/sh
cp -f /usr/local/centovacast/var/preupdate_backup/login-logo.png \
    /usr/local/centovacast/web/theme/images/
rm -rf /usr/local/centovacast/var/preupdate_backup
```

5.8 Init Script

In some cases it may be desirable to run custom commands at the beginning of the Centova Cast init script (`/etc/init.d/centovacast`), for example to set custom resource limits (via ulimit or similar) or to perform custom tasks at startup or shutdown.

Such commands may be added to `/usr/local/centovacast/etc/init.local` and will be automatically included at the beginning of the Centova Cast init script.

Note that because `init.local` is sourced at the very beginning of the init script, its commands will be invoked regardless of whether Centova Cast is being signaled to start, stop, restart, or perform some other valid or invalid action.

5.9 Menu Customizations

The menus displayed in the client, reseller, and administration areas are dynamically generated depending on the permissions of the logged-in user, and thus are not found in any of Centova Cast's template files.

The menus can still be customized, however, by directly editing the menu definition file. This file is located in:

```
/usr/local/centovacast/web/menu.php
```

The format of this file is a series of definitions of multidimensional PHP arrays. Accordingly, some PHP experience may be necessary to effectively customize this file without damaging it.

5.9.1 Menu Definitions

The general format of the menu definitions is:

```
$menu_admin = array(
    // menu sections for the administration area
);

$menu_reseller = array(
    // menu sections for the reseller area
);

$menu_client = array(
    // menu section for the client area
);
```

5.9.2 Menu Sections

Each menu section within the $menu_xxx arrays above is defined as:

```
array(
    'name'=>__('Title'),        // the section title to display in the menu
    'url'=>'index.php?foo=bar', // the URL to launch when the section heading is clicked
    'icon'=>'name',             // the icon to display next to the section title;
                                // corresponds to /theme/images/nav/<icon>.png
    'condition'=>'...',         // conditions required to display this section (see below)
    'djpermission'=>'...',      // DJ permissions required for this section (see below)

    'items'=>array(
    // menu items for this section
    )
),
```

5.9.3 Menu Items

Each menu item within the 'items' arrays in the menu section definition is defined as:

```
array(
    'name'=>__('Item name'),      // the name to display for the menu item
    'title'=>__('Tip text'),      // a tip to display on mouse hover
    'url'=>'index.php?page=foo',  // the URL to launch when the item is clicked
    'confirm'=>__('Prompt text'), // an optional confirmation message to display when clicked
    'target'=>'...',              // the HTML target="xxx" attribute for the link
                                  // eg: use '_blank' to open in a new window
    'condition'=>'...',           // conditions required to display this item (see below)
    'djpermission'=>'...'         // DJ permissions required for this item (see below)
),
```

5.9.4 Conditions

Some menu items are only relevant if the stream is in a particular state; for example, the "start server" link would only be relevant when the server is not currently running.

Conditions allow each menu item to be dynamically displayed or hidden based on various conditions and states. Each menu section and item supports a `condition` option which specifies a list of condition(s) that must be met for the section or item to be displayed.

The format of the `condition` list is one or more conditions separated by commas, eg:

```
'condition'=>'foo'          // condition "foo" must be true
'condition'=>'foo,bar,baz'  // conditions "foo", "bar", and "baz" must be true
'condition'=>'foo,!bar,baz' // conditions "foo" and "baz" must be true, but "bar" must NOT
'condition'=>''             // no conditions; always display the item
```

If the `condition` option is empty or omitted entirely, the menu section or item is always displayed. If one of the conditions in the `condition` option is preceded by an exclamation mark (!) it is negated; the item will only be displayed if that condition is NOT true.

The following conditions are available:

Client Area

- `usesource` - true if the account is configured to support an autoDJ
- `autodjup` - true if the autoDJ is up
- `autodjdown` - true if the autoDJ is down
- `serverup` - true if the streaming server is up
- `canstartautodj` - true if all other conditions allow the autoDJ to be started
- `privileged` - true if the client is logged in from a reseller or admin account
- `serverreload` - true if the streaming server supports reloading its configuration without a full restart

Reseller Area

- No conditions are currently supported.

Administrator Area

- No conditions are currently supported.

5.9.5 DJ Permissions

When a DJ account logs in to Centova Cast, the standard client menu definition is used to generate the navigation menu for the DJ. Most DJ accounts will have permissions restrictions, however, which prevent the DJ from using many of the features displayed in the menu.

To avoid displaying menu sections or items that the DJ cannot actually use, each menu section and item supports a `djpermission` option which specifies a list of permission(s) that the DJ must possess in order to see the menu item or section.

The format of the `djpermission` list is zero or more permissions separated by plus (+) character (indicating "AND") or pipe (|) character (indicating "OR"), eg:

```
'djpermission'=>'foo'              // require the "foo" permission
'djpermission'=>'foo+bar+baz'      // require the "foo", "bar", and "baz" permissions
'djpermission'=>'foo|bar|baz'      // require any ONE of the "foo" OR "bar" OR "baz" permission
'djpermission'=>''                 // no permissions required; always display the item
```

If the `djpermission` option is empty or is omitted entirely, the DJ is always permitted to see the menu section or item. Note that pipe and plus characters cannot be combined within a single `djpermission` list.

The following DJ permissions are available:

- `controlserver` - DJ must have the *Start/stop the stream* permission
- `controlautodj` - DJ must have the *Start/stop the autoDJ* permission
- `manageplaylists` - DJ must have the *Manage playlist settings* permission
- `medialibrary` - DJ must have the *Access media library* permission
- `managefiles` - DJ must have the *Manage media files* permission
- `viewstatistics` - DJ must have the *View statistics* permission
- `viewlisteners` - DJ must have the *View listeners* permission
- `viewlogs` - DJ must have the *View logs* permission
- `denied` - a special permission which indicates that the DJ should *never* see this menu item/section

It is important to note that this is *not* an access control mechanism in itself; this only controls which menu items are displayed to the DJ. If a menu item includes an incorrect `djpermission` setting which allows the DJ to see a menu item to which he does not have access, the DJ will still receive a *permission denied* error when he clicks the link. Similarly, the `djpermission` setting cannot be used to deny access to a page to which the DJ has been given access; while it may be used to hide the link, the DJ will still have access to the page if he happens to know the URL and enters it manually.

5.10 Wrapping Server Applications

On occasion, a server administrator may wish to configure Centova Cast to run a “wrapper” application when launching the streaming server, instead of directly launching SHOUTcast DNAS or IceCast. This may be useful when running certain proxy applications, advertising software, or similar, which needs to be stopped and started in tandem with the server software.

In most cases, Centova Cast’s event scripts system will suffice for launching such applications, however in certain cases the application in question may expect to launch DNAS or IceCast itself. In these cases, a wrapper script such as below may be used to “proxy” all signals and PID management between the third-party application and Centova Cast.

5.10.1 A Sample Wrapper Script

This wrapper script serves as an example of how to force Centova Cast to control an application other than SHOUTcast DNAS (or IceCast) when performing start, stop, and reload actions from within Centova Cast.

Note that this script is provided as a courtesy only, and (as with any customization to Centova Cast) is unsupported by Centova Technologies.

```
#!/usr/bin/env bash
# ============================================================================
# Centova Cast - Copyright 2014, Centova Technologies
# Skeleton server application wrapper script
# ============================================================================
#
# This is a stub script into which you can add your code to control your
# third-party application.
#

# This function is invoked when the streaming server needs to be started;
# replace the commands below with whatever commands are needed to start
# your third-party application.
function start_server {

    # put your commands to start your application here; if you need
    # the account's username, it's available as $USERNAME

    # ...

    # set RESULT to 0 if the application started successfully, or 1 if
    # it failed to start
    RESULT=$?

    return $RESULT
}
```

```
# This function is invoked when the streaming server needs to be stopped;
# replace the commands below with whatever commands are needed to stop
# your third-party application.
function stop_server {

    # put your commands to stop your application here

    return 0
}

# This function is invoked when the streaming server needs to close and
# reopen its log files; replace the commands below with whatever commands are
# needed to reload your third-party application.
function reload_server {

    # put your commands to reload your application here

    return 0
}

# This function is invoked to determine whether the streaming server is
# still running; replace the commands below with whatever commands are
# needed to check whether your third-party application is still
# running.
#
# Note that this is invoked every 2 seconds while monitoring the process,
# so for performance reasons it should NOT perform any heavy processing.
function check_server_running {

    # put your commands to check your third-party application state
    # here

    # set RUNNING to 1 if the application is still running, or 0 if
    # it is no longer running
    RUNNING=0

    return $RUNNING
}

# Everything below is wrapper code and can be left alone.

function stop_and_exit {
    stop_server
    exit 0
}

# trap signals
```

```
trap stop_and_exit INT TERM EXIT
trap reload_server HUP

while [ ! -z "$1" ]; do
    U=$1
    shift
done
U=${U#*\/}
USERNAME=${U%%\/*}

start_server
[ $? -gt 0 ] && exit $?

while [ true ]; do
    check_server_running
    [ $? -gt 0 ] && break
    sleep 2
done

exit 0
```

Note that this script will NOT do anything useful on its own. To make it work for your particular scenario, you MUST have at least some familiarity with bash scripting to make the changes noted in the script comments.

5.10.2 Implementing the Wrapper Script

To implement a wrapper script on your server, save the example script above as (for example) `/usr/local/centovacast/bin/wrapper.sh`. (Technically you can use any path or name, as long as filesystem permissions are observed.)

Next, make any changes that are necessary to launch your application correctly and obtain the PID of the streaming server.

Next, run:

```
chmod 0775 /usr/local/centovacast/bin/wrapper.sh
```

Finally, edit `/usr/local/centovacast/etc/cc-control.conf` and replace the path to SHOUTcast DNAS (or IceCast) with the path to the wrapper script. For example, if you want the wrapper script to be executed in place of SHOUTcast DNAS v2, find the line that begins with `SHOUTCAST2_BIN=` and replace it with:

```
SHOUTCAST2_BIN=/usr/local/centovacast/bin/wrapper.sh
```

This change takes effect immediately, and the next time any SHOUTcast DNAS v2 streams are stopped, started, or reloaded, the action will be performed through the wrapper script.

5.10.3 Practical Example: Controlling Apache

This script implements a simple wrapper which lets Centova Cast control the Apache web server instead of SHOUTcast DNAS. In more general terms, it's an example of wrapping an application that is controlled via an init script, or another short-lived control process.

```
#!/usr/bin/env bash
# ============================================================================
# Centova Cast - Copyright 2014, Centova Technologies
# Example application wrapper script to control Apache
# ============================================================================
#
# This script shows how to implement a wrapper for a third-party application
# that is controlled via an init script (or any other process management tool
# that takes separate start and stop commands).
#
# This (mostly useless) example demonstrates how to start or stop the Apache
# webserver when a user starts or stops his stream.
#

# This function is invoked when the streaming server needs to be started;
# replace the commands below with whatever commands are needed to start
# your third-party application.
function start_server {

    # start Apache via its init script; the 'ccuser' account would of
    # course need to be in /etc/sudoers for this to work
    sudo /etc/init.d/apache2 start

    # set RESULT to 0 if the application started successfully, or 1 if
    # it failed to start
    RESULT=$?

    return $RESULT
}

# This function is invoked when the streaming server needs to be stopped;
# replace the commands below with whatever commands are needed to stop
# your third-party application.
function stop_server {

    # stop Apache via its init script
    sudo /etc/init.d/apache2 stop

}

# This function is invoked when the streaming server needs to close and
# reopen its log files; replace the commands below with whatever commands are
```

```
# needed to reload your third-party application.
function reload_server {

    # reload Apache via its init script
    sudo /etc/init.d/apache2 force-reload

}

# This function is invoked to determine whether the streaming server is
# still running; replace the commands below with whatever commands are
# needed to check whether your third-party application is still
# running.
#
# Note that this is invoked every 2 seconds while monitoring the process,
# so for performance reasons it should NOT perform any heavy processing.
function check_server_running {

    # check Apache's status via its init script; the init script will
    # return an exit code of 0 if it's running, or 1 if it's not
    sudo /etc/init.d/apache2 status

    if [ $? -eq 0 ]; then
     RUNNING=1
    else
     RUNNING=0
    fi

    return $RUNNING
}

# Everything below is wrapper code and can be left alone.

function stop_and_exit {
    stop_server
    exit 0
}

# trap signals
trap stop_and_exit INT TERM EXIT
trap reload_server HUP

while [ ! -z "$1" ]; do
    U=$1
    shift
done
U=${U#*\/}
USERNAME=${U%%\/*}
```

```
start_server
[ $? -gt 0 ] && exit $?

while [ true ]; do
    check_server_running
    [ $? -gt 0 ] && break
    sleep 2
done

exit 0
```

5.10.4 Practical Example: Indirectly Controlling DNAS2

This script implements yet another simple wrapper which lets Centova Cast indirectly manage a SHOUTcast DNAS v2 process. In more general terms, it's an example of wrapping a long-lived process from an application which daemonizes and runs in the background after starting up.

```
#!/usr/bin/env bash
# ============================================================================
# Centova Cast - Copyright 2014, Centova Technologies
# Example application wrapper script to indirectly control DNAS2
# ============================================================================
#
# This script shows how to implement a wrapper for a third-party application
# that daemonizes itself and executes the streaming server as a subprocess
# which it then manages.
#
# This (mostly useless) example demonstrates running SHOUTcast DNAS v2
# as if it were in itself a third-party application.
#

# This function is invoked when the streaming server needs to be started;
# replace the commands below with whatever commands are needed to start
# your third-party application.
function start_server {

    # start DNAS v2 by changing to the virtual host directory and launching
    # the sc_serv process
    VHOSTDIR=/usr/local/centovacast/var/vhosts/$USERNAME
    [ ! -d $VHOSTDIR ] && echo "$VHOSTDIR does not exist" && return 1
    cd $VHOSTDIR
    /usr/local/centovacast/shoutcast22/sc_serv etc/server.conf

    # since, in this example, our process is going to continue running in the
    # background, we need to make note of its PID here so we can control
    # it later
    SERVERPID=$!
```

```
    # set RESULT to 0 if the application started successfully, or 1 if
    # it failed to start
    RESULT=$?

    return $RESULT
}

# This function is invoked when the streaming server needs to be stopped;
# replace the commands below with whatever commands are needed to stop
# your third-party application.
function stop_server {

    # we made note of the DNAS2 PID in start_server, and can now simply
    # send SIGTERM to that PID
    [ ! -z "$SERVERPID" -a -d /proc/$SERVERPID ] && kill $SERVERPID

}

# This function is invoked when the streaming server needs to close and
# reopen its log files; replace the commands below with whatever commands are
# needed to reload your third-party application.
function reload_server {

    # we made note of the DNAS2 PID in start_server, and can now simply
    # send SIGHUP to that PID
    kill -HUP $SERVERPID

}

# This function is invoked to determine whether the streaming server is
# still running; replace the commands below with whatever commands are
# needed to check whether your third-party application is still
# running.
#
# Note that this is invoked every 2 seconds while monitoring the process,
# so for performance reasons it should NOT perform any heavy processing.
function check_server_running {

    # we made note of the DNAS2 PID in start_server, and every active PID
    # under Linux is found under /proc/<pid>, so we can simply test for
    # its existence to determine whether DNAS2 is still running
    if [ -d /proc/$SERVERPID ]; then
    RUNNING=1
    else
    RUNNING=0
    fi
```

```
    return $RUNNING
}

# Everything below is wrapper code and can be left alone.

function stop_and_exit {
    stop_server
    exit 0
}

# trap signals
trap stop_and_exit INT TERM EXIT
trap reload_server HUP

while [ ! -z "$1" ]; do
    U=$1
    shift
done
U=${U#*\/}
USERNAME=${U%%\/*}

start_server
[ $? -gt 0 ] && exit $?

while [ true ]; do
    check_server_running
    [ $? -gt 0 ] && break
    sleep 2
done

exit 0
```

5.10.5 Further development

These examples demonstrate a few common scenarios for third-party application management, however these are not by any means the only ways you can manage application processes. You could, for example, use `curl` or `wget` to control applications managed via a SOAP API.

If you're having trouble with a particular implementation, consult any systems administrator who is experienced with bash shell scripting for assistance.

5.11 Event Script Reference

The following events are available.

5.11.1 Event: playlist_advanced

Description

Called every time a track is selected from a playlist for the autoDJ.

Note that is imperative that your script work *very* quickly. During the execution of your script, the autoDJ is forced to wait to receive the information for the next track. If your script takes too long, the current song may end before the autoDJ knows which track to play next, and as a result, there may be silence on the stream and/or the server may believe that the source has died due to inactivity. You may wish to design your script to fork, exit, and and continue operation in the background if processing will take more than a few hundred milliseconds.

Parameters

The following parameters are passed, in the order shown, to this script:

- **username** `(string)`
  the username of the account
- **pathname** `(string)`
  the pathname of the track to be played
- **artist** `(string)`
  the artist of the track to be played
- **album** `(string)`
  the album of the track to be played
- **title** `(string)`
  the title of the track to be played
- **length** `(int)`
  the length of the track to be played, in seconds
- **royaltycode** `(string)`
  the royalty reporting code of the track to be played

Return Value

Ignored.

Sample Code

```
#!/usr/bin/env bash
# playlist_advanced
USERNAME="$playlist_advanced"
PATHNAME="$playlist_advanced"
ARTIST="$playlist_advanced"
ALBUM="$playlist_advanced"
TITLE="$playlist_advanced"
LENGTH="$playlist_advanced"
ROYALTYCODE="$playlist_advanced"

# implementation details here ...
```

5.11.2 Event: pre-create-reseller

Description

Called just before a reseller account is created.

Parameters

The following parameters are passed, in the order shown, to this script:

- **username** `(string)`
  the username of the account
- **password** `(string)`
  the password for the account
- **email** `(string)`
  the email address for the account
- **maxclients** `(int)`
  the listener limit for the account
- **maxbitrate** `(int)`
  the maximum bit rate for the account
- **transferlimit** `(int)`
  the data transfer limit for the account
- **diskquota** `(int)`
  the disk quota for the account
- **usesource** `(int)`
  1 if the reseller can create autoDJ-enabled accounts, otherwise 0
- **mountlimit** `(int)`
  the mount point limit for the account

Return Value

The last line of output from the script determines whether or not the associated event will continue.

If the last line of output contains only the number `1`, the event will proceed normally.

Any other value prevents the account from being created.

Sample Code

```
#!/usr/bin/env bash
# pre-create-reseller
USERNAME="$pre-create-reseller"
PASSWORD="$pre-create-reseller"
EMAIL="$pre-create-reseller"
MAXCLIENTS="$pre-create-reseller"
MAXBITRATE="$pre-create-reseller"
TRANSFERLIMIT="$pre-create-reseller"
DISKQUOTA="$pre-create-reseller"
USESOURCE="$pre-create-reseller"
MOUNTLIMIT="$pre-create-reseller"
ERROR=""

# implementation details here ...

# if you want to force Centova Cast to abort the event, your script should
# set ERROR to an error message string
[ -z "$ERROR" ] && echo "1" || echo "$ERROR"
```

5.11.3 Event: pre-create-account

Description

Called just before a streaming account is created.

Parameters

The following parameters are passed, in the order shown, to this script:

- **username** (`string`)
  the username of the account
- **password** (`string`)
  the password for the account
- **email** (`string`)
  the email address for the account

- **ipaddress** (string)
  the IP address for the account
- **port** (int)
  the port number for the account
- **maxclients** (int)
  the listener limit for the account
- **maxbitrate** (int)
  the maximum bit rate for the account
- **transferlimit** (int)
  the data transfer limit for the account
- **diskquota** (int)
  the disk quota for the account
- **usesource** (int)
  1 if the autoDJ is enabled, otherwise 0
- **mountlimit** (int)
  the mount point limit for the account

Return Value

The last line of output from the script determines whether or not the associated event will continue.

If the last line of output contains only the number 1, the event will proceed normally.

Any other value prevents the account from being created.

Sample Code

```
#!/usr/bin/env bash
# pre-create-account
USERNAME="$pre-create-account"
PASSWORD="$pre-create-account"
EMAIL="$pre-create-account"
IPADDRESS="$pre-create-account"
PORT="$pre-create-account"
MAXCLIENTS="$pre-create-account"
MAXBITRATE="$pre-create-account"
TRANSFERLIMIT="$pre-create-account"
DISKQUOTA="$pre-create-account"
USESOURCE="$pre-create-account"
MOUNTLIMIT="$pre-create-account"
ERROR=""

# implementation details here ...
```

```
# if you want to force Centova Cast to abort the event, your script should
# set ERROR to an error message string
[ -z "$ERROR" ] && echo "1" || echo "$ERROR"
```

5.11.4 Event: post-create-reseller

Description

Called just after a reseller account is created.

Parameters

The following parameters are passed, in the order shown, to this script:

- **username** (string)
  the username of the account
- **password** (string)
  the password for the account
- **email** (string)
  the email address for the account
- **maxclients** (int)
  the listener limit for the account
- **maxbitrate** (int)
  the maximum bit rate for the account
- **transferlimit** (int)
  the data transfer limit for the account
- **diskquota** (int)
  the disk quota for the account
- **usesource** (int)
  1 if the reseller can create autoDJ-enabled accounts, otherwise 0
- **mountlimit** (int)
  the mount point limit for the account

Return Value

Ignored.

Sample Code

```
#!/usr/bin/env bash
# post-create-reseller
USERNAME="$post-create-reseller"
PASSWORD="$post-create-reseller"
EMAIL="$post-create-reseller"
MAXCLIENTS="$post-create-reseller"
MAXBITRATE="$post-create-reseller"
TRANSFERLIMIT="$post-create-reseller"
DISKQUOTA="$post-create-reseller"
USESOURCE="$post-create-reseller"
MOUNTLIMIT="$post-create-reseller"

# implementation details here ...
```

5.11.5 Event: post-create-account

Description

Called just after a streaming account is created.

Parameters

The following parameters are passed, in the order shown, to this script:

- **username** `(string)`
  the username of the account
- **password** `(string)`
  the password for the account
- **email** `(string)`
  the email address for the account
- **ipaddress** `(string)`
  the IP address for the account
- **port** `(int)`
  the port number for the account
- **maxclients** `(int)`
  the listener limit for the account
- **maxbitrate** `(int)`
  the maximum bit rate for the account
- **transferlimit** `(int)`
  the data transfer limit for the account

- **diskquota** `(int)`
  the disk quota for the account
- **usesource** `(int)`
  1 if the autoDJ is enabled, otherwise 0
- **mountlimit** `(int)`
  the mount point limit for the account

Return Value

Ignored.

Sample Code

```
#!/usr/bin/env bash
# post-create-account
USERNAME="$post-create-account"
PASSWORD="$post-create-account"
EMAIL="$post-create-account"
IPADDRESS="$post-create-account"
PORT="$post-create-account"
MAXCLIENTS="$post-create-account"
MAXBITRATE="$post-create-account"
TRANSFERLIMIT="$post-create-account"
DISKQUOTA="$post-create-account"
USESOURCE="$post-create-account"
MOUNTLIMIT="$post-create-account"

# implementation details here ...
```

5.11.6 Event: pre-terminate-account

Description

Called just before a streaming account is terminated.

Parameters

The following parameters are passed, in the order shown, to this script:

- **username** `(string)`
  the username of the account

Return Value

The last line of output from the script determines whether or not the associated event will continue.

If the last line of output contains only the number `1`, the event will proceed normally.

Any other value prevents the account from being terminated.

Sample Code

```
#!/usr/bin/env bash
# pre-terminate-account
USERNAME="$pre-terminate-account"
ERROR=""

# implementation details here ...

# if you want to force Centova Cast to abort the event, your script should
# set ERROR to an error message string
[ -z "$ERROR" ] && echo "1" || echo "$ERROR"
```

5.11.7 Event: pre-terminate-reseller

Description

Called just before a reseller account is terminated.

Parameters

The following parameters are passed, in the order shown, to this script:

- **username** (`string`)
  the username of the account

Return Value

The last line of output from the script determines whether or not the associated event will continue.

If the last line of output contains only the number `1`, the event will proceed normally.

Any other value prevents the account from being terminated.

Sample Code

```
#!/usr/bin/env bash
# pre-terminate-reseller
USERNAME="$pre-terminate-reseller"
ERROR=""
```

```
# implementation details here ...

# if you want to force Centova Cast to abort the event, your script should
# set ERROR to an error message string
[ -z "$ERROR" ] && echo "1" || echo "$ERROR"
```

5.11.8 Event: post-terminate-account

Description

Called just after a streaming account is terminated

Parameters

The following parameters are passed, in the order shown, to this script:

- **username** (string)
  the username of the account

Return Value

Ignored.

Sample Code

```
#!/usr/bin/env bash
# post-terminate-account
USERNAME="$post-terminate-account"

# implementation details here ...
```

5.11.9 Event: post-terminate-reseller

Description

Called just after a reseller account is terminated

Parameters

The following parameters are passed, in the order shown, to this script:

- **username** (string)
  the username of the account

Return Value

Ignored.

Sample Code

```
#!/usr/bin/env bash
# post-terminate-reseller
USERNAME="$post-terminate-reseller"

# implementation details here ...
```

5.11.10 Event: pre-reparent-account

Description

Called just before a streaming account is moved to a new reseller

Parameters

The following parameters are passed, in the order shown, to this script:

- **username** (`string`)
  the username of the account
- **newreseller** (`string`)
  the username of the new reseller account to own the streaming account

Return Value

The last line of output from the script determines whether or not the associated event will continue.

If the last line of output contains only the number `1`, the event will proceed normally.

Any other value prevents the streaming server from being reparented.

Sample Code

```
#!/usr/bin/env bash
# pre-reparent-account
USERNAME="$pre-reparent-account"
NEWRESELLER="$pre-reparent-account"
ERROR=""

# implementation details here ...

# if you want to force Centova Cast to abort the event, your script should
```

```
# set ERROR to an error message string
[ -z "$ERROR" ] && echo "1" || echo "$ERROR"
```

5.11.11 Event: post-reparent-account

Description

Called just after a streaming account is moved to a new reseller

Parameters

The following parameters are passed, in the order shown, to this script:

- **username** (string)
  the username of the account
- **newreseller** (string)
  the username of the new reseller account to own the streaming account

Return Value

Ignored.

Sample Code

```
#!/usr/bin/env bash
# post-reparent-account
USERNAME="$post-reparent-account"
NEWRESELLER="$post-reparent-account"

# implementation details here ...
```

5.11.12 Event: pre-account-status

Description

Called just before a streaming account's status (enabled/disabled) is changed

Parameters

The following parameters are passed, in the order shown, to this script:

- **username** (string)
  the username of the account
- **status** (string)
  the new status for the account

Return Value

The last line of output from the script determines whether or not the associated event will continue.

If the last line of output contains only the number `1`, the event will proceed normally.

Any other value prevents the account's status from changing.

Sample Code

```
#!/usr/bin/env bash
# pre-account-status
USERNAME="$pre-account-status"
STATUS="$pre-account-status"
ERROR=""

# implementation details here ...

# if you want to force Centova Cast to abort the event, your script should
# set ERROR to an error message string
[ -z "$ERROR" ] && echo "1" || echo "$ERROR"
```

5.11.13 Event: post-account-status

Description

Called just after a streaming account's status (enabled/disabled) is changed

Parameters

The following parameters are passed, in the order shown, to this script:

- **username** `(string)`
  the username of the account
- **status** `(string)`
  the new status for the account

Return Value

Ignored.

Sample Code

```
#!/usr/bin/env bash
# post-account-status
USERNAME="$post-account-status"
```

```
STATUS="$post-account-status"

# implementation details here ...
```

5.11.14 Event: pre-start-server

Description

Called just before a streaming server is started.

Parameters

The following parameters are passed, in the order shown, to this script:

- **username** (`string`)
  the username of the account

Return Value

The last line of output from the script determines whether or not the associated event will continue.

If the last line of output contains only the number `1`, the event will proceed normally.

Any other value prevents the streaming server from starting.

Sample Code

```
#!/usr/bin/env bash
# pre-start-server
USERNAME="$pre-start-server"
ERROR=""

# implementation details here ...

# if you want to force Centova Cast to abort the event, your script should
# set ERROR to an error message string
[ -z "$ERROR" ] && echo "1" || echo "$ERROR"
```

5.11.15 Event: post-start-server

Description

Called just after a streaming server is started.

Parameters

The following parameters are passed, in the order shown, to this script:

- **username** (`string`)
  the username of the account

Return Value

Ignored.

Sample Code

```
#!/usr/bin/env bash
# post-start-server
USERNAME="$post-start-server"

# implementation details here ...
```

5.11.16 Event: pre-start-source

Description

Called just before an autoDJ is started.

Parameters

The following parameters are passed, in the order shown, to this script:

- **username** (`string`)
  the username of the account

Return Value

The last line of output from the script determines whether or not the associated event will continue.

If the last line of output contains only the number `1`, the event will proceed normally.

Any other value prevents the autoDJ from starting.

Sample Code

```
#!/usr/bin/env bash
# pre-start-source
USERNAME="$pre-start-source"
ERROR=""
```

```
# implementation details here ...

# if you want to force Centova Cast to abort the event, your script should
# set ERROR to an error message string
[ -z "$ERROR" ] && echo "1" || echo "$ERROR"
```

5.11.17 Event: pre-start-app

Description

Called just before a supplemental application is started.

Parameters

The following parameters are passed, in the order shown, to this script:

- **type** (`string`)
  the application type
- **username** (`string`)
  the username of the account

Return Value

The last line of output from the script determines whether or not the associated event will continue.

If the last line of output contains only the number `1`, the event will proceed normally.

Any other value prevents the application from starting.

Sample Code

```
#!/usr/bin/env bash
# pre-start-app
TYPE="$pre-start-app"
USERNAME="$pre-start-app"
ERROR=""

# implementation details here ...

# if you want to force Centova Cast to abort the event, your script should
# set ERROR to an error message string
[ -z "$ERROR" ] && echo "1" || echo "$ERROR"
```

5.11.18 Event: post-start-source

Description

Called just after an autoDJ is started.

Parameters

The following parameters are passed, in the order shown, to this script:

- **username** (`string`)
  the username of the account

Return Value

Ignored.

Sample Code

```
#!/usr/bin/env bash
# post-start-source
USERNAME="$post-start-source"

# implementation details here ...
```

5.11.19 Event: post-start-app

Description

Called just after a supplemental application is started.

Parameters

The following parameters are passed, in the order shown, to this script:

- **type** (`string`)
  the application type
- **username** (`string`)
  the username of the account

Return Value

Ignored.

Sample Code

```
#!/usr/bin/env bash
# post-start-app
TYPE="$post-start-app"
USERNAME="$post-start-app"

# implementation details here ...
```

5.11.20 Event: pre-reload

Description

Called just before a streaming server is reloaded.

Parameters

The following parameters are passed, in the order shown, to this script:

- **username** (`string`)
  the username of the account

Return Value

The last line of output from the script determines whether or not the associated event will continue.

If the last line of output contains only the number `1`, the event will proceed normally.

Any other value prevents the streaming server from reloading.

Sample Code

```
#!/usr/bin/env bash
# pre-reload
USERNAME="$pre-reload"
ERROR=""

# implementation details here ...

# if you want to force Centova Cast to abort the event, your script should
# set ERROR to an error message string
[ -z "$ERROR" ] && echo "1" || echo "$ERROR"
```

5.11.21 Event: post-reload

Description

Called just after a streaming server is reloaded.

Parameters

The following parameters are passed, in the order shown, to this script:

- **username** (`string`)
  the username of the account

Return Value

Ignored.

Sample Code

```
#!/usr/bin/env bash
# post-reload
USERNAME="$post-reload"

# implementation details here ...
```

5.11.22 Event: pre-stop-source

Description

Called just before an autoDJ is stopped.

Parameters

The following parameters are passed, in the order shown, to this script:

- **username** (`string`)
  the username of the account

Return Value

The last line of output from the script determines whether or not the associated event will continue.

If the last line of output contains only the number `1`, the event will proceed normally.

Any other value prevents the autoDJ from stopping.

Sample Code

```
#!/usr/bin/env bash
# pre-stop-source
USERNAME="$pre-stop-source"
ERROR=""
```

```
# implementation details here ...

# if you want to force Centova Cast to abort the event, your script should
# set ERROR to an error message string
[ -z "$ERROR" ] && echo "1" || echo "$ERROR"
```

5.11.23 Event: pre-stop-app

Description

Called just before a supplemental application is stopped.

Parameters

The following parameters are passed, in the order shown, to this script:

- **type** (string)
  the application type
- **username** (string)
  the username of the account

Return Value

The last line of output from the script determines whether or not the associated event will continue.

If the last line of output contains only the number 1, the event will proceed normally.

Any other value prevents the application from stopping.

Sample Code

```
#!/usr/bin/env bash
# pre-stop-app
TYPE="$pre-stop-app"
USERNAME="$pre-stop-app"
ERROR=""

# implementation details here ...

# if you want to force Centova Cast to abort the event, your script should
# set ERROR to an error message string
[ -z "$ERROR" ] && echo "1" || echo "$ERROR"
```

5.11.24 Event: post-stop-source

Description

Called just after an autoDJ is stopped.

Parameters

The following parameters are passed, in the order shown, to this script:

- **username** (`string`)
  the username of the account

Return Value

Ignored.

Sample Code

```
#!/usr/bin/env bash
# post-stop-source
USERNAME="$post-stop-source"

# implementation details here ...
```

5.11.25 Event: post-stop-app

Description

Called just after a supplemental application is stopped.

Parameters

The following parameters are passed, in the order shown, to this script:

- **app** (`string`)
  the application type
- **username** (`string`)
  the username of the account

Return Value

Ignored.

Sample Code

```
#!/usr/bin/env bash
# post-stop-app
APP="$post-stop-app"
USERNAME="$post-stop-app"

# implementation details here ...
```

5.11.26 Event: pre-stop-server

Description

Called just before a streaming server is stopped.

Parameters

The following parameters are passed, in the order shown, to this script:

- **username** (`string`)
  the username of the account

Return Value

The last line of output from the script determines whether or not the associated event will continue.

If the last line of output contains only the number `1`, the event will proceed normally.

Any other value prevents the streaming server from stopping.

Sample Code

```
#!/usr/bin/env bash
# pre-stop-server
USERNAME="$pre-stop-server"
ERROR=""

# implementation details here ...

# if you want to force Centova Cast to abort the event, your script should
# set ERROR to an error message string
[ -z "$ERROR" ] && echo "1" || echo "$ERROR"
```

5.11.27 Event: post-stop-server

Description

Called just after a streaming server is stopped.

Parameters

The following parameters are passed, in the order shown, to this script:

- **username** (`string`)
  the username of the account

Return Value

Ignored.

Sample Code

```
#!/usr/bin/env bash
# post-stop-server
USERNAME="$post-stop-server"

# implementation details here ...
```

5.11.28 Event: pre-reindex

Description

Called just before a media library is reindexed (updated).

Parameters

The following parameters are passed, in the order shown, to this script:

- **username** (`string`)
  the username of the account

Return Value

The last line of output from the script determines whether or not the associated event will continue.

If the last line of output contains only the number `1`, the event will proceed normally.

Any other value prevents the media library from updating.

Sample Code

```
#!/usr/bin/env bash
# pre-reindex
USERNAME="$pre-reindex"
ERROR=""
```

```
# implementation details here ...

# if you want to force Centova Cast to abort the event, your script should
# set ERROR to an error message string
[ -z "$ERROR" ] && echo "1" || echo "$ERROR"
```

5.11.29 Event: post-reindex

Description

Called just after a media library is reindexed (updated).

Parameters

The following parameters are passed, in the order shown, to this script:

- **username** (string)
  the username of the account

Return Value

Ignored.

Sample Code

```
#!/usr/bin/env bash
# post-reindex
USERNAME="$post-reindex"

# implementation details here ...
```

5.11.30 Event: pre-process-logs

Description

Called just before an account's logs are processed.

Parameters

The following parameters are passed, in the order shown, to this script:

- **username** (string)
  the username of the account

Return Value

The last line of output from the script determines whether or not the associated event will continue.

If the last line of output contains only the number `1`, the event will proceed normally.

Any other value prevents the processing of the account's logs.

Sample Code

```
#!/usr/bin/env bash
# pre-process-logs
USERNAME="$pre-process-logs"
ERROR=""

# implementation details here ...

# if you want to force Centova Cast to abort the event, your script should
# set ERROR to an error message string
[ -z "$ERROR" ] && echo "1" || echo "$ERROR"
```

5.11.31 Event: post-process-logs

Description

Called just after an account's logs have been processed.

Parameters

The following parameters are passed, in the order shown, to this script:

- **username** (`string`)
  the username of the account

Return Value

Ignored.

Sample Code

```
#!/usr/bin/env bash
# post-process-logs
USERNAME="$post-process-logs"

# implementation details here ...
```

5.11.32 Event: pre-rotate-logs

Description

Called just before an account's logs are rotated.

Parameters

The following parameters are passed, in the order shown, to this script:

- **username** (`string`)
  the username of the account

Return Value

The last line of output from the script determines whether or not the associated event will continue.

If the last line of output contains only the number `1`, the event will proceed normally.

Any other value prevents the rotation of the account's logs.

Sample Code

```
#!/usr/bin/env bash
# pre-rotate-logs
USERNAME="$pre-rotate-logs"
ERROR=""

# implementation details here ...

# if you want to force Centova Cast to abort the event, your script should
# set ERROR to an error message string
[ -z "$ERROR" ] && echo "1" || echo "$ERROR"
```

5.11.33 Event: post-rotate-logs

Description

Called just after an account's logs have been rotated.

Parameters

The following parameters are passed, in the order shown, to this script:

- **username** (`string`)
  the username of the account

Return Value

Ignored.

Sample Code

```
#!/usr/bin/env bash
# post-rotate-logs
USERNAME="$post-rotate-logs"

# implementation details here ...
```

5.11.34 Event: pre-update-reseller

Description

Called just before a reseller account is updated.

Parameters

The following parameters are passed, in the order shown, to this script:

- **username** `(string)`
  the username of the account
- **password** `(string)`
  the password for the account
- **email** `(string)`
  the email address for the account
- **maxclients** `(int)`
  the listener limit for the account
- **maxbitrate** `(int)`
  the maximum bit rate for the account
- **transferlimit** `(int)`
  the data transfer limit for the account
- **diskquota** `(int)`
  the disk quota for the account
- **usesource** `(int)`
  1 if the reseller can create autoDJ-enabled accounts, otherwise 0
- **mountlimit** `(int)`
  the mount point limit for the account

Return Value

The last line of output from the script determines whether or not the associated event will continue.

If the last line of output contains only the number `1`, the event will proceed normally.

Any other value prevents the update from being saved.

Sample Code

```
#!/usr/bin/env bash
# pre-update-reseller
USERNAME="$pre-update-reseller"
PASSWORD="$pre-update-reseller"
EMAIL="$pre-update-reseller"
MAXCLIENTS="$pre-update-reseller"
MAXBITRATE="$pre-update-reseller"
TRANSFERLIMIT="$pre-update-reseller"
DISKQUOTA="$pre-update-reseller"
USESOURCE="$pre-update-reseller"
MOUNTLIMIT="$pre-update-reseller"
ERROR=""

# implementation details here ...

# if you want to force Centova Cast to abort the event, your script should
# set ERROR to an error message string
[ -z "$ERROR" ] && echo "1" || echo "$ERROR"
```

5.11.35 Event: pre-update-account

Description

Called just before a streaming account is updated.

Parameters

The following parameters are passed, in the order shown, to this script:

- **username** (`string`)
  the username of the account
- **password** (`string`)
  the password for the account
- **email** (`string`)
  the email address for the account

- **ipaddress** (string)
  the IP address for the account
- **port** (int)
  the port number for the account
- **maxclients** (int)
  the listener limit for the account
- **maxbitrate** (int)
  the maximum bit rate for the account
- **transferlimit** (int)
  the data transfer limit for the account
- **diskquota** (int)
  the disk quota for the account
- **usesource** (int)
  1 if the autoDJ is enabled, otherwise 0
- **mountlimit** (int)
  the mount point limit for the account

Return Value

The last line of output from the script determines whether or not the associated event will continue.

If the last line of output contains only the number 1, the event will proceed normally.

Any other value prevents the update from being saved.

Sample Code

```
#!/usr/bin/env bash
# pre-update-account
USERNAME="$pre-update-account"
PASSWORD="$pre-update-account"
EMAIL="$pre-update-account"
IPADDRESS="$pre-update-account"
PORT="$pre-update-account"
MAXCLIENTS="$pre-update-account"
MAXBITRATE="$pre-update-account"
TRANSFERLIMIT="$pre-update-account"
DISKQUOTA="$pre-update-account"
USESOURCE="$pre-update-account"
MOUNTLIMIT="$pre-update-account"
ERROR=""

# implementation details here ...
```

```
# if you want to force Centova Cast to abort the event, your script should
# set ERROR to an error message string
[ -z "$ERROR" ] && echo "1" || echo "$ERROR"
```

5.11.36 Event: post-update-reseller

Description

Called just after a reseller account is updated.

Parameters

The following parameters are passed, in the order shown, to this script:

- **username** (string)
  the username of the account
- **password** (string)
  the password for the account
- **email** (string)
  the email address for the account
- **maxclients** (int)
  the listener limit for the account
- **maxbitrate** (int)
  the maximum bit rate for the account
- **transferlimit** (int)
  the data transfer limit for the account
- **diskquota** (int)
  the disk quota for the account
- **usesource** (int)
  1 if the reseller can create autoDJ-enabled accounts, otherwise 0
- **mountlimit** (int)
  the mount point limit for the account

Return Value

Ignored.

Sample Code

```
#!/usr/bin/env bash
# post-update-reseller
USERNAME="$post-update-reseller"
PASSWORD="$post-update-reseller"
EMAIL="$post-update-reseller"
MAXCLIENTS="$post-update-reseller"
MAXBITRATE="$post-update-reseller"
TRANSFERLIMIT="$post-update-reseller"
DISKQUOTA="$post-update-reseller"
USESOURCE="$post-update-reseller"
MOUNTLIMIT="$post-update-reseller"

# implementation details here ...
```

5.11.37 Event: post-update-account

Description

Called just after a streaming account is updated.

Parameters

The following parameters are passed, in the order shown, to this script:

- **username** `(string)`
  the username of the account
- **password** `(string)`
  the password for the account
- **email** `(string)`
  the email address for the account
- **ipaddress** `(string)`
  the IP address for the account
- **port** `(int)`
  the port number for the account
- **maxclients** `(int)`
  the listener limit for the account
- **maxbitrate** `(int)`
  the maximum bit rate for the account
- **transferlimit** `(int)`
  the data transfer limit for the account

- **diskquota** (int)
  the disk quota for the account
- **usesource** (int)
  1 if the autoDJ is enabled, otherwise 0
- **mountlimit** (int)
  the mount point limit for the account

Return Value

Ignored.

Sample Code

```
#!/usr/bin/env bash
# post-update-account
USERNAME="$post-update-account"
PASSWORD="$post-update-account"
EMAIL="$post-update-account"
IPADDRESS="$post-update-account"
PORT="$post-update-account"
MAXCLIENTS="$post-update-account"
MAXBITRATE="$post-update-account"
TRANSFERLIMIT="$post-update-account"
DISKQUOTA="$post-update-account"
USESOURCE="$post-update-account"
MOUNTLIMIT="$post-update-account"

# implementation details here ...
```

5.11.38 Event: send-email

Description

Called when Centova Cast needs to send an email message (usually for notifications).

Parameters

The following parameters are passed, in the order shown, to this script:

- **name** (string)
  the name of the email template to send
- **recipients** (array)
  a list of recipient email addresses for the message

- **subject** (string)
  the subject for the message
- **variables** (array)
  a list of variables available to be populated into the message text
- **text** (string)
  the text/plain version of the message body
- **html** (string)
  the text/html version of the message body

Return Value

The last line of output from the script determines whether or not the associated event will continue.

If the last line of output contains only the number 1, the event will proceed normally.

Any other value triggers an error and prevents the email from being sent by Centova Cast.

Sample Code

```
#!/usr/bin/env bash
# send-email
NAME="$send-email"
RECIPIENTS="$send-email"
SUBJECT="$send-email"
VARIABLES="$send-email"
TEXT="$send-email"
HTML="$send-email"
ERROR=""

# implementation details here ...

# if you want to force Centova Cast to abort the event, your script should
# set ERROR to an error message string
[ -z "$ERROR" ] && echo "1" || echo "$ERROR"
```

5.11.39 Event: pre-rename-account

Description

Called just before a streaming account is renamed.

Parameters

The following parameters are passed, in the order shown, to this script:

- **old_username** (string)
  the original (current) username of the account
- **new_username** (string)
  the new username to be assigned to the account

Return Value

The last line of output from the script determines whether or not the associated event will continue.

If the last line of output contains only the number 1, the event will proceed normally.

Any other value prevents the account from being renamed.

Sample Code

```
#!/usr/bin/env bash
# pre-rename-account
OLD_USERNAME="$pre-rename-account"
NEW_USERNAME="$pre-rename-account"
ERROR=""

# implementation details here ...

# if you want to force Centova Cast to abort the event, your script should
# set ERROR to an error message string
[ -z "$ERROR" ] && echo "1" || echo "$ERROR"
```

5.11.40 Event: pre-rename-reseller

Description

Called just before a reseller account is renamed.

Parameters

The following parameters are passed, in the order shown, to this script:

- **old_username** (string)
  the original (current) username of the account
- **new_username** (string)
  the new username to be assigned to the account

Return Value

The last line of output from the script determines whether or not the associated event will continue.

If the last line of output contains only the number `1`, the event will proceed normally.

Any other value prevents the account from being renamed.

Sample Code

```
#!/usr/bin/env bash
# pre-rename-reseller
OLD_USERNAME="$pre-rename-reseller"
NEW_USERNAME="$pre-rename-reseller"
ERROR=""

# implementation details here ...

# if you want to force Centova Cast to abort the event, your script should
# set ERROR to an error message string
[ -z "$ERROR" ] && echo "1" || echo "$ERROR"
```

5.11.41 Event: post-rename-account

Description

Called just after a streaming account is renamed

Parameters

The following parameters are passed, in the order shown, to this script:

- **old_username** `(string)`
  the original (old) username of the account
- **new_username** `(string)`
  the new (current) username for the account

Return Value

Ignored.

Sample Code

```
#!/usr/bin/env bash
# post-rename-account
OLD_USERNAME="$post-rename-account"
```

```
NEW_USERNAME="$post-rename-account"

# implementation details here ...
```

5.11.42 Event: post-rename-reseller

Description

Called just after a reseller account is renamed

Parameters

The following parameters are passed, in the order shown, to this script:

- **old_username** (string)
  the original (old) username of the account
- **new_username** (string)
  the new (current) username for the account

Return Value

Ignored.

Sample Code

```
#!/usr/bin/env bash
# post-rename-reseller
OLD_USERNAME="$post-rename-reseller"
NEW_USERNAME="$post-rename-reseller"

# implementation details here ...
```

5.11.43 Event: server-outage-restarted

Description

Called when a streaming server is restarted due to an outage

Parameters

The following parameters are passed, in the order shown, to this script:

- **username** (string)
  the username of the account

Return Value

Ignored.

Sample Code

```
#!/usr/bin/env bash
# server-outage-restarted
USERNAME="$server-outage-restarted"

# implementation details here ...
```

5.11.44 Event: server-outage-restart-failed

Description

Called when a streaming server cannot be restarted after an outage

Parameters

The following parameters are passed, in the order shown, to this script:

- **username** `(string)`
  the username of the account

Return Value

Ignored.

Sample Code

```
#!/usr/bin/env bash
# server-outage-restart-failed
USERNAME="$server-outage-restart-failed"

# implementation details here ...
```

5.11.45 Event: source-outage-restarted

Description

Called when an autoDJ is restarted due to an outage

Parameters

The following parameters are passed, in the order shown, to this script:

- **username** (`string`)
  the username of the account

Return Value

Ignored.

Sample Code

```
#!/usr/bin/env bash
# source-outage-restarted
USERNAME="$source-outage-restarted"

# implementation details here ...
```

5.11.46 Event: source-outage-restart-failed

Description

Called when an autoDJ cannot be restarted after an outage

Parameters

The following parameters are passed, in the order shown, to this script:

- **username** (`string`)
  the username of the account

Return Value

Ignored.

Sample Code

```
#!/usr/bin/env bash
# source-outage-restart-failed
USERNAME="$source-outage-restart-failed"

# implementation details here ...
```

5.11.47 Event: app-outage-restarted

Description

Called when a supplemental application is restarted due to an outage

Parameters

The following parameters are passed, in the order shown, to this script:

- **username** `(string)`
  the username of the account

Return Value

Ignored.

Sample Code

```
#!/usr/bin/env bash
# app-outage-restarted
USERNAME="$app-outage-restarted"

# implementation details here ...
```

5.11.48 Event: app-outage-restart-failed

Description

Called when a supplemental application cannot be restarted after an outage

Parameters

The following parameters are passed, in the order shown, to this script:

- **username** `(string)`
  the username of the account

Return Value

Ignored.

Sample Code

```
#!/usr/bin/env bash
# app-outage-restart-failed
USERNAME="$app-outage-restart-failed"

# implementation details here ...
```

5.11.49 Event: bitrate-exceeded

Description

Called when a stream is broadcasting in excess of its configured bit rate limit.

Parameters

The following parameters are passed, in the order shown, to this script:

- **username** (`string`)
  the username of the account

Return Value

The last line of output from the script determines whether or not the associated event will continue.

If the last line of output contains only the number `1`, the event will proceed normally.

Any other value prevents the bit rate limit from being enforced and allows the stream to continue broadcasting.

Sample Code

```
#!/usr/bin/env bash
# bitrate-exceeded
USERNAME="$bitrate-exceeded"
ERROR=""

# implementation details here ...

# if you want to force Centova Cast to abort the event, your script should
# set ERROR to an error message string
[ -z "$ERROR" ] && echo "1" || echo "$ERROR"
```

5.12 Event Notification Scripts

In addition to its plugin API, Centova Cast includes a mechanism by which user-defined scripts can automatically be invoked whenever Centova Cast performs one of a set of specific actions.

5.12.1 Disclaimer

Please note that this event script capability is provided as a courtesy to our clients, and Centova Technologies cannot provide technical support related to the use of these scripts to perform external tasks.

5.12.2 Getting Started

For security and performance reasons, support for event notification scripts is disabled by default. You must enable the `EVENT_SCRIPTS` setting in `/usr/local/centovacast/etc/centovacast.conf` before the event notification scripts will be invoked by Centova Cast.

5.12.3 Important Notes

- For performance reasons, Centova Cast caches the list of event notification scripts the first time an event is triggered. As such, if you add a new event notification script after starting Centova Cast, it will not immediately be recognized by Centova Cast. To force Centova Cast to update its cache, simply restart Centova Cast (on the web interface server) by running `/etc/init.d/centovacast restart`, or clear the cache (on the web interface server) by running `/usr/local/centovacast/sbin/clear_web_cache`, or wait 24hr for the cache to expire.
- Event notification scripts will be executed on the stream hosting server on which the relevant account is hosted, which may not (necessarily) be the same server that hosts your Centova Cast web interface.
- Event notification scripts will be executed under the user ID of the Centova Cast daemon account, which is `ccuser` by default. As such, the scripts will have access to the stream accounts' configuration files and media, but will NOT have root privileges.
- Scripts that allow a return value MUST output "1" or "0" to indicate whether or not Centova Cast may proceed with the operation. If the last line generated by your script is anything other than 1, Centova Cast will abort with an error. If the last line of output generated by your script is a text string, it will be displayed as part of the error message in Centova Cast.
- Note that when implementing code for "pre-" events, there is no guarantee that the operation that is about to be performed will succeed. You should use "pre-" events only for the purposes of (potentially) aborting the operation. Any code which depends on the successful completion of the operation (such as account provisioning code, etc.) should be triggered by the "post-" version of each event.

- Centova Cast will by default wait up to 30 seconds for an event notification script to finish. If the script does not finish in this period of time, it will be terminated by Centova Cast. Remember that in most cases, a user will be waiting for a page to load in the web interface while the event notification script is running, so it's important that the script completes quickly. The time limit can be changed by adding an `EVENT_SCRIPT_TIMEOUT=30` line to `/usr/local/centovacast/etc/centovacast.conf`, replacing `30` with the desired timeout.

5.13 Event Script Structure

5.13.1 Overview

Event scripts may be written in any programming language supported by your server; Centova Cast simply executes the script as an executable file and reads the output, so it is unconcerned with the implementation details.

5.13.2 Filename and Location

Event scripts are always executed on the stream hosting server. If you only run a single Centova Cast server, this implementation detail is irrelevant, however if you use a single web interface to control multiple Centova Cast stream hosting servers, you must ensure that the event scripts are created on the hosting servers, not the web interface server.

To implement a script for a particular event, create a file in `/usr/local/centovacast/var/events/` called `eventname.sh`, where *eventname* is the name of the event. A list of available event names, and their parameters, is provided in the Event Reference section of the event scripts documentation.

For example, for the **pre-reload** event, you would create a script named `/usr/local/centovacast/var/events/pre-`

5.13.3 Implementation

As noted above, the event script's implementation details are left to the developer to decide. Centova Cast only requires that the file be a valid executable file, be it a script or a binary application.

Some events (noted in the Event Reference) require the script to pass a return value to back to Centova Cast. These scripts must pass the return value as the last line of output.

Valid return values include `1`, indicating success, `0`, indicating failure, or a text string, representing an error message which will be displayed in the web interface or server logs. Any return value other than success will result in Centova Cast aborting the operation it was attempting to perform.

Event notification scripts will be executed under the user ID of the Centova Cast daemon account, which is `ccuser` by default. As such, the scripts will have access to the stream accounts' configuration files and media, but will NOT have root privileges.

Chapter 6

Files and Paths

This section provides an overview of the files and paths used by Centova Cast, to assist advanced systems administrators in diagnosing problems, customizing Centova Cast, or integrating Centova Cast with third-party solutions.

6.1 Log Files

Centova Cast maintains log files for its web, application, and FTP servers under the `/usr/local/centovacast/var/lo` directory. The log files include:

- **cc-web.log**
  the error/info log file for the Centova Cast web server
- **cc-web_access.log**
  the access log file for the Centova Cast web server
- **cc-ftpd.log**
  the log file for the Centova Cast FTP server
- **control/master.log**
  the log file for the Centova Cast daemon (critical information may be logged to syslog instead)
- **imaged/*.log**
  the log files for the Centova Cast image daemon
- **comet/cc-comet.log**
  the log file for the Centova Cast comet daemon

Additionally, separate logs are maintained for each client account under the `/usr/local/centovacast/var/vhosts/U` directory. The per-account log files include:

- **access.log**
  the usage log file for the streaming server software used by the account (IceCast, SHOUTcast, etc.)
- **error.log**
  the error/diagnostic/debug log file for the streaming server software used by the account
- **source.log**
  the log file for the autoDJ software used by the account (ices, sc_trans, etc); some autoDJ software may use alternate names for this log file
- **nextsong.log**
  the log file for the Centova Cast autoDJ interface, which provides instructions to the autoDJ software indicating what songs should be played, and when
- **reports/*.zip**
  the monthly statistics report files for the account

6.2 Client Data (Linux)

On Linux servers, client data (configuration files, media files, etc.) is stored under the `/usr/local/centovacast/var/v` directory.

If you want your client data stored on a different (perhaps larger) partition, you may set up a bind mount to point elsewhere. For example, you might edit `/etc/fstab` and add:

```
/usr/local/centovacast/var/vhosts    /opt/largedisk/foo    none    bind
```

Replace `/opt/largedisk/foo` with the path to the directory in which you want the client data to be stored. Finally, run the following command to activate the bind mount:

```
mount /usr/local/centovacast/var/vhosts
```

You only need to run the mount command once; the next time you reboot, the bind mount will be configured automatically from `/etc/fstab`.

6.3 Client Data (Windows)

On Windows servers, client data (configuration files, media files, etc.) is stored in under `C:\CCVHosts` folder. This location is configurable during the installation of the Windows Control Daemon.

6.4 Cron Job

Centova Cast maintains its own crontab file in `/etc/cron.d/centovacast` instead of modifying `/etc/crontab` directly. You can modify this file but bear in mind that it may be overwritten by future updates as needed.

6.5 Configuration

Centova Technologies does not recommend modifying the Centova Cast configuration files except as directed by the Centova Technologies helpdesk staff. The purpose of this section is to identify the purpose of each configuration file under `/usr/local/centovacast/etc/` to assist advanced administrators who may have prior experience with the underlying software used by Centova Cast.

6.5.1 Web Interface

- **centovacast.conf**
  This is the master configuration file for Centova Cast's web interface.
- **cc-panel.conf**
  This file sets basic configuration options for Centova Cast's web server.
- **caching.conf**
  This file configures Centova Cast's cache options.
- **cc-appserver.conf**
  This file controls how Centova Cast's FastCGI backend manages its processes. If your web interface sees a lot of traffic and you start seeing HTTP code 5xx errors in the web interface, increasing `APPSERVER_CHILDREN` in this file may help.
- **cc-web.conf**
  This is the main configuration file for Centova Cast's web server. Currently, Centova Cast's web server is nginx, so this and the files under the `web.d/` subdirectory are nginx configuration files.
- **web.d/cc-interface.conf**
  This is the web server configuration file for the Centova Cast web interface.
- **web.d/cc-content.conf**
  This is the web server configuration file for on-demand content and other static content served from clients' home directories.
- **cc-system.conf**
  This is the internal configuration file for Centova Cast's application server. The application server in this case happens to be PHP in FastCGI mode, and this file happens to be a php.ini. Note that Centova Cast uses PHP in a manner that is very different than a typical Apache/PHP configuration, and these settings have been carefully tuned for correct operation; administrators who are familiar with PHP should be advised that modifying this file is discouraged and may not yield the expected results.
- **cc-dynamic.conf**
  This file is dynamically generated from the contents of various other configuration files at each startup. It should not be modified directly.

6.5.2 FTP Server

- **cc-ftpd.conf**
  This is the configuration file for Centova Cast's FTP server. Normally you shouldn't need to modify this unless you want to configure bandwidth throttling or bind the FTP server to a specific address.

6.5.3 Control Daemon (Linux)

The Centova Cast control daemon is, in simplified terms, an agent which runs on all Centova Cast stream hosting servers and manages application execution and disk access. In a scenario where a hosting provider has multiple physical servers hosting streams, installing the control daemon on each server allows all of the servers to be remotely managed by a single Centova Cast web interface server, substantially reducing the memory and CPU overhead on each hosting server.

- **cc-control.conf**
  This is the main configuration file for Centova Cast's control daemon. It is well-documented with comments but you shouldn't normally need to modify it.
- **license.conf**
  This file contains your license key. If you need to install a new license in future, update the key here.
- **rpcaccess**
  Configures the list of IP addresses which are permitted to connect to the control daemon. Normally this should only include `localhost` (`127.0.0.1`) and the IP address of your Centova Cast web interface.
- **rpcshadow**
  Configures the secret key used by the Centova Cast web interface to connect to the control daemon. Keep this private and secure at all times, as it allows unrestricted access to your server.

6.5.4 Control Daemon (Windows)

The Windows control daemon stores all of its configuration data in the Windows registry under the `HKEY_LOCAL_MACHINE\SOFTWARE\Centova Technologies\Centova Cast Control Daemon` registry key.

6.5.5 Image Daemon

The Centova Cast image daemon is a high-performance image manipulation server used by Centova Cast to resize, crop, and otherwise manipulate album cover images. While it may seem odd to have a server daemon dedicated to this purpose in an application where image processing is not a primary function, it quickly becomes apparent that, during track importing, image manipulation becomes a significant performance bottleneck.

Based on the well-known imlib2 image processing library, this daemon outperforms virtually any other general-purpose image manipulation solution on Linux, and the raw speed at which the daemon is able to process images dramatically reduces the time required to import new tracks.

- **cc-imaged.conf**
  This is the main configuration file for Centova Cast's image daemon. It is well-documented with comments but you shouldn't normally need to modify it.

6.5.6 Comet Daemon

The Centova Cast comet daemon handles persistent "push" connections to clients, allowing for realtime status information to be transmitted to users' browsers.

- **cc-comet.conf**
  This is the configuration file for the comet daemon. You shouldn't normally need to modify this.

6.5.7 General Configuration

- **cc-services.conf**
  This file tells Centova Cast what services are running on the local machine. This file need not be modified manually; installing a service as described in section 2 of the quick reference will automatically update this file.
- **update.conf**
  Configures the URL and channel for Centova Cast updates. Typically this should not be modified except as instructed by Centova Technologies.

6.5.8 Other Files

Any other files found under the `etc/` directory are support files for Centova Cast's service applications and should not be modified.

6.6 Account Files and Paths

This section provides an overview of thefiles and paths used by Centova Cast for each individual client account, to assist advanced systems administrators in diagnosing problems, customizing Centova Cast, or integrating Centova Cast with third-party solutions.

This document refers to the *per-account* files located under the directory:

```
/usr/local/centovacast/var/vhosts/USERNAME
```

For a description of the *core* files and paths, see Core Files and Paths instead.

6.6.1 Log Files

Centova Cast maintains log files for each client account under the directory:

```
/usr/local/centovacast/var/vhosts/USERNAME/var/log
```

The per-account log files include:

- **access.log**
  the usage log file for the streaming server software used by the account (IceCast, SHOUTcast, etc.)
- **error.log**
  the error/diagnostic/debug log file for the streaming server software used by the account
- **source.log**
  the log file for the autoDJ software used by the account (ices, sc_trans, etc); some autoDJ software may use alternate names for this log file
- **nextsong.log**
  the log file for the Centova Cast autoDJ interface, which provides instructions to the autoDJ software indicating what songs should be played, and when
- **reports/*.zip**
  the monthly statistics report files for the account

6.6.2 Configuration

Centova Technologies does not recommend modifying the Centova Cast configuration files except as directed by the Centova Technologies helpdesk staff. The purpose of this section is to identify the purpose of each configuration file under each account's `etc/` directory to assist advanced administrators who may have prior experience with the underlying software used by Centova Cast.

- **server.conf**
  This is the configuration file for the streaming server software (SHOUTcast DNAS, IceCast, etc.) configured for use with the account. The format of this file will vary depending on which streaming server software is in use; the manual for the streaming server software should be consulted for details.

- **source.conf**
  This is the configuration file for the autoDJ software (ices, sc_trans, etc.) configured for use with the account. The format of this file will vary depending on which autoDJ software is in use; the manual for the autoDJ software should be consulted for details.

- **nextsong.conf**
  This file is used to configure the integration between Centova Cast and the autoDJ software configured for use with the account. In most cases this file should never be modified.

- **playlist.conf**
  (sc_trans v2 only.) This file is an intermediate playlist which directs sc_trans v2 to use Centova Cast's autoDJ for track selection.

6.7 Core Files and Paths

This section provides an overview of the core files and paths used by Centova Cast, to assist advanced systems administrators in diagnosing problems, customizing Centova Cast, or integrating Centova Cast with third-party solutions.

This document refers to the *core* files located under the main `/usr/local/centovacast/` directory. For a description of *per-account* files and paths, see Account Files and Paths instead.

6.7.1 Log Files

Centova Cast maintains log files for its web, application, and FTP servers under the directory:

```
/usr/local/centovacast/var/log/
```

The log files include:

- **cc-web.log**
  the error/info log file for the Centova Cast web server
- **cc-web_access.log**
  the access log file for the Centova Cast web server
- **cc-ftpd.log**
  the log file for the Centova Cast FTP server
- **control/master.log**
  the log file for the Centova Cast daemon (critical information may be logged to syslog instead)
- **imaged/*.log**
  the log files for the Centova Cast image daemon

6.7.2 Client Data (Linux)

On Linux servers, client data (configuration files, media files, etc.) is stored under the directory:

```
/usr/local/centovacast/var/vhosts/
```

If you want your client data stored on a different (perhaps larger) partition, you may set up a bind mount to point elsewhere. For example, you might edit `/etc/fstab` and add:

```
/opt/largedisk/foo    /usr/local/centovacast/var/vhosts    none    bind
```

Replace `/opt/largedisk/foo` with the path to the directory in which you want the client data to be stored. Finally, run the following command to activate the bind mount:

```
mount /usr/local/centovacast/var/vhosts
```

You only need to run the mount command once; the next time you reboot, the bind mount will be configured automatically from `/etc/fstab`.

For a description of the files and paths located under each account's client data directory, see Account Files and Paths.

6.7.3 Client Data (Windows)

On Windows servers, client data (configuration files, media files, etc.) is stored in under `C:\CCVHosts` folder. This location is configurable during the installation of the Windows Control Daemon.

6.7.4 Cron Job

Centova Cast maintains its own crontab file in `/etc/cron.d/centovacast` instead of modifying `/etc/crontab` directly. You can modify this file but bear in mind that it may be overwritten by future updates as needed.

6.7.5 Configuration

Centova Technologies does not recommend modifying the Centova Cast configuration files except as directed by the Centova Technologies helpdesk staff. The purpose of this section is to identify the purpose of each configuration file under `/usr/local/centovacast/etc/` to assist advanced administrators who may have prior experience with the underlying software used by Centova Cast.

Web Interface

- **centovacast.conf**
  This is the master configuration file for Centova Cast's web interface. These options correspond to the options previously located in `/home/centovacast/system/config.php` in Centova Cast v2.x.
- **cc-appserver.conf**
  This file controls how Centova Cast's FastCGI backend manages its processes. If your web interface sees a lot of traffic and you start seeing HTTP code 5xx errors in the web interface, increasing `APPSERVER_CHILDREN` in this file may help.
- **cc-web.conf**
  This is the main configuration file for Centova Cast's web server. Currently, Centova Cast's web server is nginx, so this and the files under the `web.d/` subdirectory are nginx configuration files.
- **web.d/cc-interface.conf**
  This is the web server configuration file for the Centova Cast web interface.
- **web.d/cc-content.conf**
  This is the web server configuration file for on-demand content and other static content served from clients' home directories.

- **cc-system.conf**
  This is the internal configuration file for Centova Cast's application server. The application server in this case happens to be PHP in FastCGI mode, and this file happens to be a php.ini. Note that Centova Cast uses PHP in a manner that is very different than a typical Apache/PHP configuration, and these settings have been carefully tuned for correct operation; administrators who are familiar with PHP should be advised that modifying this file is discouraged and may not yield the expected results.

FTP Server

- **cc-ftpd.conf**
  This is the configuration file for Centova Cast's FTP server. Normally you shouldn't need to modify this unless you want to configure bandwidth throttling or bind the FTP server to a specific address.

Control Daemon (Linux)

The Centova Cast control daemon is, in simplified terms, an agent which runs on all Centova Cast stream hosting servers and manages application execution and disk access. In a scenario where a hosting provider has multiple physical servers hosting streams, installing the control daemon on each server allows all of the servers to be remotely managed by a single Centova Cast web interface server, substantially reducing the memory and CPU overhead on each hosting server.

- **cc-control.conf**
  This is the main configuration file for Centova Cast's control daemon. It is well-documented with comments but you shouldn't normally need to modify it.
- **license.conf**
  This file contains your license key. If you need to install a new license in future, update the key here.
- **rpcaccess**
  Configures the list of IP addresses which are permitted to connect to the control daemon. Normally this should only include `localhost` (`127.0.0.1`) and the IP address of your Centova Cast web interface.
- **rpcshadow**
  Configures the secret key used by the Centova Cast web interface to connect to the control daemon. Keep this private and secure at all times, as it allows unrestricted access to your server.

Control Daemon (Windows)

The Windows control daemon stores all of its configuration data in the Windows registry under the `HKEY_LOCAL_MACHINE\SOFTWARE\Centova Technologies\Centova Cast Control Daemon` registry key.

Image Daemon

The Centova Cast image daemon is a high-performance image manipulation server used by Centova Cast to resize, crop, and otherwise manipulate album cover images. While it may seem odd to have a server daemon dedicated to this purpose in an application where image processing is not a primary function, it quickly becomes apparent that, during track importing, image manipulation becomes a significant performance bottleneck.

Based on the well-known imlib2 image processing library, this daemon outperforms virtually any other general-purpose image manipulation solution on Linux, and the raw speed at which the daemon is able to process images dramatically reduces the time required to import new tracks.

- **cc-imaged.conf**
  This is the main configuration file for Centova Cast's image daemon. It is well-documented with comments but you shouldn't normally need to modify it.

General Configuration

- **cc-services.conf**
  This file tells Centova Cast what services are running on the local machine. This file need not be modified manually; installing a service as described in section 2 of the quick reference will automatically update this file.
- **update.conf**
  Configures the URL and channel for Centova Cast updates. Typically this should not be modified except as instructed by Centova Technologies.

Other Files

Any other files found under the `etc/` directory are support files for Centova Cast's service applications and should not be modified.

6.7.6 nextsong.log File Format

The `nextsong.log` file, under the `/usr/local/centovacast/var/vhosts/USERNAME/var/log` directory, is a diagnostic log created by the Centova Cast autoDJ interface to record information about the songs it instructed the autoDJ to play.

Formally the log file is just freeform text and may not adhere to any specific format (particularly when debugging is enabled). However, lines that begin with a date and time followed by a ">" character are playback records.

Playback Records

In playback records, everything after the ">" character is a CSV-formatted line with the following columns:

- *duration* - indicates how long it took for Centova Cast to identify the next track to be played
- *source* - indicates from where Centova Cast obtained the next track information (usually “database” to indicate the autoDJ database; may also be “static” indicating that the autoDJ database was unreachable and a local static playlist was used; “fallback” indicating that a static playlist was not available and `fallbackfile.mp3` was served; or “error” indicating that `fallbackfile.mp3` was unavailable and the master `station_unavailable.mp3` was served)
- *pathname* - the full pathname to the song that was played
- *formatted* - the formatted metadata for the song that was played
- *artist* - the artist name for the song that was played
- *title* - the title for the song that was played
- *royaltyid* - the royalty ID for the song that was played
- *timelimit* - the time limit that was imposed
- *playlistid* - the ID number of the playlist from which the track was selected
- *playlist* - the name of the playlist from which the track was selected
- *message* - the diagnostic message generated while handling this track (usually `OK` – anything else indicates an error condition occurred)

Any one of these fields may be empty except *duration* and *message*.

6.8 Event Script Reference

The following events are available.

6.8.1 Event: playlist_advanced

Description

Called every time a track is selected from a playlist for the autoDJ.

Note that is imperative that your script work *very* quickly. During the execution of your script, the autoDJ is forced to wait to receive the information for the next track. If your script takes too long, the current song may end before the autoDJ knows which track to play next, and as a result, there may be silence on the stream and/or the server may believe that the source has died due to inactivity. You may wish to design your script to fork, exit, and and continue operation in the background if processing will take more than a few hundred milliseconds.

Parameters

The following parameters are passed, in the order shown, to this script:

- **username** `(string)`
  the username of the account
- **pathname** `(string)`
  the pathname of the track to be played
- **artist** `(string)`
  the artist of the track to be played
- **album** `(string)`
  the album of the track to be played
- **title** `(string)`
  the title of the track to be played
- **length** `(int)`
  the length of the track to be played, in seconds
- **royaltycode** `(string)`
  the royalty reporting code of the track to be played

Return Value

Returning `PluginHooks::OK` causes the event to proceed normally.

Returning `PluginHooks::COMPLETE` prevents any further event handlers from running and causes the event to proceed normally.

Sample Code

```
<?php
class PluginHooks_myplugin extends PluginHooks {

    public function install_hooks() {
        PluginHooks::register('playlist_advanced',array(&$this,'handle_playlist_advanced'));
    }

    /**
     * Handles the playlist_advanced event
     *
     * @param string $username the username of the account
     * @param string $pathname the pathname of the track to be played
     * @param string $artist the artist of the track to be played
     * @param string $album the album of the track to be played
     * @param string $title the title of the track to be played
     * @param int $length the length of the track to be played, in seconds
     * @param string $royaltycode the royalty reporting code of the track to be played
     *
     * @return int always returns PluginHooks::OK
     */
    public function handle_playlist_advanced($username,$pathname,$artist,$album,$title,$length
        try {
            /* implementation details here ... */
        } catch (Exception $e) {
            /* swallow the exception */
        }
        return PluginHooks::OK;
    }

}
```

6.8.2 Event: pre-create-reseller

Description

Called just before a reseller account is created.

Parameters

The following parameters are passed, in the order shown, to this script:

- **username** (`string`)
  the username of the account

- **password** (string)
  the password for the account
- **email** (string)
  the email address for the account
- **maxclients** (int)
  the listener limit for the account
- **maxbitrate** (int)
  the maximum bit rate for the account
- **transferlimit** (int)
  the data transfer limit for the account
- **diskquota** (int)
  the disk quota for the account
- **usesource** (int)
  1 if the reseller can create autoDJ-enabled accounts, otherwise 0
- **mountlimit** (int)
  the mount point limit for the account

Return Value

Returning `PluginHooks::OK` causes the event to proceed normally.

Returning `PluginHooks::COMPLETE` prevents any further event handlers from running and causes the event to proceed normally.

Returning `PluginHooks::ERROR` prevents the event from running. The handler may also specify an error message by calling `$this->set_error()` with the error string.

For this event, aborting prevents the account from being created.

Sample Code

```
<?php
class PluginHooks_myplugin extends PluginHooks {

    public function install_hooks() {
        PluginHooks::register('pre-create-reseller',array(&$this,'handle_precreatereseller'));
    }

    /**
     * Handles the pre-create-reseller event
     *
     * @param string $username the username of the account
     * @param string $password the password for the account
     * @param string $email the email address for the account
```

```
     * @param int $maxclients the listener limit for the account
     * @param int $maxbitrate the maximum bit rate for the account
     * @param int $transferlimit the data transfer limit for the account
     * @param int $diskquota the disk quota for the account
     * @param int $usesource 1 if the reseller can create autoDJ-enabled accounts, otherwise 0
     * @param int $mountlimit the mount point limit for the account
     *
     * @return int returns PluginHooks::OK on success, PluginHooks::ERROR on error
     */
    public function handle_precreatereseller($username,$password,$email,$maxclients,$maxbitrate
        try {
            /* implementation details here ... */
        } catch (Exception $e) {
            $this->set_error($e->getMessage());
            return PluginHooks::ERROR;
        }
        return PluginHooks::OK;
    }

}
```

6.8.3 Event: pre-create-account

Description

Called just before a streaming account is created.

Parameters

The following parameters are passed, in the order shown, to this script:

- **username** (string)
  the username of the account
- **password** (string)
  the password for the account
- **email** (string)
  the email address for the account
- **ipaddress** (string)
  the IP address for the account
- **port** (int)
  the port number for the account
- **maxclients** (int)
  the listener limit for the account

- **maxbitrate** (int)
  the maximum bit rate for the account
- **transferlimit** (int)
  the data transfer limit for the account
- **diskquota** (int)
  the disk quota for the account
- **usesource** (int)
  1 if the autoDJ is enabled, otherwise 0
- **mountlimit** (int)
  the mount point limit for the account

Return Value

Returning `PluginHooks::OK` causes the event to proceed normally.

Returning `PluginHooks::COMPLETE` prevents any further event handlers from running and causes the event to proceed normally.

Returning `PluginHooks::ERROR` prevents the event from running. The handler may also specify an error message by calling `$this->set_error()` with the error string.

For this event, aborting prevents the account from being created.

Sample Code

```
<?php
class PluginHooks_myplugin extends PluginHooks {

    public function install_hooks() {
        PluginHooks::register('pre-create-account',array(&$this,'handle_precreateaccount'));
    }

    /**
     * Handles the pre-create-account event
     *
     * @param string $username the username of the account
     * @param string $password the password for the account
     * @param string $email the email address for the account
     * @param string $ipaddress the IP address for the account
     * @param int $port the port number for the account
     * @param int $maxclients the listener limit for the account
     * @param int $maxbitrate the maximum bit rate for the account
     * @param int $transferlimit the data transfer limit for the account
     * @param int $diskquota the disk quota for the account
     * @param int $usesource 1 if the autoDJ is enabled, otherwise 0
     * @param int $mountlimit the mount point limit for the account
```

```
     *
     * @return int returns PluginHooks::OK on success, PluginHooks::ERROR on error
     */
    public function handle_precreateaccount($username,$password,$email,$ipaddress,$port,$maxcli
        try {
            /* implementation details here ... */
        } catch (Exception $e) {
            $this->set_error($e->getMessage());
            return PluginHooks::ERROR;
        }
        return PluginHooks::OK;
    }

}
```

6.8.4 Event: post-create-reseller

Description

Called just after a reseller account is created.

Parameters

The following parameters are passed, in the order shown, to this script:

- **username** (string)
  the username of the account
- **password** (string)
  the password for the account
- **email** (string)
  the email address for the account
- **maxclients** (int)
  the listener limit for the account
- **maxbitrate** (int)
  the maximum bit rate for the account
- **transferlimit** (int)
  the data transfer limit for the account
- **diskquota** (int)
  the disk quota for the account
- **usesource** (int)
  1 if the reseller can create autoDJ-enabled accounts, otherwise 0
- **mountlimit** (int)
  the mount point limit for the account

Return Value

Returning `PluginHooks::OK` causes the event to proceed normally.

Returning `PluginHooks::COMPLETE` prevents any further event handlers from running and causes the event to proceed normally.

Sample Code

```
<?php
class PluginHooks_myplugin extends PluginHooks {

    public function install_hooks() {
        PluginHooks::register('post-create-reseller',array(&$this,'handle_postcreatereseller'))
    }

    /**
     * Handles the post-create-reseller event
     *
     * @param string $username the username of the account
     * @param string $password the password for the account
     * @param string $email the email address for the account
     * @param int $maxclients the listener limit for the account
     * @param int $maxbitrate the maximum bit rate for the account
     * @param int $transferlimit the data transfer limit for the account
     * @param int $diskquota the disk quota for the account
     * @param int $usesource 1 if the reseller can create autoDJ-enabled accounts, otherwise 0
     * @param int $mountlimit the mount point limit for the account
     *
     * @return int always returns PluginHooks::OK
     */
    public function handle_postcreatereseller($username,$password,$email,$maxclients,$maxbitrat
        try {
            /* implementation details here ... */
        } catch (Exception $e) {
            /* swallow the exception */
        }
        return PluginHooks::OK;
    }

}
```

6.8.5 Event: post-create-account

Description

Called just after a streaming account is created.

Parameters

The following parameters are passed, in the order shown, to this script:

- **username** (string)
  the username of the account
- **password** (string)
  the password for the account
- **email** (string)
  the email address for the account
- **ipaddress** (string)
  the IP address for the account
- **port** (int)
  the port number for the account
- **maxclients** (int)
  the listener limit for the account
- **maxbitrate** (int)
  the maximum bit rate for the account
- **transferlimit** (int)
  the data transfer limit for the account
- **diskquota** (int)
  the disk quota for the account
- **usesource** (int)
  1 if the autoDJ is enabled, otherwise 0
- **mountlimit** (int)
  the mount point limit for the account

Return Value

Returning `PluginHooks::OK` causes the event to proceed normally.

Returning `PluginHooks::COMPLETE` prevents any further event handlers from running and causes the event to proceed normally.

Sample Code

```
<?php
class PluginHooks_myplugin extends PluginHooks {

    public function install_hooks() {
        PluginHooks::register('post-create-account',array(&$this,'handle_postcreateaccount'));
```

```
    }

    /**
     * Handles the post-create-account event
     *
     * @param string $username the username of the account
     * @param string $password the password for the account
     * @param string $email the email address for the account
     * @param string $ipaddress the IP address for the account
     * @param int $port the port number for the account
     * @param int $maxclients the listener limit for the account
     * @param int $maxbitrate the maximum bit rate for the account
     * @param int $transferlimit the data transfer limit for the account
     * @param int $diskquota the disk quota for the account
     * @param int $usesource 1 if the autoDJ is enabled, otherwise 0
     * @param int $mountlimit the mount point limit for the account
     *
     * @return int always returns PluginHooks::OK
     */
    public function handle_postcreateaccount($username,$password,$email,$ipaddress,$port,$maxcl
        try {
            /* implementation details here ... */
        } catch (Exception $e) {
            /* swallow the exception */
        }
        return PluginHooks::OK;
    }

}
```

6.8.6 Event: pre-terminate-account

Description

Called just before a streaming account is terminated.

Parameters

The following parameters are passed, in the order shown, to this script:

- **username** (`string`)
  the username of the account

Return Value

Returning `PluginHooks::OK` causes the event to proceed normally.

Returning `PluginHooks::COMPLETE` prevents any further event handlers from running and causes the event to proceed normally.

Returning `PluginHooks::ERROR` prevents the event from running. The handler may also specify an error message by calling `$this->set_error()` with the error string.

For this event, aborting prevents the account from being terminated.

Sample Code

```
<?php
class PluginHooks_myplugin extends PluginHooks {

    public function install_hooks() {
        PluginHooks::register('pre-terminate-account',array(&$this,'handle_preterminateaccount
    }

    /**
     * Handles the pre-terminate-account event
     *
     * @param string $username the username of the account
     *
     * @return int returns PluginHooks::OK on success, PluginHooks::ERROR on error
     */
    public function handle_preterminateaccount($username) {
        try {
            /* implementation details here ... */
        } catch (Exception $e) {
            $this->set_error($e->getMessage());
            return PluginHooks::ERROR;
        }
        return PluginHooks::OK;
    }

}
```

6.8.7 Event: pre-terminate-reseller

Description

Called just before a reseller account is terminated.

Parameters

The following parameters are passed, in the order shown, to this script:

- **username** (`string`)
  the username of the account

Return Value

Returning `PluginHooks::OK` causes the event to proceed normally.

Returning `PluginHooks::COMPLETE` prevents any further event handlers from running and causes the event to proceed normally.

Returning `PluginHooks::ERROR` prevents the event from running. The handler may also specify an error message by calling `$this->set_error()` with the error string.

For this event, aborting prevents the account from being terminated.

Sample Code

```
<?php
class PluginHooks_myplugin extends PluginHooks {

    public function install_hooks() {
        PluginHooks::register('pre-terminate-reseller',array(&$this,'handle_preterminateresell
    }

    /**
     * Handles the pre-terminate-reseller event
     *
     * @param string $username the username of the account
     *
     * @return int returns PluginHooks::OK on success, PluginHooks::ERROR on error
     */
    public function handle_preterminatereseller($username) {
        try {
            /* implementation details here ... */
        } catch (Exception $e) {
            $this->set_error($e->getMessage());
            return PluginHooks::ERROR;
        }
        return PluginHooks::OK;
    }

}
```

6.8.8 Event: post-terminate-account

Description

Called just after a streaming account is terminated

Parameters

The following parameters are passed, in the order shown, to this script:

- **username** (string)
  the username of the account

Return Value

Returning PluginHooks::OK causes the event to proceed normally.

Returning PluginHooks::COMPLETE prevents any further event handlers from running and causes the event to proceed normally.

Sample Code

```
<?php
class PluginHooks_myplugin extends PluginHooks {

    public function install_hooks() {
        PluginHooks::register('post-terminate-account',array(&$this,'handle_postterminateaccou
    }

    /**
     * Handles the post-terminate-account event
     *
     * @param string $username the username of the account
     *
     * @return int always returns PluginHooks::OK
     */
    public function handle_postterminateaccount($username) {
        try {
            /* implementation details here ... */
        } catch (Exception $e) {
            /* swallow the exception */
        }
        return PluginHooks::OK;
    }

}
```

6.8.9 Event: post-terminate-reseller

Description

Called just after a reseller account is terminated

Parameters

The following parameters are passed, in the order shown, to this script:

- **username** (string)
  the username of the account

Return Value

Returning PluginHooks::OK causes the event to proceed normally.

Returning PluginHooks::COMPLETE prevents any further event handlers from running and causes the event to proceed normally.

Sample Code

```
<?php
class PluginHooks_myplugin extends PluginHooks {

    public function install_hooks() {
        PluginHooks::register('post-terminate-reseller',array(&$this,'handle_postterminateresel
    }

    /**
     * Handles the post-terminate-reseller event
     *
     * @param string $username the username of the account
     *
     * @return int always returns PluginHooks::OK
     */
    public function handle_postterminatereseller($username) {
        try {
            /* implementation details here ... */
        } catch (Exception $e) {
            /* swallow the exception */
        }
        return PluginHooks::OK;
    }

}
```

6.8.10 Event: pre-reparent-account

Description

Called just before a streaming account is moved to a new reseller

Parameters

The following parameters are passed, in the order shown, to this script:

- **username** (string)
  the username of the account
- **newreseller** (string)
  the username of the new reseller account to own the streaming account

Return Value

Returning PluginHooks::OK causes the event to proceed normally.

Returning PluginHooks::COMPLETE prevents any further event handlers from running and causes the event to proceed normally.

Returning PluginHooks::ERROR prevents the event from running. The handler may also specify an error message by calling $this->set_error() with the error string.

For this event, aborting prevents the streaming server from being reparented.

Sample Code

```
<?php
class PluginHooks_myplugin extends PluginHooks {

    public function install_hooks() {
        PluginHooks::register('pre-reparent-account',array(&$this,'handle_prereparentaccount'))
    }

    /**
     * Handles the pre-reparent-account event
     *
     * @param string $username the username of the account
     * @param string $newreseller the username of the new reseller account to own the streamin
     *
     * @return int returns PluginHooks::OK on success, PluginHooks::ERROR on error
     */
    public function handle_prereparentaccount($username,$newreseller) {
        try {
            /* implementation details here ... */
        } catch (Exception $e) {
            $this->set_error($e->getMessage());
            return PluginHooks::ERROR;
        }
        return PluginHooks::OK;
    }

}
```

6.8.11 Event: post-reparent-account

Description

Called just after a streaming account is moved to a new reseller

Parameters

The following parameters are passed, in the order shown, to this script:

- **username** (string)
  the username of the account
- **newreseller** (string)
  the username of the new reseller account to own the streaming account

Return Value

Returning `PluginHooks::OK` causes the event to proceed normally.

Returning `PluginHooks::COMPLETE` prevents any further event handlers from running and causes the event to proceed normally.

Sample Code

```
<?php
class PluginHooks_myplugin extends PluginHooks {

    public function install_hooks() {
        PluginHooks::register('post-reparent-account',array(&$this,'handle_postreparentaccount
    }

    /**
     * Handles the post-reparent-account event
     *
     * @param string $username the username of the account
     * @param string $newreseller the username of the new reseller account to own the streamin
     *
     * @return int always returns PluginHooks::OK
     */
    public function handle_postreparentaccount($username,$newreseller) {
        try {
            /* implementation details here ... */
        } catch (Exception $e) {
            /* swallow the exception */
        }
        return PluginHooks::OK;
```

```
    }

}
```

6.8.12 Event: pre-account-status

Description

Called just before a streaming account's status (enabled/disabled) is changed

Parameters

The following parameters are passed, in the order shown, to this script:

- **username** (`string`)
  the username of the account
- **status** (`string`)
  the new status for the account

Return Value

Returning `PluginHooks::OK` causes the event to proceed normally.

Returning `PluginHooks::COMPLETE` prevents any further event handlers from running and causes the event to proceed normally.

Returning `PluginHooks::ERROR` prevents the event from running. The handler may also specify an error message by calling `$this->set_error()` with the error string.

For this event, aborting prevents the account's status from changing.

Sample Code

```
<?php
class PluginHooks_myplugin extends PluginHooks {

    public function install_hooks() {
        PluginHooks::register('pre-account-status',array(&$this,'handle_preaccountstatus'));
    }

    /**
     * Handles the pre-account-status event
     *
     * @param string $username the username of the account
     * @param string $status the new status for the account
     *
     * @return int returns PluginHooks::OK on success, PluginHooks::ERROR on error
```

```
     */
    public function handle_preaccountstatus($username,$status) {
        try {
            /* implementation details here ... */
        } catch (Exception $e) {
            $this->set_error($e->getMessage());
            return PluginHooks::ERROR;
        }
        return PluginHooks::OK;
    }

}
```

6.8.13 Event: post-account-status

Description

Called just after a streaming account's status (enabled/disabled) is changed

Parameters

The following parameters are passed, in the order shown, to this script:

- **username** (`string`)
  the username of the account
- **status** (`string`)
  the new status for the account

Return Value

Returning `PluginHooks::OK` causes the event to proceed normally.

Returning `PluginHooks::COMPLETE` prevents any further event handlers from running and causes the event to proceed normally.

Sample Code

```
<?php
class PluginHooks_myplugin extends PluginHooks {

    public function install_hooks() {
        PluginHooks::register('post-account-status',array(&$this,'handle_postaccountstatus'));
    }

    /**
```

```
     * Handles the post-account-status event
     *
     * @param string $username the username of the account
     * @param string $status the new status for the account
     *
     * @return int always returns PluginHooks::OK
     */
    public function handle_postaccountstatus($username,$status) {
        try {
            /* implementation details here ... */
        } catch (Exception $e) {
            /* swallow the exception */
        }
        return PluginHooks::OK;
    }

}
```

6.8.14 Event: pre-start-server

Description

Called just before a streaming server is started.

Parameters

The following parameters are passed, in the order shown, to this script:

- **username** (`string`)
  the username of the account

Return Value

Returning `PluginHooks::OK` causes the event to proceed normally.

Returning `PluginHooks::COMPLETE` prevents any further event handlers from running and causes the event to proceed normally.

Returning `PluginHooks::ERROR` prevents the event from running. The handler may also specify an error message by calling `$this->set_error()` with the error string.

For this event, aborting prevents the streaming server from starting.

Sample Code

```
<?php
class PluginHooks_myplugin extends PluginHooks {
```

```
    public function install_hooks() {
        PluginHooks::register('pre-start-server',array(&$this,'handle_prestartserver'));
    }

    /**
     * Handles the pre-start-server event
     *
     * @param string $username the username of the account
     *
     * @return int returns PluginHooks::OK on success, PluginHooks::ERROR on error
     */
    public function handle_prestartserver($username) {
        try {
            /* implementation details here ... */
        } catch (Exception $e) {
            $this->set_error($e->getMessage());
            return PluginHooks::ERROR;
        }
        return PluginHooks::OK;
    }

}
```

6.8.15 Event: post-start-server

Description

Called just after a streaming server is started.

Parameters

The following parameters are passed, in the order shown, to this script:

- **username** (`string`)
  the username of the account

Return Value

Returning `PluginHooks::OK` causes the event to proceed normally.

Returning `PluginHooks::COMPLETE` prevents any further event handlers from running and causes the event to proceed normally.

Sample Code

```
<?php
class PluginHooks_myplugin extends PluginHooks {

    public function install_hooks() {
        PluginHooks::register('post-start-server',array(&$this,'handle_poststartserver'));
    }

    /**
     * Handles the post-start-server event
     *
     * @param string $username the username of the account
     *
     * @return int always returns PluginHooks::OK
     */
    public function handle_poststartserver($username) {
        try {
            /* implementation details here ... */
        } catch (Exception $e) {
            /* swallow the exception */
        }
        return PluginHooks::OK;
    }

}
```

6.8.16 Event: pre-start-source

Description

Called just before an autoDJ is started.

Parameters

The following parameters are passed, in the order shown, to this script:

- **username** (`string`)
  the username of the account

Return Value

Returning `PluginHooks::OK` causes the event to proceed normally.

Returning `PluginHooks::COMPLETE` prevents any further event handlers from running and causes the event to proceed normally.

Returning `PluginHooks::ERROR` prevents the event from running. The handler may also specify an error message by calling `$this->set_error()` with the error string.

For this event, aborting prevents the autoDJ from starting.

Sample Code

```php
<?php
class PluginHooks_myplugin extends PluginHooks {

    public function install_hooks() {
        PluginHooks::register('pre-start-source',array(&$this,'handle_prestartsource'));
    }

    /**
     * Handles the pre-start-source event
     *
     * @param string $username the username of the account
     *
     * @return int returns PluginHooks::OK on success, PluginHooks::ERROR on error
     */
    public function handle_prestartsource($username) {
        try {
            /* implementation details here ... */
        } catch (Exception $e) {
            $this->set_error($e->getMessage());
            return PluginHooks::ERROR;
        }
        return PluginHooks::OK;
    }

}
```

6.8.17 Event: pre-start-app

Description

Called just before a supplemental application is started.

Parameters

The following parameters are passed, in the order shown, to this script:

- **type** (`string`)
  the application type
- **username** (`string`)
  the username of the account

Return Value

Returning `PluginHooks::OK` causes the event to proceed normally.

Returning `PluginHooks::COMPLETE` prevents any further event handlers from running and causes the event to proceed normally.

Returning `PluginHooks::ERROR` prevents the event from running. The handler may also specify an error message by calling `$this->set_error()` with the error string.

For this event, aborting prevents the application from starting.

Sample Code

```
<?php
class PluginHooks_myplugin extends PluginHooks {

    public function install_hooks() {
        PluginHooks::register('pre-start-app',array(&$this,'handle_prestartapp'));
    }

    /**
     * Handles the pre-start-app event
     *
     * @param string $type the application type
     * @param string $username the username of the account
     *
     * @return int returns PluginHooks::OK on success, PluginHooks::ERROR on error
     */
    public function handle_prestartapp($type,$username) {
        try {
            /* implementation details here ... */
        } catch (Exception $e) {
            $this->set_error($e->getMessage());
            return PluginHooks::ERROR;
        }
        return PluginHooks::OK;
    }

}
```

6.8.18 Event: post-start-source

Description

Called just after an autoDJ is started.

Parameters

The following parameters are passed, in the order shown, to this script:

- **username** (`string`)
  the username of the account

Return Value

Returning `PluginHooks::OK` causes the event to proceed normally.

Returning `PluginHooks::COMPLETE` prevents any further event handlers from running and causes the event to proceed normally.

Sample Code

```
<?php
class PluginHooks_myplugin extends PluginHooks {

    public function install_hooks() {
        PluginHooks::register('post-start-source',array(&$this,'handle_poststartsource'));
    }

    /**
     * Handles the post-start-source event
     *
     * @param string $username the username of the account
     *
     * @return int always returns PluginHooks::OK
     */
    public function handle_poststartsource($username) {
        try {
            /* implementation details here ... */
        } catch (Exception $e) {
            /* swallow the exception */
        }
        return PluginHooks::OK;
    }

}
```

6.8.19 Event: post-start-app

Description

Called just after a supplemental application is started.

Parameters

The following parameters are passed, in the order shown, to this script:

- **type** (string)
  the application type
- **username** (string)
  the username of the account

Return Value

Returning PluginHooks::OK causes the event to proceed normally.

Returning PluginHooks::COMPLETE prevents any further event handlers from running and causes the event to proceed normally.

Sample Code

```
<?php
class PluginHooks_myplugin extends PluginHooks {

    public function install_hooks() {
        PluginHooks::register('post-start-app',array(&$this,'handle_poststartapp'));
    }

    /**
     * Handles the post-start-app event
     *
     * @param string $type the application type
     * @param string $username the username of the account
     *
     * @return int always returns PluginHooks::OK
     */
    public function handle_poststartapp($type,$username) {
        try {
            /* implementation details here ... */
        } catch (Exception $e) {
            /* swallow the exception */
        }
        return PluginHooks::OK;
    }

}
```

6.8.20 Event: pre-reload

Description

Called just before a streaming server is reloaded.

Parameters

The following parameters are passed, in the order shown, to this script:

- **username** (`string`)
  the username of the account

Return Value

Returning `PluginHooks::OK` causes the event to proceed normally.

Returning `PluginHooks::COMPLETE` prevents any further event handlers from running and causes the event to proceed normally.

Returning `PluginHooks::ERROR` prevents the event from running. The handler may also specify an error message by calling `$this->set_error()` with the error string.

For this event, aborting prevents the streaming server from reloading.

Sample Code

```php
<?php
class PluginHooks_myplugin extends PluginHooks {

    public function install_hooks() {
        PluginHooks::register('pre-reload',array(&$this,'handle_prereload'));
    }

    /**
     * Handles the pre-reload event
     *
     * @param string $username the username of the account
     *
     * @return int returns PluginHooks::OK on success, PluginHooks::ERROR on error
     */
    public function handle_prereload($username) {
        try {
            /* implementation details here ... */
        } catch (Exception $e) {
            $this->set_error($e->getMessage());
            return PluginHooks::ERROR;
        }
```

```
        return PluginHooks::OK;
    }

}
```

6.8.21 Event: post-reload

Description

Called just after a streaming server is reloaded.

Parameters

The following parameters are passed, in the order shown, to this script:

- **username** (`string`)
  the username of the account

Return Value

Returning `PluginHooks::OK` causes the event to proceed normally.

Returning `PluginHooks::COMPLETE` prevents any further event handlers from running and causes the event to proceed normally.

Sample Code

```
<?php
class PluginHooks_myplugin extends PluginHooks {

    public function install_hooks() {
        PluginHooks::register('post-reload',array(&$this,'handle_postreload'));
    }

    /**
     * Handles the post-reload event
     *
     * @param string $username the username of the account
     *
     * @return int always returns PluginHooks::OK
     */
    public function handle_postreload($username) {
        try {
            /* implementation details here ... */
        } catch (Exception $e) {
            /* swallow the exception */
```

```
        }
        return PluginHooks::OK;
    }

}
```

6.8.22 Event: pre-stop-source

Description

Called just before an autoDJ is stopped.

Parameters

The following parameters are passed, in the order shown, to this script:

- **username** (`string`)
  the username of the account

Return Value

Returning `PluginHooks::OK` causes the event to proceed normally.

Returning `PluginHooks::COMPLETE` prevents any further event handlers from running and causes the event to proceed normally.

Returning `PluginHooks::ERROR` prevents the event from running. The handler may also specify an error message by calling `$this->set_error()` with the error string.

For this event, aborting prevents the autoDJ from stopping.

Sample Code

```
<?php
class PluginHooks_myplugin extends PluginHooks {

    public function install_hooks() {
        PluginHooks::register('pre-stop-source',array(&$this,'handle_prestopsource'));
    }

    /**
     * Handles the pre-stop-source event
     *
     * @param string $username the username of the account
     *
     * @return int returns PluginHooks::OK on success, PluginHooks::ERROR on error
     */
```

```
    public function handle_prestopsource($username) {
        try {
            /* implementation details here ... */
        } catch (Exception $e) {
            $this->set_error($e->getMessage());
            return PluginHooks::ERROR;
        }
        return PluginHooks::OK;
    }

}
```

6.8.23 Event: pre-stop-app

Description

Called just before a supplemental application is stopped.

Parameters

The following parameters are passed, in the order shown, to this script:

- **type** (`string`)
  the application type
- **username** (`string`)
  the username of the account

Return Value

Returning `PluginHooks::OK` causes the event to proceed normally.

Returning `PluginHooks::COMPLETE` prevents any further event handlers from running and causes the event to proceed normally.

Returning `PluginHooks::ERROR` prevents the event from running. The handler may also specify an error message by calling `$this->set_error()` with the error string.

For this event, aborting prevents the application from stopping.

Sample Code

```
<?php
class PluginHooks_myplugin extends PluginHooks {

    public function install_hooks() {
        PluginHooks::register('pre-stop-app',array(&$this,'handle_prestopapp'));
    }
```

```
    /**
     * Handles the pre-stop-app event
     *
     * @param string $type the application type
     * @param string $username the username of the account
     *
     * @return int returns PluginHooks::OK on success, PluginHooks::ERROR on error
     */
    public function handle_prestopapp($type,$username) {
        try {
            /* implementation details here ... */
        } catch (Exception $e) {
            $this->set_error($e->getMessage());
            return PluginHooks::ERROR;
        }
        return PluginHooks::OK;
    }

}
```

6.8.24 Event: post-stop-source

Description

Called just after an autoDJ is stopped.

Parameters

The following parameters are passed, in the order shown, to this script:

- **username** (`string`)
  the username of the account

Return Value

Returning `PluginHooks::OK` causes the event to proceed normally.

Returning `PluginHooks::COMPLETE` prevents any further event handlers from running and causes the event to proceed normally.

Sample Code

```
<?php
class PluginHooks_myplugin extends PluginHooks {
```

```
    public function install_hooks() {
        PluginHooks::register('post-stop-source',array(&$this,'handle_poststopsource'));
    }

    /**
     * Handles the post-stop-source event
     *
     * @param string $username the username of the account
     *
     * @return int always returns PluginHooks::OK
     */
    public function handle_poststopsource($username) {
        try {
            /* implementation details here ... */
        } catch (Exception $e) {
            /* swallow the exception */
        }
        return PluginHooks::OK;
    }

}
```

6.8.25 Event: post-stop-app

Description

Called just after a supplemental application is stopped.

Parameters

The following parameters are passed, in the order shown, to this script:

- **app** (`string`)
  the application type
- **username** (`string`)
  the username of the account

Return Value

Returning `PluginHooks::OK` causes the event to proceed normally.

Returning `PluginHooks::COMPLETE` prevents any further event handlers from running and causes the event to proceed normally.

Sample Code

```
<?php
class PluginHooks_myplugin extends PluginHooks {

    public function install_hooks() {
        PluginHooks::register('post-stop-app',array(&$this,'handle_poststopapp'));
    }

    /**
     * Handles the post-stop-app event
     *
     * @param string $app the application type
     * @param string $username the username of the account
     *
     * @return int always returns PluginHooks::OK
     */
    public function handle_poststopapp($app,$username) {
        try {
            /* implementation details here ... */
        } catch (Exception $e) {
            /* swallow the exception */
        }
        return PluginHooks::OK;
    }

}
```

6.8.26 Event: pre-stop-server

Description

Called just before a streaming server is stopped.

Parameters

The following parameters are passed, in the order shown, to this script:

- **username** (`string`)
  the username of the account

Return Value

Returning `PluginHooks::OK` causes the event to proceed normally.

Returning `PluginHooks::COMPLETE` prevents any further event handlers from running and causes the event to proceed normally.

Returning `PluginHooks::ERROR` prevents the event from running. The handler may also specify an error message by calling `$this->set_error()` with the error string.

For this event, aborting prevents the streaming server from stopping.

Sample Code

```
<?php
class PluginHooks_myplugin extends PluginHooks {

    public function install_hooks() {
        PluginHooks::register('pre-stop-server',array(&$this,'handle_prestopserver'));
    }

    /**
     * Handles the pre-stop-server event
     *
     * @param string $username the username of the account
     *
     * @return int returns PluginHooks::OK on success, PluginHooks::ERROR on error
     */
    public function handle_prestopserver($username) {
        try {
            /* implementation details here ... */
        } catch (Exception $e) {
            $this->set_error($e->getMessage());
            return PluginHooks::ERROR;
        }
        return PluginHooks::OK;
    }

}
```

6.8.27 Event: post-stop-server

Description

Called just after a streaming server is stopped.

Parameters

The following parameters are passed, in the order shown, to this script:

- **username** `(string)`
  the username of the account

Return Value

Returning `PluginHooks::OK` causes the event to proceed normally.

Returning `PluginHooks::COMPLETE` prevents any further event handlers from running and causes the event to proceed normally.

Sample Code

```
<?php
class PluginHooks_myplugin extends PluginHooks {

    public function install_hooks() {
        PluginHooks::register('post-stop-server',array(&$this,'handle_poststopserver'));
    }

    /**
     * Handles the post-stop-server event
     *
     * @param string $username the username of the account
     *
     * @return int always returns PluginHooks::OK
     */
    public function handle_poststopserver($username) {
        try {
            /* implementation details here ... */
        } catch (Exception $e) {
            /* swallow the exception */
        }
        return PluginHooks::OK;
    }

}
```

6.8.28 Event: pre-reindex

Description

Called just before a media library is reindexed (updated).

Parameters

The following parameters are passed, in the order shown, to this script:

- **username** (`string`)
  the username of the account

Return Value

Returning `PluginHooks::OK` causes the event to proceed normally.

Returning `PluginHooks::COMPLETE` prevents any further event handlers from running and causes the event to proceed normally.

Returning `PluginHooks::ERROR` prevents the event from running. The handler may also specify an error message by calling `$this->set_error()` with the error string.

For this event, aborting prevents the media library from updating.

Sample Code

```
<?php
class PluginHooks_myplugin extends PluginHooks {

    public function install_hooks() {
        PluginHooks::register('pre-reindex',array(&$this,'handle_prereindex'));
    }

    /**
     * Handles the pre-reindex event
     *
     * @param string $username the username of the account
     *
     * @return int returns PluginHooks::OK on success, PluginHooks::ERROR on error
     */
    public function handle_prereindex($username) {
        try {
            /* implementation details here ... */
        } catch (Exception $e) {
            $this->set_error($e->getMessage());
            return PluginHooks::ERROR;
        }
        return PluginHooks::OK;
    }

}
```

6.8.29 Event: post-reindex

Description

Called just after a media library is reindexed (updated).

Parameters

The following parameters are passed, in the order shown, to this script:

- **username** (string)
  the username of the account

Return Value

Returning PluginHooks::OK causes the event to proceed normally.

Returning PluginHooks::COMPLETE prevents any further event handlers from running and causes the event to proceed normally.

Sample Code

```
<?php
class PluginHooks_myplugin extends PluginHooks {

    public function install_hooks() {
        PluginHooks::register('post-reindex',array(&$this,'handle_postreindex'));
    }

    /**
     * Handles the post-reindex event
     *
     * @param string $username the username of the account
     *
     * @return int always returns PluginHooks::OK
     */
    public function handle_postreindex($username) {
        try {
            /* implementation details here ... */
        } catch (Exception $e) {
            /* swallow the exception */
        }
        return PluginHooks::OK;
    }

}
```

6.8.30 Event: pre-process-logs

Description

Called just before an account's logs are processed.

Parameters

The following parameters are passed, in the order shown, to this script:

- **username** (string)
  the username of the account

Return Value

Returning PluginHooks::OK causes the event to proceed normally.

Returning PluginHooks::COMPLETE prevents any further event handlers from running and causes the event to proceed normally.

Returning PluginHooks::ERROR prevents the event from running. The handler may also specify an error message by calling $this->set_error() with the error string.

For this event, aborting prevents the processing of the account's logs.

Sample Code

```
<?php
class PluginHooks_myplugin extends PluginHooks {

    public function install_hooks() {
        PluginHooks::register('pre-process-logs',array(&$this,'handle_preprocesslogs'));
    }

    /**
     * Handles the pre-process-logs event
     *
     * @param string $username the username of the account
     *
     * @return int returns PluginHooks::OK on success, PluginHooks::ERROR on error
     */
    public function handle_preprocesslogs($username) {
        try {
            /* implementation details here ... */
        } catch (Exception $e) {
            $this->set_error($e->getMessage());
            return PluginHooks::ERROR;
        }
        return PluginHooks::OK;
    }

}
```

6.8.31 Event: post-process-logs

Description

Called just after an account's logs have been processed.

Parameters

The following parameters are passed, in the order shown, to this script:

- **username** (`string`)
  the username of the account

Return Value

Returning `PluginHooks::OK` causes the event to proceed normally.

Returning `PluginHooks::COMPLETE` prevents any further event handlers from running and causes the event to proceed normally.

Sample Code

```
<?php
class PluginHooks_myplugin extends PluginHooks {

    public function install_hooks() {
        PluginHooks::register('post-process-logs',array(&$this,'handle_postprocesslogs'));
    }

    /**
     * Handles the post-process-logs event
     *
     * @param string $username the username of the account
     *
     * @return int always returns PluginHooks::OK
     */
    public function handle_postprocesslogs($username) {
        try {
            /* implementation details here ... */
        } catch (Exception $e) {
            /* swallow the exception */
        }
        return PluginHooks::OK;
    }

}
```

6.8.32 Event: pre-rotate-logs

Description

Called just before an account's logs are rotated.

Parameters

The following parameters are passed, in the order shown, to this script:

- **username** (`string`)
  the username of the account

Return Value

Returning `PluginHooks::OK` causes the event to proceed normally.

Returning `PluginHooks::COMPLETE` prevents any further event handlers from running and causes the event to proceed normally.

Returning `PluginHooks::ERROR` prevents the event from running. The handler may also specify an error message by calling `$this->set_error()` with the error string.

For this event, aborting prevents the rotation of the account's logs.

Sample Code

```
<?php
class PluginHooks_myplugin extends PluginHooks {

    public function install_hooks() {
        PluginHooks::register('pre-rotate-logs',array(&$this,'handle_prerotatelogs'));
    }

    /**
     * Handles the pre-rotate-logs event
     *
     * @param string $username the username of the account
     *
     * @return int returns PluginHooks::OK on success, PluginHooks::ERROR on error
     */
    public function handle_prerotatelogs($username) {
        try {
            /* implementation details here ... */
        } catch (Exception $e) {
            $this->set_error($e->getMessage());
            return PluginHooks::ERROR;
        }
        return PluginHooks::OK;
    }

}
```

6.8.33 Event: post-rotate-logs

Description

Called just after an account's logs have been rotated.

Parameters

The following parameters are passed, in the order shown, to this script:

- **username** (`string`)
  the username of the account

Return Value

Returning `PluginHooks::OK` causes the event to proceed normally.

Returning `PluginHooks::COMPLETE` prevents any further event handlers from running and causes the event to proceed normally.

Sample Code

```php
<?php
class PluginHooks_myplugin extends PluginHooks {

    public function install_hooks() {
        PluginHooks::register('post-rotate-logs',array(&$this,'handle_postrotatelogs'));
    }

    /**
     * Handles the post-rotate-logs event
     *
     * @param string $username the username of the account
     *
     * @return int always returns PluginHooks::OK
     */
    public function handle_postrotatelogs($username) {
        try {
            /* implementation details here ... */
        } catch (Exception $e) {
            /* swallow the exception */
        }
        return PluginHooks::OK;
    }

}
```

6.8.34 Event: pre-update-reseller

Description

Called just before a reseller account is updated.

Parameters

The following parameters are passed, in the order shown, to this script:

- **username** `(string)`
  the username of the account
- **password** `(string)`
  the password for the account
- **email** `(string)`
  the email address for the account
- **maxclients** `(int)`
  the listener limit for the account
- **maxbitrate** `(int)`
  the maximum bit rate for the account
- **transferlimit** `(int)`
  the data transfer limit for the account
- **diskquota** `(int)`
  the disk quota for the account
- **usesource** `(int)`
  1 if the reseller can create autoDJ-enabled accounts, otherwise 0
- **mountlimit** `(int)`
  the mount point limit for the account

Return Value

Returning `PluginHooks::OK` causes the event to proceed normally.

Returning `PluginHooks::COMPLETE` prevents any further event handlers from running and causes the event to proceed normally.

Returning `PluginHooks::ERROR` prevents the event from running. The handler may also specify an error message by calling `$this->set_error()` with the error string.

For this event, aborting prevents the update from being saved.

Sample Code

```
<?php
class PluginHooks_myplugin extends PluginHooks {

    public function install_hooks() {
        PluginHooks::register('pre-update-reseller',array(&$this,'handle_preupdatereseller'));
    }

    /**
     * Handles the pre-update-reseller event
     *
     * @param string $username the username of the account
     * @param string $password the password for the account
     * @param string $email the email address for the account
     * @param int $maxclients the listener limit for the account
     * @param int $maxbitrate the maximum bit rate for the account
     * @param int $transferlimit the data transfer limit for the account
     * @param int $diskquota the disk quota for the account
     * @param int $usesource 1 if the reseller can create autoDJ-enabled accounts, otherwise 0
     * @param int $mountlimit the mount point limit for the account
     *
     * @return int returns PluginHooks::OK on success, PluginHooks::ERROR on error
     */
    public function handle_preupdatereseller($username,$password,$email,$maxclients,$maxbitrate
        try {
            /* implementation details here ... */
        } catch (Exception $e) {
            $this->set_error($e->getMessage());
            return PluginHooks::ERROR;
        }
        return PluginHooks::OK;
    }

}
```

6.8.35 Event: pre-update-account

Description

Called just before a streaming account is updated.

Parameters

The following parameters are passed, in the order shown, to this script:

- **username** (string)
  the username of the account
- **password** (string)
  the password for the account
- **email** (string)
  the email address for the account
- **ipaddress** (string)
  the IP address for the account
- **port** (int)
  the port number for the account
- **maxclients** (int)
  the listener limit for the account
- **maxbitrate** (int)
  the maximum bit rate for the account
- **transferlimit** (int)
  the data transfer limit for the account
- **diskquota** (int)
  the disk quota for the account
- **usesource** (int)
  1 if the autoDJ is enabled, otherwise 0
- **mountlimit** (int)
  the mount point limit for the account

Return Value

Returning `PluginHooks::OK` causes the event to proceed normally.

Returning `PluginHooks::COMPLETE` prevents any further event handlers from running and causes the event to proceed normally.

Returning `PluginHooks::ERROR` prevents the event from running. The handler may also specify an error message by calling `$this->set_error()` with the error string.

For this event, aborting prevents the update from being saved.

Sample Code

```
<?php
class PluginHooks_myplugin extends PluginHooks {

    public function install_hooks() {
        PluginHooks::register('pre-update-account',array(&$this,'handle_preupdateaccount'));
```

```
    }

    /**
     * Handles the pre-update-account event
     *
     * @param string $username the username of the account
     * @param string $password the password for the account
     * @param string $email the email address for the account
     * @param string $ipaddress the IP address for the account
     * @param int $port the port number for the account
     * @param int $maxclients the listener limit for the account
     * @param int $maxbitrate the maximum bit rate for the account
     * @param int $transferlimit the data transfer limit for the account
     * @param int $diskquota the disk quota for the account
     * @param int $usesource 1 if the autoDJ is enabled, otherwise 0
     * @param int $mountlimit the mount point limit for the account
     *
     * @return int returns PluginHooks::OK on success, PluginHooks::ERROR on error
     */
    public function handle_preupdateaccount($username,$password,$email,$ipaddress,$port,$maxcli
        try {
            /* implementation details here ... */
        } catch (Exception $e) {
            $this->set_error($e->getMessage());
            return PluginHooks::ERROR;
        }
        return PluginHooks::OK;
    }

}
```

6.8.36 Event: post-update-reseller

Description

Called just after a reseller account is updated.

Parameters

The following parameters are passed, in the order shown, to this script:

- **username** (`string`)
  the username of the account
- **password** (`string`)
  the password for the account

- **email** (string)
  the email address for the account
- **maxclients** (int)
  the listener limit for the account
- **maxbitrate** (int)
  the maximum bit rate for the account
- **transferlimit** (int)
  the data transfer limit for the account
- **diskquota** (int)
  the disk quota for the account
- **usesource** (int)
  1 if the reseller can create autoDJ-enabled accounts, otherwise 0
- **mountlimit** (int)
  the mount point limit for the account

Return Value

Returning PluginHooks::OK causes the event to proceed normally.

Returning PluginHooks::COMPLETE prevents any further event handlers from running and causes the event to proceed normally.

Sample Code

```
<?php
class PluginHooks_myplugin extends PluginHooks {

    public function install_hooks() {
        PluginHooks::register('post-update-reseller',array(&$this,'handle_postupdatereseller'))
    }

    /**
     * Handles the post-update-reseller event
     *
     * @param string $username the username of the account
     * @param string $password the password for the account
     * @param string $email the email address for the account
     * @param int $maxclients the listener limit for the account
     * @param int $maxbitrate the maximum bit rate for the account
     * @param int $transferlimit the data transfer limit for the account
     * @param int $diskquota the disk quota for the account
     * @param int $usesource 1 if the reseller can create autoDJ-enabled accounts, otherwise 0
     * @param int $mountlimit the mount point limit for the account
     *
```

```
     * @return int always returns PluginHooks::OK
     */
    public function handle_postupdatereseller($username,$password,$email,$maxclients,$maxbitrat
        try {
            /* implementation details here ... */
        } catch (Exception $e) {
            /* swallow the exception */
        }
        return PluginHooks::OK;
    }

}
```

6.8.37 Event: post-update-account

Description

Called just after a streaming account is updated.

Parameters

The following parameters are passed, in the order shown, to this script:

- **username** (string)
  the username of the account
- **password** (string)
  the password for the account
- **email** (string)
  the email address for the account
- **ipaddress** (string)
  the IP address for the account
- **port** (int)
  the port number for the account
- **maxclients** (int)
  the listener limit for the account
- **maxbitrate** (int)
  the maximum bit rate for the account
- **transferlimit** (int)
  the data transfer limit for the account
- **diskquota** (int)
  the disk quota for the account

- **usesource** (int)
  1 if the autoDJ is enabled, otherwise 0
- **mountlimit** (int)
  the mount point limit for the account

Return Value

Returning PluginHooks::OK causes the event to proceed normally.

Returning PluginHooks::COMPLETE prevents any further event handlers from running and causes the event to proceed normally.

Sample Code

```
<?php
class PluginHooks_myplugin extends PluginHooks {

    public function install_hooks() {
        PluginHooks::register('post-update-account',array(&$this,'handle_postupdateaccount'));
    }

    /**
     * Handles the post-update-account event
     *
     * @param string $username the username of the account
     * @param string $password the password for the account
     * @param string $email the email address for the account
     * @param string $ipaddress the IP address for the account
     * @param int $port the port number for the account
     * @param int $maxclients the listener limit for the account
     * @param int $maxbitrate the maximum bit rate for the account
     * @param int $transferlimit the data transfer limit for the account
     * @param int $diskquota the disk quota for the account
     * @param int $usesource 1 if the autoDJ is enabled, otherwise 0
     * @param int $mountlimit the mount point limit for the account
     *
     * @return int always returns PluginHooks::OK
     */
    public function handle_postupdateaccount($username,$password,$email,$ipaddress,$port,$maxc
        try {
            /* implementation details here ... */
        } catch (Exception $e) {
            /* swallow the exception */
        }
        return PluginHooks::OK;
    }
```

```
}
```

6.8.38 Event: send-email

Description

Called when Centova Cast needs to send an email message (usually for notifications).

Parameters

The following parameters are passed, in the order shown, to this script:

- **name** (string)
  the name of the email template to send
- **recipients** (array)
  a list of recipient email addresses for the message
- **subject** (string)
  the subject for the message
- **variables** (array)
  a list of variables available to be populated into the message text
- **text** (string)
  the text/plain version of the message body
- **html** (string)
  the text/html version of the message body

Return Value

Returning `PluginHooks::OK` causes the event to proceed normally.

Returning `PluginHooks::COMPLETE` prevents any further event handlers from running and prevents the email from being sent by Centova Cast (useful if you use an external notification system).

Returning `PluginHooks::ERROR` prevents the event from running. The handler may also specify an error message by calling `$this->set_error()` with the error string.

For this event, aborting triggers an error and prevents the email from being sent by Centova Cast.

Sample Code

```
<?php
class PluginHooks_myplugin extends PluginHooks {

    public function install_hooks() {
```

```
        PluginHooks::register('send-email',array(&$this,'handle_sendemail'));
    }

    /**
     * Handles the send-email event
     *
     * @param string $name the name of the email template to send
     * @param array $recipients a list of recipient email addresses for the message
     * @param string $subject the subject for the message
     * @param array $variables a list of variables available to be populated into the message
     * @param string $text the text/plain version of the message body
     * @param string $html the text/html version of the message body
     *
     * @return int returns PluginHooks::OK on success, PluginHooks::ERROR on error
     */
    public function handle_sendemail($name,$recipients,$subject,$variables,$text,$html) {
        try {
            /* implementation details here ... */
        } catch (Exception $e) {
            $this->set_error($e->getMessage());
            return PluginHooks::ERROR;
        }
        return PluginHooks::OK;
    }

}
```

6.8.39 Event: pre-rename-account

Description

Called just before a streaming account is renamed.

Parameters

The following parameters are passed, in the order shown, to this script:

- **old_username** (`string`)
  the original (current) username of the account
- **new_username** (`string`)
  the new username to be assigned to the account

Return Value

Returning `PluginHooks::OK` causes the event to proceed normally.

Returning `PluginHooks::COMPLETE` prevents any further event handlers from running and causes the event to proceed normally.

Returning `PluginHooks::ERROR` prevents the event from running. The handler may also specify an error message by calling `$this->set_error()` with the error string.

For this event, aborting prevents the account from being renamed.

Sample Code

```
<?php
class PluginHooks_myplugin extends PluginHooks {

    public function install_hooks() {
        PluginHooks::register('pre-rename-account',array(&$this,'handle_prerenameaccount'));
    }

    /**
     * Handles the pre-rename-account event
     *
     * @param string $old_username the original (current) username of the account
     * @param string $new_username the new username to be assigned to the account
     *
     * @return int returns PluginHooks::OK on success, PluginHooks::ERROR on error
     */
    public function handle_prerenameaccount($old_username,$new_username) {
        try {
            /* implementation details here ... */
        } catch (Exception $e) {
            $this->set_error($e->getMessage());
            return PluginHooks::ERROR;
        }
        return PluginHooks::OK;
    }

}
```

6.8.40 Event: pre-rename-reseller

Description

Called just before a reseller account is renamed.

Parameters

The following parameters are passed, in the order shown, to this script:

- **old_username** (string)
  the original (current) username of the account
- **new_username** (string)
  the new username to be assigned to the account

Return Value

Returning PluginHooks::OK causes the event to proceed normally.

Returning PluginHooks::COMPLETE prevents any further event handlers from running and causes the event to proceed normally.

Returning PluginHooks::ERROR prevents the event from running. The handler may also specify an error message by calling $this->set_error() with the error string.

For this event, aborting prevents the account from being renamed.

Sample Code

```
<?php
class PluginHooks_myplugin extends PluginHooks {

    public function install_hooks() {
        PluginHooks::register('pre-rename-reseller',array(&$this,'handle_prerenamereseller'));
    }

    /**
     * Handles the pre-rename-reseller event
     *
     * @param string $old_username the original (current) username of the account
     * @param string $new_username the new username to be assigned to the account
     *
     * @return int returns PluginHooks::OK on success, PluginHooks::ERROR on error
     */
    public function handle_prerenamereseller($old_username,$new_username) {
        try {
            /* implementation details here ... */
        } catch (Exception $e) {
            $this->set_error($e->getMessage());
            return PluginHooks::ERROR;
        }
        return PluginHooks::OK;
    }

}
```

6.8.41 Event: post-rename-account

Description

Called just after a streaming account is renamed

Parameters

The following parameters are passed, in the order shown, to this script:

- **old_username** (string)
  the original (old) username of the account
- **new_username** (string)
  the new (current) username for the account

Return Value

Returning `PluginHooks::OK` causes the event to proceed normally.

Returning `PluginHooks::COMPLETE` prevents any further event handlers from running and causes the event to proceed normally.

Sample Code

```php
<?php
class PluginHooks_myplugin extends PluginHooks {

    public function install_hooks() {
        PluginHooks::register('post-rename-account',array(&$this,'handle_postrenameaccount'));
    }

    /**
     * Handles the post-rename-account event
     *
     * @param string $old_username the original (old) username of the account
     * @param string $new_username the new (current) username for the account
     *
     * @return int always returns PluginHooks::OK
     */
    public function handle_postrenameaccount($old_username,$new_username) {
        try {
            /* implementation details here ... */
        } catch (Exception $e) {
            /* swallow the exception */
        }
        return PluginHooks::OK;
```

```
    }

}
```

6.8.42 Event: post-rename-reseller

Description

Called just after a reseller account is renamed

Parameters

The following parameters are passed, in the order shown, to this script:

- **old_username** (string)
  the original (old) username of the account
- **new_username** (string)
  the new (current) username for the account

Return Value

Returning PluginHooks::OK causes the event to proceed normally.

Returning PluginHooks::COMPLETE prevents any further event handlers from running and causes the event to proceed normally.

Sample Code

```
<?php
class PluginHooks_myplugin extends PluginHooks {

    public function install_hooks() {
        PluginHooks::register('post-rename-reseller',array(&$this,'handle_postrenamereseller'))
    }

    /**
     * Handles the post-rename-reseller event
     *
     * @param string $old_username the original (old) username of the account
     * @param string $new_username the new (current) username for the account
     *
     * @return int always returns PluginHooks::OK
     */
    public function handle_postrenamereseller($old_username,$new_username) {
        try {
```

```
            /* implementation details here ... */
        } catch (Exception $e) {
            /* swallow the exception */
        }
        return PluginHooks::OK;
    }

}
```

6.8.43 Event: server-outage-restarted

Description

Called when a streaming server is restarted due to an outage

Parameters

The following parameters are passed, in the order shown, to this script:

- **username** (`string`)
  the username of the account

Return Value

Returning `PluginHooks::OK` causes the event to proceed normally.

Returning `PluginHooks::COMPLETE` prevents any further event handlers from running and causes the event to proceed normally.

Sample Code

```
<?php
class PluginHooks_myplugin extends PluginHooks {

    public function install_hooks() {
        PluginHooks::register('server-outage-restarted',array(&$this,'handle_serveroutageresta
    }

    /**
     * Handles the server-outage-restarted event
     *
     * @param string $username the username of the account
     *
     * @return int always returns PluginHooks::OK
     */
    public function handle_serveroutagerestarted($username) {
```

```
        try {
            /* implementation details here ... */
        } catch (Exception $e) {
            /* swallow the exception */
        }
        return PluginHooks::OK;
    }

}
```

6.8.44 Event: server-outage-restart-failed

Description

Called when a streaming server cannot be restarted after an outage

Parameters

The following parameters are passed, in the order shown, to this script:

- **username** (`string`)
  the username of the account

Return Value

Returning `PluginHooks::OK` causes the event to proceed normally.

Returning `PluginHooks::COMPLETE` prevents any further event handlers from running and causes the event to proceed normally.

Sample Code

```
<?php
class PluginHooks_myplugin extends PluginHooks {

    public function install_hooks() {
        PluginHooks::register('server-outage-restart-failed',array(&$this,'handle_serveroutage
    }

    /**
     * Handles the server-outage-restart-failed event
     *
     * @param string $username the username of the account
     *
     * @return int always returns PluginHooks::OK
     */
```

```
    public function handle_serveroutagerestartfailed($username) {
        try {
            /* implementation details here ... */
        } catch (Exception $e) {
            /* swallow the exception */
        }
        return PluginHooks::OK;
    }

}
```

6.8.45 Event: source-outage-restarted

Description

Called when an autoDJ is restarted due to an outage

Parameters

The following parameters are passed, in the order shown, to this script:

- **username** (`string`)
  the username of the account

Return Value

Returning `PluginHooks::OK` causes the event to proceed normally.

Returning `PluginHooks::COMPLETE` prevents any further event handlers from running and causes the event to proceed normally.

Sample Code

```
<?php
class PluginHooks_myplugin extends PluginHooks {

    public function install_hooks() {
        PluginHooks::register('source-outage-restarted',array(&$this,'handle_sourceoutageresta
    }

    /**
     * Handles the source-outage-restarted event
     *
     * @param string $username the username of the account
     *
     * @return int always returns PluginHooks::OK
```

```
     */
    public function handle_sourceoutagerestarted($username) {
        try {
            /* implementation details here ... */
        } catch (Exception $e) {
            /* swallow the exception */
        }
        return PluginHooks::OK;
    }

}
```

6.8.46 Event: source-outage-restart-failed

Description

Called when an autoDJ cannot be restarted after an outage

Parameters

The following parameters are passed, in the order shown, to this script:

- **username** (`string`)
  the username of the account

Return Value

Returning `PluginHooks::OK` causes the event to proceed normally.

Returning `PluginHooks::COMPLETE` prevents any further event handlers from running and causes the event to proceed normally.

Sample Code

```
<?php
class PluginHooks_myplugin extends PluginHooks {

    public function install_hooks() {
        PluginHooks::register('source-outage-restart-failed',array(&$this,'handle_sourceoutage
    }

    /**
     * Handles the source-outage-restart-failed event
     *
     * @param string $username the username of the account
     *
```

```
     * @return int always returns PluginHooks::OK
     */
    public function handle_sourceoutagerestartfailed($username) {
        try {
            /* implementation details here ... */
        } catch (Exception $e) {
            /* swallow the exception */
        }
        return PluginHooks::OK;
    }

}
```

6.8.47 Event: app-outage-restarted

Description

Called when a supplemental application is restarted due to an outage

Parameters

The following parameters are passed, in the order shown, to this script:

- **username** (`string`)
  the username of the account

Return Value

Returning `PluginHooks::OK` causes the event to proceed normally.

Returning `PluginHooks::COMPLETE` prevents any further event handlers from running and causes the event to proceed normally.

Sample Code

```
<?php
class PluginHooks_myplugin extends PluginHooks {

    public function install_hooks() {
        PluginHooks::register('app-outage-restarted',array(&$this,'handle_appoutagerestarted'))
    }

    /**
     * Handles the app-outage-restarted event
     *
     * @param string $username the username of the account
```

```
     *
     * @return int always returns PluginHooks::OK
     */
    public function handle_appoutagerestarted($username) {
        try {
            /* implementation details here ... */
        } catch (Exception $e) {
            /* swallow the exception */
        }
        return PluginHooks::OK;
    }

}
```

6.8.48 Event: app-outage-restart-failed

Description

Called when a supplemental application cannot be restarted after an outage

Parameters

The following parameters are passed, in the order shown, to this script:

- **username** (`string`)
  the username of the account

Return Value

Returning `PluginHooks::OK` causes the event to proceed normally.

Returning `PluginHooks::COMPLETE` prevents any further event handlers from running and causes the event to proceed normally.

Sample Code

```
<?php
class PluginHooks_myplugin extends PluginHooks {

    public function install_hooks() {
        PluginHooks::register('app-outage-restart-failed',array(&$this,'handle_appoutagerestart
    }

    /**
     * Handles the app-outage-restart-failed event
     *
```

```
     * @param string $username the username of the account
     *
     * @return int always returns PluginHooks::OK
     */
    public function handle_appoutagerestartfailed($username) {
        try {
            /* implementation details here ... */
        } catch (Exception $e) {
            /* swallow the exception */
        }
        return PluginHooks::OK;
    }

}
```

6.8.49 Event: bitrate-exceeded

Description

Called when a stream is broadcasting in excess of its configured bit rate limit.

Parameters

The following parameters are passed, in the order shown, to this script:

- **username** (`string`)
  the username of the account

Return Value

Returning `PluginHooks::OK` causes the event to proceed normally.

Returning `PluginHooks::COMPLETE` prevents any further event handlers from running and causes the event to proceed normally.

Returning `PluginHooks::ERROR` prevents the event from running. The handler may also specify an error message by calling `$this->set_error()` with the error string.

For this event, aborting prevents the bit rate limit from being enforced and allows the stream to continue broadcasting.

Sample Code

```
<?php
class PluginHooks_myplugin extends PluginHooks {

    public function install_hooks() {
```

```
        PluginHooks::register('bitrate-exceeded',array(&$this,'handle_bitrateexceeded'));
    }

    /**
     * Handles the bitrate-exceeded event
     *
     * @param string $username the username of the account
     *
     * @return int returns PluginHooks::OK on success, PluginHooks::ERROR on error
     */
    public function handle_bitrateexceeded($username) {
        try {
            /* implementation details here ... */
        } catch (Exception $e) {
            $this->set_error($e->getMessage());
            return PluginHooks::ERROR;
        }
        return PluginHooks::OK;
    }

}
```

6.9 Example Plugin

The following is a complete plugin that verifies that each new account is created using a strong password (specifically: that it contains at least one digit, one symbol, one uppercase letter, and one lowercase letter).

If the password appears to be weak, it the plugin instructs Centova Cast to refuse to create the account and return a descriptive error message.

```
<?php
/* save as /usr/local/centovacast/system/plugins/passwordstrength/hooks.php */
class PluginHooks_passwordstrength extends PluginHooks {

    public function install_hooks() {
        PluginHooks::register('pre-create-account',array(&$this,'handle_precreateaccount'));
    }

    /**
     * Handles the pre-create-account event
     *
     * @param string $username the username of the account
     * @param string $password the password for the account
     * @param string $email the email address for the account
     * @param string $ipaddress the IP address for the account
     * @param int $port the port number for the account
     * @param int $maxclients the listener limit for the account
     * @param int $maxbitrate the maximum bit rate for the account
     * @param int $transferlimit the data transfer limit for the account
     * @param int $diskquota the disk quota for the account
     * @param int $usesource 1 if the autoDJ is enabled, otherwise 0
     * @param int $mountlimit the mount point limit for the account
     *
     * @return int returns PluginHooks::OK on success, PluginHooks::ERROR on error
     */
    public function handle_precreateaccount($username,$password,$email,$ipaddress,$port,$maxcli
        try {
            if (!preg_match('/[A-Z]/',$password)) throw new Exception('Missing uppercase charac
            if (!preg_match('/[a-z]/',$password)) throw new Exception('Missing lowercase charac
            if (!preg_match('/[0-9]/',$password)) throw new Exception('Missing digits.');
            if (preg_match('/^[A-Za-z0-9]+$/',$password)) throw new Exception('Missing symbol c
        } catch (Exception $e) {
            $this->set_error('Weak password: must contain at least one digit, one symbol, and l
            return PluginHooks::ERROR;
        }
        return PluginHooks::OK;
    }

}
```

6.10 Plugins API

6.10.1 Introduction

Centova Cast v3 and up support a new plugin system allowing developers to integrate third-party functionality directly into Centova Cast.

6.10.2 Overview

Plugins are written in the PHP programming language. To extend Centova Cast using other programming languages, the Event Scripts interface should be used instead.

At the beginning of each request, all plugins register callbacks to be invoked when specific actions are performed by Centova Cast. When Centova Cast performs an action for which a callback is registered, it invokes the callback passing parameters containing the details of the event.

6.10.3 Environment

Note that Centova Cast v3 and up provides its own operating environment including a self-contained PHP engine, and does not rely on any existing PHP interpreter or settings on the server. As such, the list of supported extensions and functions may not match those which you have used on other PHP installations.

Most notably, for security reasons, Centova Cast's PHP engine has no notion of `exec()` or any other process execution functions, so it is not possible to launch external applications from within Centova Cast plugins.

6.11 Plugin Structure

6.11.1 Directory Layout

Each plugin is located in its own, unique directory under the `/usr/local/centovacast/system/plugins/` directory on the Centova Cast web interface server.

Within the plugin directory, a single file named `hooks.php` must exist. This file defines the events for which the plugin should be invoked, as described below.

Any other files created in the plugin directory are ignored by Centova Cast.

6.11.2 The hooks.php File

The `hooks.php` file must contain a PHP class named `PluginHooks_pluginname`, where `pluginname` matches the name of the plugin's directory. For example, to create a plugin named `foo`, one might create a file called `/usr/local/centovacast/system/plugins/foo/hooks.php` containing a class named `PluginHooks_foo`.

The `PluginHooks_pluginname` class must inherit from the `PluginHooks` class which is defined elsewhere by Centova Cast. For example:

```
class PluginHooks_foo extends PluginHooks {

}
```

To register itself with Centova Cast, the class must implement a method named `install_hooks()`, described in the next section. Aside from the `install_hooks()` method, Centova Cast does not impose any further structural requirements on the class and the implementation details are left up to the developer.

6.11.3 The install_hooks Method

Within the plugin class, the `install_hooks()` method is used to register callbacks which will be called by Centova Cast whenever a specific action is performed within Centova Cast. This method should call the `PluginHooks::register()` method once for each callback to be registered.

A typical `install_hooks()` method looks like:

```
public function install_hooks() {
    PluginHooks::register('eventname','callback_function');
    /* optionally, additional PluginHooks::register() calls here */
}
```

In the above example, the plugin asks Centova Cast to invoke a function named `callback_function()` every time an event named `eventname` is triggered.

The complete list of available events is provided in the Event Reference section of the plugin API documentation.

6.11.4 Return Values

A callback should typically return `PluginHooks::OK` to indicate that the event was handled successfully.

Some events accept return values, however, which allow the plugin to abort the execution of the event. (For example, the `pre-create-account` event allows you to return an error to stop the account from being created.)

For events that accept return values, a callback may return `PluginHooks::ERROR` to indicate that Centova Cast should not proceed with the event. The plugin may also optionally return an error message by calling `$this->set_error()`. Any such error message will be displayed in the web interface (when applicable) and in the Centova Cast event log.