NATIVECAST DUAL STREAMING SYSTEM

NativeCast Distribution Complete

WEB HLS + TwitCasting + B1 Site + Control Center + iTunes Playlist Timer

配布統合版 v2.0 / Complete Guide v5

重要: 前回配布パッケージの問題点と今回の修正

旧WebCore: sftp.exeを毎サイクル起動
→ 1本のssh.exeを維持し、Native SFTP v3で継続転送する方式へ変更。

旧設定例: list_size=8
→ 現行運用と一致する2秒TS x 20本、約40秒windowへ統一。

書き込み途中の公開リスク
→ .uploadingへ全量送信、size確認後にrenameして正式公開。

転送破損の切り分け不足
→ 全TS size照合、1/20 SHA-256 readback、local TS structure検証を追加。

playlist snapshotの一時Access Denied
→ SSHを切る前にlocal処理を3回 x 150ms再試行するHOTFIX。

TwitCoreのgeneric改名 / 再配布時の文字化け
→ 実運用安定版の日本語ファイル名と実行名を固定。Control Center互換を維持。

クレジット表記が「作曲/作曲者」
→ B1 / Web映像 / Twit映像を「製作者」に統一。内部Composer値は維持。

Playlist切替タイマー未収録
→ 曲終了後 PAUSE → Playlist Select → PLAY を厳守するv1.0.1 VERIFIEDを追加。

この配布版には実SFTP接続情報、秘密鍵、RTMP Stream Key、secure.clixml、実ログ、実HLS断片を含めません。配布用の匿名設定例だけを収録します。

0. 前書き - なぜこの仕組みを作ったのか

この仕組みの出発点は、まだ広く知られていないオリジナル楽曲を、できるだけ良い音で、止めずに届け続けたいという思いでした。私が制作・運営しているloves21.net / LOVE's21も、その楽曲と製作者に出会える場所を残すためのサイトです。NativeCastを作る前にも、POPOPOで配信を続けるため、手元の機材や音源の再生方法を何度も組み替えてきました。

POPOPOで配信を始めた頃は、素性のよく分からない中国製の外部サウンドインターフェースを使っていました。音源の再生には、そのインターフェースに接続したXperia 1 IVを使用し、音声をWindows上のAndroidエミュレーターへ取り込んで配信していました。

その後、Windows上のエミュレーターが不安定になったことをきっかけに、配信機材をiPadへ変更しました。同時に、サウンドインターフェースをZOOM AMS-24へ変更しました。この時期は、Xperia 1 IVからの音声をAMS-24のマイク入力用ジャック2本へ入れ、ZOOM側のミキサーを通してiPadから配信する構成でした。

さらに、楽曲ごとにチューニングを変えたいと考え、音源を再生する機材をXperia 1 IVからMacBook Airへ変更しました。再生プレイヤーには、一時期RealPlayerを使っていました。

その後、POPOPOアプリがMacでも利用できるようになると、ZOOM AMS-24のループバック機能を利用する構成へ移行しました。音源の再生プレイヤーもiTunesへ変更し、このMacBook Air・iTunes・AMS-24の組み合わせを、POPOPOでの配信を終えるまで使い続けました。

こうした運用の積み重ねが、現在の配信システムにつながっています。端末やアプリの都合に左右される部分を減らし、音声の取り込み、楽曲情報、配信、監視を自分で管理できる仕組みにしたいと考えました。Windowsで再生する音を、ツイキャスのような外部配信サービスと、自分で管理するWebサイトの両方へ送り続けるために制作したのがNativeCast Coreです。

0. 前書き（続き）

本書で重視しているのは、費用を抑えながら、楽曲を守り、音質と配信の継続を両立することです。高価な放送設備や最新のゲーミングPCを前提にせず、手元の少し古いWindows PC、低価格のレンタルサーバー、自分で管理できる小さなWebサイトを組み合わせ、個人でも継続できる仕組みを目指しています。

ここでいう「楽曲を守る」は、コピーを完全に防ぐDRMを意味しません。WebサーバーにWAVやFLACなどの元マスター音源を置かず、PCで再生した音を配信用のAAC-LCへ変換し、Web側にはHLSの短い配信断片だけを順次置きます。古い断片を残し続けない運用により、元音源ファイルを直接公開する方式とは区別します。ただし、再生中の音を録音することまでは防げません。

音声はWindowsの実出力をWASAPIループバックで取り込み、映像はハードウェアH.264へ任せてCPU負荷を抑えます。音声設定は旧構成では48kHzステレオ・AAC-LC 256kbps、2026年10月10日のAutoMaster RC1試験構成では320kbpsです。Web HLSとツイキャスRTMPを別のCoreに分離し、一方の障害がもう一方へ波及しにくい構造にしています。現行構成の詳細と検証範囲は、巻末の10月10日付追補に記載しています。

LOVE's21が届けたい中心にあるのは、個人や小さなコミュニティで生まれたオリジナル楽曲、発表の場が失われた作品、まだ広く知られていない製作者の音です。聴いた人が作品名や製作者を知り、別の作品を探したり、自分も創作を始めたりする。そのような出会いは、再生回数やランキングだけでは測れません。このサイトと配信システムは、その機会と作品への道筋を残すためのものです。

音源を一度公開して終わりにするのではなく、独自サイトを常設の入口にし、ライブ配信、Now Playing、曲順、製作者紹介、外部リンクをつなぐことで、作品に出会い続けてもらう場を作ります。ただし、配信の仕組みを用意することと、著作権や利用許諾を得ることは別です。公開する楽曲の権利、製作者の意思、各サービスの規約を尊重して運用します。

OBS StudioやFFmpegは、配信や音声・映像処理で広く使われる成熟したツールです。本システムでも途中段階でFFmpegを使用し、安定して動くところまで到達しました。そのうえで、長時間運用時の管理、障害の切り分け、配布のしやすさを考え、最終的に配信データ経路を専用NativeCast Coreへ集約しました。本書は、その試行と改善の経緯を、私自身の運用経験と確認結果に基づいて記録するものです。

0.1 独自WEBサイトを持つ意味 - オリジナル楽曲の「居場所」を自分で持つ

LOVE's21のトップページには、「ツイキャスで常時配信が難しいため、楽曲紹介ページでも聴けるようにしました。」という説明があります。この一文は、独自WEBサイトを持つ意味をよく表しています。外部配信サービスは、人が集まりやすく、発見される入口として非常に有効です。しかし、配信できる時間、表示方法、プレーヤー、広告、UI、仕様変更などはサービス側に左右されます。そこで、自分のWEBサイトにも聴ける場所を用意し、配信サービスと独自サイトを「どちらか一方」ではなく「二つの入口」として使います。独自サイトは、単なる予備プレーヤーではありません。曲名だけでなく、アーティスト、アルバム、作曲者、作品紹介、過去の曲順、制作者へのリンクなどを、配信の流れに合わせて整理できます。SNSの投稿は時間とともに流れていきますが、WEBサイトは作品の背景や制作者情報を積み上げて残せます。LOVE's21が目指すのは、知られている曲をもう一度同じように流すことだけではなく、「まだ知られていない曲へ偶然たどり着く入口」を増やすことです。ライブ配信で初めて耳にし、Now Playingで名前を知り、紹介ページで制作者を知り、その人の他の作品へ進む。この一連の導線を自分で設計できる点に、独自サイトの大きな価値があります。自作曲を公開する人にとっても同じです。配信プラットフォームへ作品を投稿するだけでなく、自分のサイトを「放送局 +曲目表 + 作品紹介 + 公式案内所」として持つことで、聴いている瞬間と作品情報を結び付けられます。これは小規模な制作者でも実現できる、新しい公開方法の一例です。□ 同じ音源を、外部配信サービスと自サイトへ別経路で届けられる。□ 外部サービスがOFFLINEでも、自サイトHLSが動いていればサイト側の音声配信を継続できる。□ サイトの見た目と配信エンジンを分離できるため、裏側のエンジンを更新しても視聴ページを維持しやすい。□ iPhone/iPad、Android、PCそれぞれに合わせた再生方法を用意できる。□ Now Playingやプレイリストを、配信サービスのプレーヤーとは独立して表示できる。□ 自分のURL・デザイン・説明文で案内でき、視聴者にとって入口が分かりやすい。設計思想「配信エンジン」「外部配信サービス」「独自WEBサイト」「操作パネル」を一体化しすぎないことが重要です。片方の故障や仕様変更を、別の部分へ連鎖させない設計を優先します。

0.2 低価格・音源保護・高音質・連続運転 - 4つを同時に成立させる

このシステムでは、音楽配信を始めるために高価な専用サーバーや最新PCを購入することを前提にしていません。私自身の運用PCは、FUJITSU FMVS75C3WH、Intel Core i5-8250U（4コア/8スレッド）、メモリ11.86GB、Intel UHDGraphics 620という構成です。Windows 11 Pro上で、D3D11ハードウェアデバイス、H.264 Hardware Encoder、Windows AAC Encoderが利用可能であることを実機診断してから設計しました。このPCは最新のハイエンド機ではありません。それでもNative HLSの長時間試験ではCore自体のCPU使用率はおおむね数%台、メモリも約100MB台で推移し、音声・映像drop=0を維持しました。重要なのは「高いPCを買う」ことではなく、手元のPCでハードウェアH.264が使えるかを事前診断し、映像処理をGPUへ任せ、音声・Mux・転送を軽量に保つことです。これにより、同程度以上のWindows PCを持っている人なら、まず試験から始められる敷居の低い構成にしています。WEBサーバーも同じ考え方です。私はスターサーバー系の低価格な契約を選び、必要以上に高いサーバーへ上げず運用しています。HLSは2秒ごとの小さなファイルを頻繁に更新するため、料金だけでなくSFTP/SSH、HTTPS、転送量、ファイル更新の安定性が重要です。2026年10月時点の現行「スターレンタルサーバー」公式料金では、ライトは長期契約時253円/月から、初期費用0円と案内されています。契約時期や旧サービス継続契約では料金体系が異なるため、本書では私個人の契約金額そのものではなく、「最安価格帯から必要条件を満たすものを選ぶ」という考え方を共有します。

安価であることは、品質を捨てることと同義ではありません。元音源は公開 WEB 領域へ置かず、配信時だけ AAC-LC 256 kbps / 48 kHz へ変換します。WEB では HLS、ツイキャスでは RTMP という別経路へ同じ品質方針で送り、どちらかが止まってももう一方まで一緒に倒れない構造にします。「守るために聴けなくなる」「高音質にしたため止まる」「安くしたため不安定になる」の三択にしないことが、この設計の核心です。

0.3 なぜ OBS を最終構成の中心に置かなかったのか

OBS Studio は、画面・カメラ・音声・複数シーンを GUI で管理し、各種配信サービスへ送る用途では非常に強力です。しかし今回のシステムは、配信者が画面を操作し続ける一般的なライブ番組よりも、「決められた音源・決められた映像・決められたメタデータを、長時間、静かに、自動で出し続ける専用装置」に近い性格を持っています。この用途では、OBS の豊富さが必ずしも利点だけになりません。

観点	OBS が得意なこと	今回の専用 Core で優先したこと
操作	GUI でシーンやソースを柔軟に変更できる	通常運転中は GUI 操作を不要にし、START/STOP/STATUS だけにする
画面構成	複数カメラ・画像・テロップを視覚的に組める	固定した B1-style 映像（曲情報を一定位置に描く配信用 Now Playing 画面）をコードで毎回同じ条件で描画する
自動運転	自動化も可能だがプロファイル・シーン・プラグイン管理が増える	単一用途の実行ファイルと設定ファイルだけで再現する
二系統配信	複数出力は構成次第で可能	WEB HLS とツイキャス RTMP を最初から別プロセスに分離する
メタデータ	プラグインやブラウザソースを組み合わせやすい	iTunes COM をコアが直接読み、映像生成と同期する
障害切分け	GUI・プラグイン・エンコーダ等の要素が増えやすい	音声、映像、Mux、Uploader、RTMP をログ値で追跡する

つまり OBS を外した理由は、性能不足ではなく「この用途では機能が多く、運用対象が増えるため」です。カメラ切替や番組制作が必要なら OBS を選ぶ方が自然です。固定型の 24 時間系ストリーム、専用表示、Now Playing、二系統独立配信を重視するなら、専用 Core の方が管理しやすいという判断です。

0.4 FFmpeg は途中で実際に使われ、安定化もできていた

WEB 配信の初期構成では、NativeCast Core が WASAPI 音声を AAC へ、B1-style 映像を H.264 へエンコードし、その H.264/AAC を FFmpeg へ渡していました。ここでいう B1-style（ビー・ワン・スタイル）は、本プロジェクト固有の呼び名です。LOVE'S21 の視聴ページ B1.html の情報設計をもとにした配信用固定映像で、854×480 の画面に、淡い桜色系の背景、回転する円盤状ロゴ、曲名、アーティスト名、アルバム名、作曲者、LIVE AUDIO 表示、loves21.net のクレジットを配置します。B1.html という WEB ページそのものを動画化しているのではなく、「同じ情報を配信映像として読みやすく再構成したレイアウト」を B1-style と呼んでいます。FFmpeg 側では再エンコードせず、主に HLS Mux の役割だけを担当させる構成でした。つまり「画質や音質を FFmpeg に任せる」のではなく、「すでに出来上がっている H.264/AAC を HLS へ包む」ために利用していました。

この方式は、HLS の規格検証や早期実装には非常に有効でした。途中では playlist の一時ファイル rename とアップローダー側のファイル参照が競合し、stream.m3u8.tmp の更新が失敗する現象が出ましたが、Uploader が配信中ファイルを直接つかまず、専用の staging snapshot を作ってから SFTP する方式へ変更することで、3 分試験を drop=0 の状態まで安定させることができました。

重要な事実
FFmpeg 版は「不安定だから捨てた」のではありません。競合を解決した後は安定して動きました。それでも最終的に外した

のは、長期運用と配布を考えた時に、専用 Core として外部実行ファイル依存を残したくなかったためです。

0.5 安定していた FFmpeg を、それでも外した理由

- 外部実行ファイルが 1 個増えるだけで、配布時にバージョン、配置場所、実行権限、PATH、更新差分を管理する必要が生まれる。
- Core 本体と FFmpeg が別プロセスになるため、停止順序、パイプ切断、異常終了、再起動の責任範囲が分かれる。
- HLS playlist/segment の生成タイミングを Core 内部から完全には把握できず、ログや Health 監視が二重になる。
- 「再エンコードはしていない」状態でも、HLS 生成の中心部分を外部ツールへ依存するため、「NativeCast だけで完結」という最終目標を満たせない。
- ツイキャス側はすでに Native RTMP 送信で独立していたため、WEB 側も同じ思想に揃えると、運用・診断・配布の統一感が上がる。
- 外部ツールのアップデートで挙動が変わる可能性を減らし、特定バージョンの FFmpeg を抱え込む必要をなくしたい。

そのため最終版では、H.264 AVCC→Annex-B、AAC raw→ADTS、PAT/PMT、PCR/PTS/DTS、188 byte TS packet、continuity counter、キーフレーム境界の segment 分割、m3u8 更新までを Core 内部へ実装しました。ここまでして初めて、タスクマネージャー上でも ffmpeg.exe が存在しない「完全 Native HLS 経路」になりました。

0.6 このサイト／システムの本当の価値 - 技術ではなく「聴く導線」を残す

最終的な価値は、「OBS を使わなかった」「FFmpeg を消した」ことそのものではありません。重要なのは、運用者が毎回複雑な操作をしなくても、同じ音質・同じ映像・同じ表示を長時間繰り返し再現でき、知られていないオリジナル楽曲が継続して誰かの耳へ届く場所を維持できることです。配信の入口を自サイトに置き、外部サービスも並行して利用し、どちらか一方の変更が全体を壊さない構造にしたことが技術面の成果です。そして、その技術を使って「聴いた人が作品名や制作者へたどり着ける」状態を作ったことが、サイトとしての成果です。

また、B1-style サイトの見た目を固定したまま、裏側の配信エンジンだけを FFmpeg 版から Native HLS 版へ差し替えられたことは、WEB 開発としても重要です。視聴者が見る UI と、配信者が管理するデータ経路を分けることで、改善のたびにサイト全体を壊す危険を減らせます。

0.7 本書で使う固有名称・専門用語 - ここだけ読めば後の章が分かる

ここから先には、配信や映像の世界では一般的でも、普段 PC を使うだけでは目にしない言葉が多く出てきます。難しい言葉を覚える必要はありません。「どの箱が何をするのか」が分かれば十分です。特に NativeCast / Core / Native / B1-style は本プロジェクト内での呼び方なので、最初に意味を固定します。

最も大切な区別は「Encode（エンコード）」と「Mux（マックス／多重化）」です。Encode は音や映像を AAC や H.264 のような圧縮形式へ変えること、Mux はすでに圧縮済みの音声と映像を、時間を合わせて 1 つの配信用コンテナへ詰めることです。FFmpeg 版の最終段階では、FFmpeg は主に Mux を担当し、Native 版ではその Mux 部分まで Core 内部へ移しました。

用語	やさしい意味	このシステムでは何をするか
NativeCast	本プロジェクトで制作した配信システム全体の呼び名。市販製品名ではありません。	Windows 上で音声取得、映像生成、エンコード、配信、監視を行う仕組みの総称です。
Core（コア）	配信の中心で動く実行プログラム。	WEB Core と TwitCasting Core を別々に動かし、片方の問題をもう片方へ広げにくくしています。
Native（ネイティブ）	この資料では「主要な配信経路を OBS/FFmpeg という別プロセスに任せない」という意味。	Windows API、Media Foundation、C#内部処理などを使って Core 側で直接処理します。すべてをゼロから自作した、という意味ではありません。

用語	やさしい意味	このシステムでは何をするか
B1.html	LOVE'S21 の視聴用 WEB ページのファイル名。	Now Playing、HLS 再生、プレイリスト、ツイキャス状態などを表示します。
B1-style	B1.html の情報設計を元にした配信用固定映像レイアウトという本プロジェクト固有名称。	854×480 画面に曲名、アーティスト、アルバム、作曲者、回転ロゴ、LIVE AUDIO、loves21.net 表記などを描画します。
HLS	HTTP Live Streaming。WEB でライブ音声・映像を配る方式。	長い 1 本の動画ではなく、短い TS ファイルと m3u8 の一覧表を順番に配ります。
m3u8	HLS で「次にどの短いファイルを再生するか」を書いたプレイリストファイル。	stream.m3u8 をブラウザが読み、そこに書かれた streamXXXX.ts を追いかけます。
Segment（セグメント）	HLS を数秒単位に分けた短いメディアファイル。	本構成では基本 2 秒ごとの TS を生成します。
MPEG-TS / TS	映像と音声を運ぶためのコンテナ形式。	HLS の各 2 秒ファイルとして使います。1 パケット 188 byte という決まりがあります。
RTMP	ライブ配信サービスへ映像と音声をリアルタイム送信する通信方式。	TwitCasting Core から配信サービスへ直接 publish する経路で使います。
SFTP	SSH の暗号化通信を使って安全にファイルを転送する方法。	WEB Core が m3u8/TS を WEB サーバーへ送るために使います。
SSH	離れたサーバーへ暗号化して接続する仕組み。	SFTP の土台。秘密鍵やユーザー名を使って認証します。
Stream Key	配信サービスが発行する秘密の配信用キー。	パスワードと同じ扱いです。PDF、HTML、ログ、配布 ZIP へ入れてはいけません。
WASAPI Loopback	Windows がスピーカーへ出している音そのものを取り込む仕組み。	iTunes 等で再生している音を仮想マイクのように Core へ渡します。
FIR	デジタル音声を整えるフィルタ方式の一つ。	この構成では音声処理チェーンの一部として使います。利用者が係数を触る必要はありません。
PCM	圧縮前の生のデジタル音声データ。	WASAPI で取り込んだ音が AAC になる前の段階です。
AAC-LC	広く使われる音声圧縮形式 AAC の標準的な方式。	本構成では 48 kHz stereo / 256 kbps を基本としています。
H.264	映像を小さなデータ量に圧縮する代表的な動画形式。	B1-style 映像を 854×480 / 30fps で配信用に圧縮します。
Codec（コーデック）	音や映像を圧縮・復元する方式や部品。	AAC は音声 Codec、H.264 は映像 Codec です。
Hardware Encoder	CPU だけでなく GPU 等の専用機能を使う映像エンコーダー。	PC 負荷を抑えながら H.264 を作るために利用します。
Media Foundation / MFT	Windows 標準の音声・映像処理基盤と、その変換部品。	H.264 ハードウェアエンコーダーの利用などに使います。
Encode（エンコード）	生の音や映像を AAC/H.264 などへ圧縮する処理。	音質・画質・ビットレートに直接関係します。
Mux / Muxer	圧縮済みの音声と映像を、時刻を合わせて配信用の器へ詰める処理。	Native HLS では AAC/H.264 を MPEG-TS へまとめます。再エンコードとは別処理です。
GOP	H.264 で、あるキーフレームから次のキーフレームまでのまとまり。	本構成では約 2 秒単位を意識し、HLS segment を安全に切りやすくします。
Keyframe / I-frame	それ単体から映像を復元し始められる基準フレーム。	HLS segment は基本的にキーフレーム境界から始めます。
AVCC / Annex-B	H.264 の NAL データの並べ方の違い。	エンコーダー出力 AVCC を、MPEG-TS で扱いやすい Annex-B へ変換します。
SPS / PPS	H.264 映像を解釈するための設定情報。	新しい segment の先頭でも正しくデコー

用語	やさしい意味	このシステムでは何をするか
		ドできるようキーフレーム前へ付けます。
ADTS	AAC 音声フレームに付ける小さなヘッダー形式。	MPEG-TS へ AAC を入れる時に必要な情報を付加します。
PAT / PMT	MPEG-TS の中に「どの番号が映像で、どの番号が音声か」を書く案内表。	各 segment 先頭で送出し、再生機がストリーム構成を理解できるようにします。
PTS / DTS / PCR	映像・音声を「いつ再生・処理するか」を合わせる時刻情報。	口と音がずれたり、時間が飛んだりしないための時計です。
Continuity Counter	MPEG-TS パケットの順番確認用カウンタ。	パケット抜けや順序異常を受信側が判断する手掛かりになります。
Now Playing	現在流れている曲名やアーティスト等の表示。	iTunes COM 等から取得し、映像・WEB ページ・ログへ反映します。
COM / iTunes COM	Windows アプリ同士が情報をやり取りする仕組みの一つ。	iTunes から現在曲のタイトル等を取得するために使います。
hls.js	HLS を JavaScript で再生するための一般的なライブラリ。	PC/Android など、ブラウザが HLS を直接扱わない場合の再生に使えます。
WebKit Native HLS	iPhone/iPad の Safari 系ブラウザが持つ HLS の標準再生機能。	iOS では hls.js よりブラウザ内蔵 HLS を優先します。
PWA	WEB サイトをホーム画面からアプリのように開ける仕組み。	通常 Safari と音声挙動が違う場合があるため、別途試験します。
Service Worker	PWA 等でキャッシュや通信を制御するバックグラウンド機能。	ライブ HLS を古い状態で固定しないよう、m3u8/TS の長期キャッシュを避けます。
HTA	HTML/CSS/JavaScript で作る Windows ローカルアプリ形式。	NativeCast Control Center の GUI に使っています。普通の WEB ページより PC 操作権限があります。
DPAPI	Windows ユーザーにひも付けて秘密情報を暗号化する仕組み。	TwitCasting の RTMP URL や Stream Key 保存に利用します。
Health Log	CPU、メモリ、キュー、drop、遅延などを定期記録した運転状態。	「音が出ているように見える」だけでなく内部が健康か判断します。
Queue（キュー）	処理待ちデータを一時的に並べる場所。	ネットワークが少し遅れても音声取得を止めないために使います。満杯が続くと危険です。
drop	処理が追いつかずデータを捨てた回数。	dropPCM/dropA/dropV は 0 を保つのが基本です。
overrun	入力や処理が追いつかずバッファがあふれる状態。	Health で False を維持するのが望ましいです。
Staging / Snapshot	配信中ファイルを直接アップロードせず、いったんコピーした安全な写し。	FFmpeg 版で playlist 更新競合を避けるために導入し、SFTP 安定化に役立ちました。
PID	Windows 上で動く各プロセスに付く番号。	Control Center が Core の稼働状態を確認する時に表示します。
Working Set / RAM	そのプログラムが実際に使っているメモリ量の目安。	長時間で増え続けないかを見ることでメモリリークの兆候を探します。
MIME Type	WEB サーバーが「このファイルは何種類か」をブラウザへ伝える情報。	m3u8 と ts を正しい MIME で返さないと再生できない環境があります。
Cache / TTL	ブラウザや中継がファイルをどのくらい保存して再利用するかの設定。	ライブ m3u8 を長くキャッシュすると古い曲を追い続けるため短くします。
JSON	設定や Now Playing 情報を、人とプログラムの両方が読みやすい形で書く形式。	metadata.json や設定ファイルで使います。
PHP / Endpoint	WEB サーバー上で動く小さなプログラムと、そのアクセス窓口。	data.php や tc_status.php が必要な JSON だけ返します。
API	プログラム同士が決められた形式で情報をやり取りする窓口。	配信状態取得等で使う場合があります。秘密情報をブラウザ側へ返さないようにしま

用語	やさしい意味	このシステムでは何をするか
		す。
Rollback	新版に問題が出た時、直前の安定版へ戻すこと。	本番更新前に必ず 1 世代前を残します。
Sanitize	人へ渡す前に接続先・鍵・ユーザー名・個人情報等を削除すること。	配布 ZIP/PDF/ログは必ず Sanitize してから共有します。

専門用語は、初回から全部暗記する必要はありません。困った時にこの表へ戻り、「これは音を圧縮する言葉か」「ネットへ送る言葉か」「WEB で再生する言葉か」を確認してください。後半の付録 D にも同じ考え方で、ログや試験時に使う言葉をまとめています。

目次

0. 前書き - なぜこの仕組みを作ったのか

0.7 本書で使う固有名称・専門用語

0.8 この冊子を一文で言うと

高価な設備を買わず、元マスター音源を WEB へ置かず、AAC-LC 256 kbps / 48 kHz の音質を維持し、WEB HLS とツイキャス RTMP を独立させて配信停止の連鎖を防ぎ、知られていないオリジナル楽曲を長く聴いてもらうための実践設計です。

1. 匿名化・配布時の基本方針

2. 完成したシステムの全体像

3. 良い所・推奨点

4. デメリット・トレードオフ

5. 実測済みの安定性と到達点

6. 次に必要なもの・今後の改善

7. 初心者向け：必要な PC・ソフト・サーバー

8. WEB 配信 Core - 全ファイル説明

9. ツイキャス Core - 全ファイル説明

10. NativeCast Control Center - 全ファイル説明

11. WEB サイト - 必要ファイルと役割

12. 初心者向け：最初から最後までの導入手順

13. 試験スキーム - BUILD/3 分/1 時間/障害/端末

14. 毎日の運用、停止、復旧、ロールバック

15. セキュリティと秘密情報の扱い

15.1 音源保護 - 「元ファイルを公開しない」ことを最優先にする

最も簡単で効果の大きい対策は、WAV、FLAC、AIFF などの元音源・マスター音源を WEB 公開フォルダへ置かないことです。本構成では、音楽アプリで再生した Windows 出力を Core が取り込み、配信用 AAC へ変換してから送ります。サーバーに必要なのはライブ再生用の m3u8 と短い TS segment であり、元ファイルそのものではありません。

HLS segment は暗号化 DRM ではありません。技術に詳しい人が配信中のデータを保存したり、再生音を録音したりすることは可能です。そのため「絶対に複製できない」とは説明しません。狙いは、最高品質の元マスターを URL で直接ダウンロードできる状態にしないこと、古い segment を長期間残さないこと、ディレクトリ一覧を無効にすること、秘密鍵・設定ファイルを WEB root へ置かないことです。

さらに強い保護が必要な場合は、HLS 暗号化、期限付き URL、認証、CDN のアクセス制御などを追加できます。ただし費用・複雑さ・端末互換性が増えます。本書の基本設計は、個人が低コストで継続できる範囲で「元音源を守る」「高音質で聴かせる」「配信を止めない」のバランスを取るものです。

16. 第三者が同じ仕組みを作るための設計図

17. 類似 WEB サイトの丸写し用構成例

18. 配布パッケージの作り方

19. 最終チェックリスト

付録 A. 設定ファイルの丸写しテンプレート

付録 B. WEB サイト補助ファイルの丸写しテンプレート

付録 C. ログ判定早見表

付録 D. 用語集

1. 匿名化・配布時の基本方針

本書は、人に渡しても接続先や個人環境を特定できないことを前提にしています。そのため、実際の運用ファイルから得た構造や手順は説明しますが、接続先を特定できる値はすべて一般化しています。

種類	本書での表記	配布時の扱い
SFTP ホスト	<SFTP_HOST>	実値を書かない
SFTP ユーザー	<SFTP_USER>	実値を書かない
SFTP 秘密鍵	<SSH_PRIVATE_KEY_PATH>	鍵そのものを絶対に添付しない
WEB 保存先	<REMOTE_WEBROOT>/live	実サーバーパスを書かない
RTMP URL	<TWITCAST_RTMP_URL>	実値を書かない
Stream Key	<TWITCAST_STREAM_KEY>	実値を書かない
Windows ユーザー	<WINDOWS_USER>	個人名を出さない
ドメイン	<YOUR_DOMAIN>	実ドメインを出さない
配信サービス ID	<TWITCAST_USER_ID>	実アカウントを出さない

配布前の最重要注意
現在動いている実運用 ZIP を、そのまま第三者へ渡してはいけません。設定画面の初期値、ソース中の fallback URL、README、絶対パスなどにプロジェクト固有の文字列が残っている可能性があります。配布専用版を別に作り、プレースホルダー化してから渡してください。

2. 完成したシステムの全体像

完成形は、1 つの巨大アプリではなく、役割ごとに分かれた 4 ブロックです。音源は同じでも、WEB 配信とツイキャス配信は別プロセスです。Control Center は両方を操作しますが、配信データそのものには入りません。

```
[Windows / iTunes]
    |
    +--> [WEB Core]
    |    WASAPI -> AAC-LC 256k / H.264 854x480 30fps
    |    -> Native MPEG-TS/HLS -> SFTP -> <YOUR_DOMAIN>/live
    |
    +--> [TwitCasting Core]
         WASAPI -> AAC-LC 256k / H.264 854x480 30fps
         -> Native RTMP -> TwitCasting

[WEB Site]
  Now Playing / Playlist / HLS Player / TwitCasting status

[Control Center]
  WEB: STATUS / START / STOP / LOG
  TWIT: STATUS / START / STOP / LOG
```

2.1 WEB Core

- Windows 既定出力を WASAPI Loopback で取り込む
- FIR 処理後 48 kHz stereo AAC-LC 256 kbps へエンコード

- B1-style（曲名・アーティスト・アルバム・作曲者・回転ロゴ・loves21.net 表記を固定配置した配信用映像）を Hardware H.264 854x480/30fps へエンコード
- Native MPEG-TS/HLS muxer で 2 秒 segment を生成
- SFTP uploader で WEB サーバーへ転送
- 通常運転は黒い DOS 画面を残さない
- 60 秒 Health 中心のログと SFTP ログを分離

2.2 TwitCasting Core

- WASAPI/FIR/AAC-LC 256 kbps
- Hardware H.264 854x480/30fps、GOP 60
- iTunes COM から Now Playing 取得
- Native RTMP で connect/handshake/publish/unpublish
- 配信時間を区切る運用では正常終了後に再 publish
- RTMP URL/Stream Key は暗号化設定へ保存
- START/STOP/STATUS/BUILD/3 分テストを分離

2.3 WEB Site

- HLS 再生
- Now Playing 表示
- プレイリスト表示
- ツイキャス LIVE/OFFLINE 表示
- iOS は WebKit Native HLS を優先
- PC/Android は hls.js を利用可能
- PWA/Service Worker はライブ HLS をキャッシュしすぎない
- デザインと配信 Core を別管理

2.4 Control Center

- HTA(HTML/CSS)でデスクトップ GUI 化
- WEB と TWIT を 2 枚のカードで表示
- STATUS/START/STOP/LOG だけに限定
- PID/メモリ/稼働時間を表示
- パネルを閉じても Core は止まらない

3. 良い所・推奨点

良い点	意味
障害分離	WEB とツイキャスが別プロセスなので、片側の停止や仕様変更をもう片側へ伝播させにくい。
外部 Mux 依存の削減	WEB HLS は Core 内部で MPEG-TS/HLS を生成するため、FFmpeg の配置・更新・プロセス管理が不要。
音質・映像品質の統一	両 Core とも同じ WASAPI/FIR/AAC と同じ H.264 系設定を基本にできる。
Now Playing の直接取得	iTunes COM を使い、音源 PC から曲情報を取得できる。
低い操作負荷	通常は Control Center だけで START/STOP/STATUS/LOG が完結する。

サイトを壊しにくい	B1-style 表示と配信エンジンを分離し、バックエンド更新時に UI を固定できる。
検証しやすい	BUILD→3 分→1 時間→本番という昇格手順が固定され、Health カウンタで判定できる。
配布しやすい思想	接続情報と Core ソースを分離し、秘密情報を ZIP へ入れない運用にできる。

3.1 推奨する運用

- WEB とツイキャスは統合プロセスにしない
- 本番 Core 更新とサイトデザイン変更を同日にしない
- 普段はサイレント起動、異常時だけ診断起動
- 安定版フォルダを 1 世代残す
- 秘密情報をソースコード・HTML・ログへ直書きしない
- HLS と RTMP はそれぞれ別の停止処理を通す

4. デメリット・トレードオフ

デメリット	何が困るか	対策
専用 Core の保守責任	FFmpeg/OBS のような巨大コミュニティに任せられず、Mux/RTMP 実装の不具合は自分で直す必要がある。	ソース・試験・ログを必ず残す。
Windows 依存	WASAPI、Media Foundation、iTunes COM、HTA など Windows 固有要素が多い。	Windows 専用システムとして割り切る。
iTunes COM 依存	利用する iTunes 環境によって COM が使えない場合がある。	fallback API または別メタデータ取得手段を用意する。
SFTP 更新頻度	2 秒 HLS は segment 更新が多く、サーバー I/O と SSH 転送が増える。	短い playlist、staging、古い TS 整理を徹底する。
ブラウザ差	iOS と PC/Android で HLS の扱いが異なる。	iOS Native HLS と hls.js を分岐する。
HTA の将来性	HTA は個人 PC では便利だが企業環境で制限されることがある。	START/STOP/STATUS を CLI に固定し、将来 WPF/WinUI へ移せるようにする。
自前 Mux の検証負荷	PAT/PMT/PCR/PTS/DTS 等のミスが再生不良につながる。	ffprobe/デコード試験と実端末試験を継続する。
長期運用	24/7 ではログ肥大、回線断、サーバー障害、Windows Update 等がある。	ログローテーション、Watchdog、ロールバックを追加する。

5. 実測済みの安定性と到達点

5.1 WEB Native HLS - 実測

実運用 PC で、Native HLS 版を連続運転したログでは、約 1 時間 19 分 21 秒の観測区間で 2,380 個の HLS segment が生成され、すべて 2.000 秒でした。Health は gap=0 / dropPCM=0 / dropA=0 / dropV=0 を維持しました。CPU は約 2.24〜2.89%、Working Set は約 103.5〜122.7 MB の範囲で、終盤は約 107 MB まで戻っており、観測区間では右肩上がりのメモリ増加は見られませんでした。

muxMax は一度 185 ms まで上がり slowMux=1 になりましたが、その後カウンタは増え続けず、音声・映像ドロップは 0 のままでした。ユーザー操作による実機確認では、WEB ページの再生開始ボタンから音が出るまで約 0.5〜1 秒以下でした。

5.2 TwitCasting Native RTMP - 実測

3 分試験ログでは、RTMP TCP 接続、handshake、publish、AAC/H.264 header 送信が成功し、Health は gap=0 / dropPCM=0 / dropA=0 / dropV=0、slowSend=0 でした。終了時も Unpublish.Success から disconnect へ進み、正常停止しています。CPU は約 1.7〜2.1%、Working Set は約 115〜117 MB の範囲でした。

> **実測値の読み方**
> これらは特定の Windows PC・GPU・回線・サーバーでの結果です。別の PC へ配布する時は「同じ数値になる」と考えず、必ずその PC で BUILD→3 分→1 時間試験をやり直してください。

6. 次に必要なもの・今後の改善

6.1 P0 - 配布版にするため必須

1. プロジェクト固有名・ドメイン・ユーザー・パスを完全に削除した Sanitized Source Package
2. ASCII 中心のファイル名へ統一した TwitCasting 配布版
3. 空の example config と初回 SETUP だけを配布し、実設定は利用者 PC で生成
4. Core/Control Center/Site Template の互換バージョン表
5. 旧安定版へ戻すロールバック手順と 1 世代バックアップ

6.2 P1 - 長期運転で推奨

6. Watchdog：プロセス停止、HLS 更新停止、SFTP 連続失敗、RTMP publish 失敗を検知
7. Windows タスクスケジューラによる自動起動
8. Health summary JSON を作り Control Center から見られるようにする
9. 通知：一定回数のエラー時だけ表示・メール等
10. ログ保存日数＋最大容量の二重ローテーション

6.3 P2 - 将来改善

11. metadata API schema 固定
12. SFTP uploader を Adapter 化し、将来 Origin/CDN へ差替可能にする
13. HTA を WPF/WinUI へ移植
14. Encoder/Muxer interface を整理し将来コーデックへ対応
15. サイトテンプレートをテーマ化して配布可能にする

7. 初心者向け：必要な PC・ソフト・サーバー

ここからは、IT 用語に慣れていない人が上から順に作業する前提で書きます。分からない項目を飛ばさず、1 つずつ確認してください。

7.1 Windows PC

実際にこのシステムを動かしている PC の実測仕様を下に示します。「この程度以上なら絶対に動く」という保証ではありませんが、少なくとも最新・高価格 PC が必須ではないことを示す実例です。購入前に、まず手元の Windows PC で BUILD と 3 分試験を行ってください。

項目	実運用 PC の値	初心者向けの見方
メーカー / 型番	FUJITSU FMVS75C3WH	一般的なノート PC 系。専用配信機ではありません。
OS	Windows 11 Pro 64-bit	Windows 10/11 64-bit を基本対象にします。
CPU	Intel Core i5-8250U / 4 cores / 8 threads	第 8 世代の省電力 Core i5。最新ハイエンド CPU ではありません。
メモリ	11.86 GB（8GB + 4GB DDR4-2133）	12GB 前後でも実運用できた実例です。
GPU	Intel UHD Graphics 620	内蔵 GPU。H.264 Hardware Encoder が利用できることを事前試験します。
ストレージ	約 472.8GB / 空き約 319.6GB（診断時）	HLS は短い一時ファイル中心なので巨大な動画保存領域は不要です。
ネットワーク	Ethernet 1Gbps / Wi-Fi 866.7Mbps link（診断時）	常時配信は有線 LAN を優先すると切断要因を減らせます。
音声	Windows 既定出力を WASAPI Loopback 取得	特別な外付け音声機器を必須にしていません。
Native 事前診断	D3D11 PASS / H.264 Hardware Encoder READY / Windows AAC READY	この 3 点を自分の PC でも確認してから本番へ進みます。

この実例から伝えたいのは「同じ型番を買う」ことではありません。Core i5-8250U + Intel 内蔵 GPU + 約 12GB RAM という、現在では特別に高価ではない構成でも成立したため、まず手元の PC を診断してから不足部分だけを考える、という順序で十分です。

必要なもの	確認方法	できなければ
Windows 10/11 64-bit	設定→システム→バージョン情報	古い 32-bit 環境は推奨しない
音声が PC から正常に出る	iTunes 等で曲を再生しスピーカーで聞く	先に Windows 音声設定を直す
iTunes COM	PowerShell で iTunes.Application COM を生成できる	COM 対応 iTunes を用意するか代替 metadata 手段を使う
Hardware H.264 MFT	3 分テストで Hardware H264 PASS を確認	GPU ドライバ更新、対応 PC を使用
Windows PowerShell 5.1	スタートで Windows PowerShell を開く	Windows 標準機能を確認
OpenSSH Client	PowerShell で ssh.exe -V	Optional Features で OpenSSH Client を追加

OpenSSH Client の確認

```
ssh.exe -V
```

バージョン文字が出れば OK です。「ssh.exe が見つからない」と出る場合は Windows の「オプション機能」から OpenSSH Client を追加してください。

iTunes COM の簡易確認

```
$i = New-Object -ComObject iTunes.Application
$i.CurrentTrack.Name
```

iTunes で曲を再生中に曲名が表示されれば、COM 取得の基本確認は OK です。エラーの場合は、利用している音楽アプリが iTunes COM を提供しているか確認してください。

7.2 WEB サーバー

私は「配信のためだけに高額な VPS や専用サーバーを契約する」のではなく、スターサーバー系の低価格なレンタルサーバーで必要条件を満たす構成を選びました。必要なのは価格の高さではなく、HTTPS、SFTP/SSH、HLS ファイルを安定して公開できること、そして想定する転送量に契約条件が合うことです。現行スターレンタルサーバーの公式料金ではライトが 253 円/月から（36 か月契約時）、初期費用 0 円です。料金は契約期間・時期・サービス世代で変わるため、申込時に公式料金と利用規約を確認してください。

- HTTPS で公開できる WEB 領域
- SFTP または SSH でファイルを書き込めるアカウント
- 公開用 live/ ディレクトリ
- m3u8 と ts を正しい MIME で返せること
- PHP を使う場合は PHP 実行環境
- 秘密鍵や SETTINGS を公開 WEB 領域へ置かない

7.3 ツイキャス側

- 配信に使用できるアカウント
- RTMP 配信用 URL
- Stream Key
- 利用規約・配信権・楽曲利用条件を利用者自身で確認

8. WEB 配信 Core - 全ファイル説明

以下は現行 WEB Core パッケージの全構成です。配布版ではファイル名は ASCII のまま維持することを推奨します。

ファイル	役割	いつ使う	初心者の扱い
BUILD_CHECK.bat	C# Core をビルドできるかだけ確認する入口	初回と更新時に実行	編集しない
HLS_UPLOAD.ps1	work_dir の m3u8/TS を staging snapshot し、SFTP でサーバーへ送る	Core から自動起動	通常編集しない
LOGS/README.txt	LOGS フォルダの説明	読むだけ	編集不要
MANIFEST_SHA256.json	配布ファイルの SHA-256 一覧。破損・改変確認用	配布時に更新	手作業編集せず再生成
NATIVE_HLS_VALIDATION.txt	Native HLS mux の offline 検証結果	参考資料	編集不要
NativeCastSiteCore.cs	WEB 配信エンジン本体。WASAPI/AAC/H.264/Native HLS	BUILD 時に exe へ変換	初心者は編集しない
README.txt	概要、通常運転、ログ、導入手順	最初に読む	配布版では固有情報を除去
RUN.ps1	BuildOnly/Test3Min/Live の共通ランチャー。Uploader も起動	bat/vbs から呼ばれる	通常編集しない
SETTINGS/README.txt	秘密設定フォルダの注意書き	読むだけ	編集不要
SETUP.bat	設定画面を開く入口	初回にダブルクリック	編集しない
SETUP.ps1	SFTP 設定 GUI、接続テスト、JSON 保存	SETUP.bat から呼ばれる	配布版の default は空にする
START.bat	サイレント起動を呼び出してすぐ閉じる	普段の開始	編集しない
START_DIAGNOSTIC.bat	DOS/PowerShell 画面を出した診断起動	不具合調査時のみ	普段は使わない
START_SILENT.vbs	黒い画面を出さず Live 起動	通常運転に推奨	編集しない
STATUS.ps1	RUNNING/STOPPED、PID、最新 Health、Upload Health を表示	STATUS.vbs から呼ばれる	編集しない
STATUS.vbs	黒い画面なしで STATUS 小窓を開く	状態確認	編集しない
STOP.bat	停止要求入口	通常停止	編集しない
STOP.ps1	STOP_REQUEST を作成して Core へ正常停止要求	STOP から呼ばれる	編集しない
STOP_SILENT.vbs	黒い画面なしで停止要求	通常停止に推奨	編集しない
TEST_3MIN.bat	3 分だけ本番同等経路で試験	初回・更新時に必須	編集しない
VERSION.json	バージョン・仕様情報	互換性確認	リリース時だけ更新

8.1 WEB Core が実行中に自動生成するもの

生成物	場所	意味	消してよいか
NativeCastSiteCore.exe	Core フォルダ	C#から BUILD された実行ファイル	停止中なら再 BUILD 可能
SETTINGS/NativeCastSiteCore.json	SETTINGS	SFTP や作業フォルダ等の実設定	配布禁止。消すと再 SETUP 必要
LOGS/NativeCastSiteCore_YYYYMMDD.log	LOGS	Core 本体 Health/曲変更/起動停止	古い日付はローテーション可

LOGS/NativeCastSiteCore_UPLOAD_YYYYMMDD.log	LOGS	SFTP 状態・エラー	古い日付はローテーション可
LOGS/NativeCastSiteCore_LAUNCHER_YYYYMMDD.log	LOGS	起動・終了・launcher error	古い日付はローテーション可
LOGS/NativeCastSiteCore_BUILD_ERROR_latest.txt	LOGS	BUILD 失敗時のみ	原因解決後は残してもよい
STOP_REQUEST	Core フォルダ	正常停止を知らせる一時ファイル	起動時/停止処理で管理
stream.m3u8 / stream*.ts	work_dir	ローカル HLS 生成物	停止中に削除可。実行中は触らない
_upload_stage 相当	work_dir 内	SFTP 用 snapshot	実行中は触らない

8.2 WEB 設定画面の入力欄 - 何を入れるか

欄	入力例(一般化)	意味	注意
PC 一時保存先	%TEMP%\NativeCastWeb-HLS	m3u8/TS を一時的に作る PC 内フォルダ	WEB 公開フォルダではない
SSH サーバー	<SFTP_HOST>	契約サーバーの SSH/SFTP ホスト名	実値は配布資料へ書かない
SSH ポート	22 または契約値	SSH 接続ポート	レンタルサーバー独自値の場合あり
SSH ユーザー	<SFTP_USER>	SFTP ログイン名	パスワードではない
秘密鍵ファイル	C:\Users\<WINDOWS_USER>\.ssh\id_ed25519	公開鍵.pub ではなく秘密鍵側	Core ZIP へコピーしない
サーバー保存先	<REMOTE_WEBROOT>/live	stream.m3u8/TS の公開先	WEB root 内の live 専用
音量倍率	1.0	通常音量倍率	まず 1.0 で試す

9. ツイキャス Core - 全ファイル説明

現行ツイキャス安定版はプロジェクト固有の日本語ファイル名を使用しています。本書では個人を特定しないため、役割を同じにした「配布版で推奨する generic name」で説明します。配布用 ZIP では文字化けを避けるため、ASCII 名への変更を強く推奨します。

配布版推奨名	役割	分類
TwitCore.cs	Native RTMP 配信エンジン本体	C# source
TwitCore_RUN.ps1	BuildOnly/Test3Min/Live の共通ランチャー	PowerShell
TwitCore_SETUP.bat / .ps1	RTMP URL/Stream Key 等を入力し、暗号化設定を保存	初回設定
TwitCore_BUILD_CHECK.bat	C# Core をビルド	初回・更新時
TwitCore_TEST_3MIN.bat	3 分 publish 試験	初回・更新時
TwitCore_START.bat	画面あり通常起動	診断寄り
TwitCore_START_SILENT.vbs	黒い画面なしで開始	普段の開始
TwitCore_STOP.bat / .ps1	STOP_REQUEST を作り正常 unpublish 要求	通常停止
TwitCore_STOP_SILENT.vbs	黒い画面なしで停止	普段の停止
TwitCore_STATUS.bat / .ps1	プロセス、CPU/RAM、最新ログを確認	状態確認
TwitCore_NOWPLAYING_2SEC.bat / .ps1	Now Playing polling を 2 秒へ調整	必要時のみ
TwitCore_THREAD_CHECK.ps1	Thread constructor の安全性チェック	開発/更新時
README.txt	操作説明	最初に読む
VERSION.json	エンジン仕様・バージョン	リリース管理
ENCODING_NOTES.txt	BAT/PS1/VBS 等の文字コード規則	配布品質
VISUAL_CHANGELOG.txt	映像描画の変更点	保守資料
LOGS/README.txt	ログ保存先説明	読むだけ
SETTINGS/README.txt	secure 設定を共有しない注意	読むだけ

9.1 ツイキャス Core が自動生成するもの

生成物	役割	注意
TwitCore.exe	C#から BUILD された実行ファイル	配布先 PC で生成する方式が安全
SETTINGS/TwitCore_secure.clixml	RTMP URL と Stream Key 等の暗号化設定	配布禁止。別 PC へコピーしても DPAPI の都合で使えない場合がある
LOGS/TwitCore_YYYYMMDD.log	RTMP、Health、Now Playing、停止ログ	Stream Key を出さない
LOGS/TwitCore_BUILD_ERROR_latest.txt	BUILD 失敗時の詳細	個人パスが含まれる場合は共有前に確認
STOP_REQUEST	正常停止要求	実行中に手で編集しない

9.2 設定時に入力する項目

順番	項目	何を入れるか	初心者向け注意

1	RTMP URL	配信サービスが案内する rtmp:// で始まる URL	WEB ページ URL ではない
2	Stream Key	配信サービスが発行する秘密のキー	絶対に人へ送らない
3	1 回の配信時間	通常はサービス側の制限より少し短め	0 は Core 側で区切らない
4	再開までの秒数	例:5	短すぎる無限再接続を避ける
5	楽曲情報更新秒	例:2	まず 2 秒でよい
6	metadata fallback URL	<YOUR_DOMAIN>/data.php 等	任意。COM 失敗時の予備
7	背景画像 URL	<YOUR_DOMAIN>/assets/player-art.png	公開可能な画像のみ

10. NativeCast Control Center - 全ファイル説明

ファイル	役割	初心者がすること
NativeCast_ControlCenter.hta	メイン GUI。WEB/TWIT の状態を 2 秒ごとに確認し、START/STOP/LOG を呼ぶ	普段はこのショートカットだけ使う
INSTALL_TO_DESKTOP.vbs	デスクトップへ HTA とショートカットを設置	最初に 1 回ダブルクリック
README.txt	初回設定の説明	困った時に読む
%APPDATA%\NativeCastControlCenter\control.ini	WEB/TWIT Core のフォルダパスだけ保存	秘密情報は入れない

10.1 初回設定

16. ZIP を展開する
17. INSTALL_TO_DESKTOP.vbs をダブルクリック
18. デスクトップの NativeCast Control Center を起動
19. 歯車ボタンを押す
20. WEB Core フォルダを BROWSE で選ぶ
21. TWIT Core フォルダを BROWSE で選ぶ
22. SAVE SETTINGS を押す
23. 両方 STOPPED なら START が有効、RUNNING なら STOP が有効になることを確認

Control Center は配信エンジンではない
Control Center を閉じても、すでに起動している Core は動き続けます。GUI に問題が起きても配信データ経路を巻き込まないための設計です。

11. WEB サイト - 必要ファイルと役割

現在の視聴ページは、HLS プレーヤー、Now Playing、Playlist、ツイキャス LIVE/OFFLINE 表示、iOS 向け処理を 1 ページへ集約する設計です。実運用 B1.html には固有 URL やアカウント ID があるため、本書では generic 構成で説明します。

ファイル	役割	扱い
index.html または player.html	視聴ページ本体。HLS Player、Now Playing、Playlist、状態表示	公開
data.php	Now Playing/Playlist JSON を返す endpoint	公開。ただし秘密情報は返さない
tc_status.php	TwitCasting が LIVE かどうかを JSON で返す endpoint	公開。API 鍵等は返さない
link_map.json	Artist/Album ごとの外部リンク対応表	任意
manifest.json	PWA 用 manifest	任意
sw.js	PWA Service Worker	任意。live HLS は長期 cache しない
assets/player-art.png	レコード/ロゴ等の画像	公開可能な権利のある画像
live/stream.m3u8	WEB Core が更新する HLS playlist	Core が自動更新
live/stream*.ts	WEB Core が更新する HLS segment	Core が自動更新・整理
data/metadata.json	第三者向け参考実装で Now Playing を受け渡す JSON	reference implementation
data/twit_status.json	第三者向け参考実装で LIVE/OFFLINE を受け渡す JSON	reference implementation

11.1 サーバーの推奨フォルダ構成

```
<WEB_ROOT>/
 index.html
 data.php
 tc_status.php
 link_map.json
 manifest.json
 sw.js
 assets/
  player-art.png
 data/
  metadata.json
  twit_status.json
 live/
  stream.m3u8
  streamXXXXXXXX.ts
```

live/を手動編集しない

live/stream.m3u8 と stream*.ts は配信中に Core が更新します。FTP ソフトや WEB 管理画面で同時に編集・移動・削除すると再生不良の原因になります。

12. 初心者向け：最初から最後までの導入手順

この章だけを上から順番に実行すれば、何を先にするか迷わない構成にしています。作業中にエラーが出たら、その先へ進まず、該当段階を直してください。

12.1 まず作業フォルダを作る

24. デスクトップまたは C:\NativeCast など、分かりやすい場所に新しいフォルダを作ります。
25. その中に WebCore / TwitCore / ControlCenter / SiteTemplate / Backup の 5 フォルダを作ります。

```
C:\NativeCast\
  WebCore\
  TwitCore\
  ControlCenter\
  SiteTemplate\
  Backup\
```

12.2 WEB Core を準備する

26. 配布用 WEB Core ZIP を C:\NativeCast\WebCore に展開します。ZIP の中身がさらに 1 つのフォルダに入っている場合は、そのフォルダの中に BUILD_CHECK.bat 等が見える位置まで確認します。
27. SETUP.bat をダブルクリックします。
28. PC 一時保存先は、よく分からなければ %TEMP%\NativeCastWeb-HLS とします。
29. SSH サーバー、SSH ポート、SSH ユーザー、秘密鍵ファイル、サーバー保存先を契約サーバーの情報に置き換えます。
30. 「SSH 接続テスト」を押し、書き込み・削除 OK が出ることを確認します。
31. 「設定を保存」を押します。
32. BUILD_CHECK.bat を実行し BUILD: PASS を確認します。
33. TEST_3MIN.bat を実行します。この時点では旧配信が同じ live/ へ書き込んでいないことを確認します。
34. 3 分後にログを確認し、gap/drop=0、SFTP error なし、Native HLS であることを確認します。
35. ブラウザでサイトを開き、HLS が再生できれば次へ進みます。

12.3 ツイキャス Core を準備する

36. 配布用 TwitCore ZIP を C:\NativeCast\TwitCore に展開します。
37. TwitCore_SETUP.bat を実行します。
38. RTMP URL を入力します。これはブラウザ URL ではなく rtmp:// で始まる配信用 URL です。
39. Stream Key を入力します。画面に見せたりメモを共有したりしないでください。
40. 1 回の配信時間、再開秒数、Now Playing 更新秒を入力します。迷った場合は配布 README の推奨値を使います。
41. TwitCore_BUILD_CHECK.bat を実行し PASS を確認します。
42. TwitCore_TEST_3MIN.bat を実行します。
43. 配信サービス側で映像・音声が出ることを確認します。
44. ログで handshake/publish、drop=0、unpublish 正常終了を確認します。

12.4 Control Center を設置する

45. Control Center ZIP を展開
46. INSTALL_TO_DESKTOP.vbs をダブルクリック
47. デスクトップショートカットを開く
48. 歯車→WEB/TWIT フォルダを選択→SAVE
49. STATUS で正しい状態が出ることを確認

12.5 本番へ切り替える

50. 旧 WEB 配信がある場合は STOP
51. 数秒待ち、旧プロセス/旧 Uploader が消えたことを確認
52. 新 WEB Core を START_SILENT
53. WEB 再生確認
54. TwitCore を START_SILENT
55. ツイキャス側再生確認
56. Control Center で両方 RUNNING を確認
57. 1 時間は設定やサーバーを触らず耐久観察

13. 試験スキーム - BUILD / 3 分 / 1 時間 / 障害 / 端末

13.1 Stage 0 - 事前診断

- [ ] Windows 音声が鳴る
- [ ] iTunes COM が曲情報を返す
- [ ] ssh.exe がある
- [ ] SFTP 鍵で書込み/削除できる
- [ ] Hardware H.264 が使える
- [ ] WEB サーバーが m3u8/ts を正しい MIME で返す

13.2 Stage 1 - BUILD

- [ ] 新フォルダへ展開
- [ ] 旧安定版は残す
- [ ] 設定を作る
- [ ] BUILD_CHECK 実行
- [ ] ERROR=0
- [ ] exe が生成される

13.3 Stage 2 - WEB 3 分試験

- [ ] Native HLS START
- [ ] FFmpeg=NONE / ffmpeg.exe なし
- [ ] HLS headers ready
- [ ] 2 秒 segment 継続
- [ ] Now Playing 追従
- [ ] gap=0
- [ ] dropPCM=0
- [ ] dropA=0
- [ ] dropV=0
- [ ] slowMux が連続増加しない
- [ ] SFTP error なし
- [ ] STOP 時に Mux STOP→正常停止

13.4 Stage 2 - TwitCasting 3 分試験

- [ ] TCP connected
- [ ] handshake PASS
- [ ] publish START
- [ ] RTMP headers sent
- [ ] Now Playing 追従
- [ ] gap/drop=0
- [ ] slowSend 増加なし
- [ ] Stream Key がログに出ない
- [ ] Unpublish.Success
- [ ] disconnect
- [ ] stopped normally

13.5 Stage 3 - 1 時間耐久

確認項目	合格目安	不合格例
gap/drop	0 を維持	増え続ける
Queue	通常 0 付近、max が上限に張り付かない	audio/media queue が詰まる
CPU	一定範囲で推移	時間とともに上がり続ける
RAM	増減し一定範囲	右肩上がり
slowMux/slowSend	単発は観察、連続増加しない	毎分増える
HLS segment	ほぼ設定秒数で継続	長さが乱れる/欠落
Now Playing	曲変更へ追従	更新停止
SFTP/RTMP	連続エラーなし	再接続ループ

13.6 Stage 4 - 障害注入

- [ ] LAN を短時間切断→復帰後に HLS/RTMP が回復するか
- [ ] SFTP 先を一時的に失敗させ、Core 音声 capture が drop しないか
- [ ] iTunes を一度閉じ、fallback または準備中表示になり、再起動後に戻るか
- [ ] ブラウザをバックグラウンド→復帰し live edge へ戻れるか
- [ ] PC をスリープさせる運用をするなら復帰後動作を確認

13.7 Stage 5 - 端末互換

- [ ] Windows Chrome/Edge
- [ ] Android Chrome
- [ ] iPhone Safari
- [ ] iPad Safari
- [ ] PWA 版を使う場合は PWA 単独試験
- [ ] Bluetooth/有線イヤホン切替
- [ ] 低速回線
- [ ] 複数同時視聴

14. 毎日の運用、停止、復旧、ロールバック

14.1 普段の操作

58. Control Center を開く
59. WEB/TWIT が RUNNING か見る
60. 必要なら LOG で最新 Health を見る
61. 停止時は必ず STOP を使う
62. 強制終了は正常停止できない時だけ

14.2 WEB 停止

STOP_SILENT.vbs または Control Center の STOP を使います。Core は STOP_REQUEST を検知し、Muxer/Uploader の終了処理を通して停止します。配信中に exe をタスクマネージャーから直接 Kill するより安全です。

14.3 ツイキャス停止

STOP を使い、可能な限り Unpublish→disconnect の順を通します。これによりサービス側へ「正常に配信終了した」状態を伝えやすくなります。

14.4 ロールバック

63. 直前安定版を Backup に残す
64. 新版を STOP
65. 新版プロセスが消えたことを確認
66. 旧安定版 START
67. ブラウザ/配信サービスで復旧確認
68. 新版ログと変更点を保存して原因調査

15. セキュリティと秘密情報の扱い

- Stream Key はパスワードと同じ扱い
- SSH 秘密鍵を ZIP や WEB root に入れない
- 実設定 JSON/CLIXML を人へ渡さない
- ログを共有する前に URL/パス/ユーザー名/曲名などを確認
- HTML の JavaScript へ秘密鍵や Stream Key を絶対に書かない
- Control Center はローカル PC 専用で外部公開しない
- サーバーの data endpoint は必要最小限の JSON だけ返す
- 配布版 source の default 値に本番接続先を残さない

本書の匿名化
この PDF 自体には、実サーバー名・実ドメイン・実ユーザー名・実 RTMP URL・Stream Key・SSH 秘密鍵パス・個人 Windows ユーザー名を記載していません。

16. 第三者が同じ仕組みを作るための設計図

16.1 データプレーンと操作プレーンを分ける

```
DATA PLANE
  Audio Capture -> Audio Encode -> Output Adapter
  Video Render  -> H.264 Encode -> Output Adapter
  Metadata Poll -> Frame State

WEB Output Adapter
  H.264/AAC -> Native TS/HLS -> SFTP

TWIT Output Adapter
  H.264/AAC -> RTMP publish

CONTROL PLANE
  SETUP / START / STOP / STATUS / LOG / Control Center
```

16.2 推奨スレッド

- Capture thread：WASAPI を最優先。Network 待ちをしない
- Audio encoder thread：bounded queue から AAC 化
- Video renderer/encoder：30fps clock で描画/H.264
- WEB mux/uploader：配信先 I/O を Capture から分離
- RTMP network：send/receive を分離
- Now Playing poller：失敗しても配信本体を止めない
- Health monitor：60 秒ごとに queue/drop/CPU/RAM/遅延を記録

16.3 Native HLS の最低要件

- AVCC NAL を Annex-B へ変換
- AAC raw へ ADTS 付与
- TS packet=188 bytes
- PAT/PMT を segment 先頭で送出
- PCR/PTS/DTS を 90kHz clock で整合
- PID ごとの continuity counter
- keyframe 境界で segment を切る
- EXTINF へ実時間
- playlist は segment 存在確認後に更新
- 古い TS は playlist から外れた後に安全に削除

16.4 RTMP の最低要件

- URL parse
- TCP connect
- RTMP handshake
- connect command
- createStream
- publish
- AAC/H.264 sequence header
- timestamp 単調増加
- server control/onStatus 監視

- unpublish→disconnect
- backoff 再接続
- 秘密情報をログへ出さない

16.5 PDF だけでなく Source Package を同梱する理由

Native MPEG-TS/HLS や RTMP 実装は数千行規模の C#で、紙面へ全行を貼ると初心者向け資料として読めなくなります。本書は「何を作り、どこへ置き、どう設定し、どう試験するか」を丸写しできる形にし、エンジン本体は匿名化済み source file として配布 ZIP へ同梱する前提です。第三者へ渡す時は、本 PDF + Sanitized Source Package の 2 点をセットにしてください。

17. 類似 WEB サイトの丸写し用構成例

この章は第三者向けの reference implementation です。現在の実サイト backend そのものではなく、同じ考え方を再現するための安全な最小例です。下記をコピーして、<YOUR_DOMAIN>等だけ自分の値へ置き換えます。

17.1 link_map.json

```
{
  "artists": {},
  "albums": {}
}
```

17.2 manifest.json

```
{
  "name": "My NativeCast Radio",
  "short_name": "NativeCast",
  "start_url": "/",
  "display": "standalone",
  "background_color": "#111318",
  "theme_color": "#111318",
  "icons": [
   {"src":"assets/player-art.png","sizes":"512x512","type":"image/png"}
  ]
}
```

17.3 data/metadata.json - 最初の仮データ

```
{
  "viewers": 0,
  "nowPlaying": {
    "title": "準備中",
    "artist": "-",
    "album": "-",
    "composer": "-"
  },
  "playlist": []
}
```

17.4 data.php - metadata.json を返すだけの安全な最小例

```
<?php
header('Content-Type: application/json; charset=utf-8');
header('Cache-Control: no-store, no-cache, must-revalidate, max-age=0');
$file = __DIR__ . '/data/metadata.json';
if (!is_file($file)) {
  echo json_encode([
    'viewers' => 0,
    'nowPlaying' => ['title'=>'準備中','artist'=>'-','album'=>'-','composer'=>'-'],
    'playlist' => []
  ], JSON_UNESCAPED_UNICODE);
  exit;
}
readfile($file);
```

17.5 data/twit_status.json

```
{ "is_live": false }
```

17.6 tc_status.php - LIVE/OFFLINE JSON を返す最小例

```
<?php
header('Content-Type: application/json; charset=utf-8');
header('Cache-Control: no-store, no-cache, must-revalidate, max-age=0');
$file = __DIR__ . '/data/twit_status.json';
if (!is_file($file)) {
 echo '{"is_live":false}';
 exit;
}
readfile($file);
```

実際の LIVE 判定を自動化する場合は、配信サービスの公式 API または利用規約に沿った方法で twit_status.json を更新してください。API 仕様は変更されるため、本書では秘密鍵や固定 endpoint を埋め込みません。

17.7 sw.js - HLS を長期キャッシュしない例

```
const CACHE = 'nativecast-static-v1';
self.addEventListener('install', e => self.skipWaiting());
self.addEventListener('activate', e => e.waitUntil(self.clients.claim()));
self.addEventListener('fetch', e => {
 const u = new URL(e.request.url);
 if (u.pathname.includes('/live/') || u.pathname.endsWith('.m3u8') || u.pathname.endsWith('.ts')) {
  e.respondWith(fetch(e.request, {cache:'no-store'}));
  return;
 }
 e.respondWith(fetch(e.request).catch(() => caches.match(e.request)));
});
```

17.8 player.html - 丸写しできる最小完成例

下の HTML は、Now Playing、Playlist、HLS 再生、ツイキャス LIVE/OFFLINE 表示を 1 ページで確認するための最小例です。デザインは簡素ですが、構造はそのまま利用できます。<TWITCAST_USER_ID>だけ自分の公開 ID へ置き換えます。

```
<!doctype html>
<html lang="ja">
<head>
<meta charset="utf-8">
<meta name="viewport" content="width=device-width,initial-scale=1">
<link rel="manifest" href="manifest.json">
<title>NativeCast Radio</title>
<script src="https://cdn.jsdelivr.net/npm/hls.js@1/dist/hls.min.js"></script>
<style>
body{font-family:system-ui,-apple-system,"Yu Gothic UI",sans-serif;background:#111318;color:#eee;margin:0}
main{max-width:760px;margin:auto;padding:24px}
.card{background:#1b1e25;border:1px solid #30343e;border-radius:16px;padding:18px;margin-bottom:14px}
button{padding:10px 16px;border:0;border-radius:10px;font-weight:700;cursor:pointer}
#tc{font-weight:700}.live{color:#ff5a72}.off{color:#9aa3b2}
li{margin:8px 0}.muted{color:#9aa3b2;font-size:13px}
</style>
</head>
<body>
<main>
 <h1>NativeCast Radio</h1>
 <div class="card">
```

```
  <div id="tc" class="off">TWITCAST: CHECKING</div>
  <h2 id="title">準備中</h2>
  <div id="artist">-</div>
  <div id="album" class="muted">-</div>
  <div id="composer" class="muted">-</div>
 </div>
 <div class="card">
  <audio id="audio" controls playsinline style="width:100%"></audio>
  <p class="muted">iPhone/iPad はブラウザ内蔵 HLS、PC/Android は hls.js を優先します。</p>
 </div>
 <div class="card">
  <h3>On-Air Playlist</h3>
  <ol id="playlist"></ol>
 </div>
</main>
<script>
const DATA_URL='data.php';
const STREAM_URL='live/stream.m3u8';
const TWITCASTING_USER='<TWITCAST_USER_ID>';
const audio=document.getElementById('audio');
let hls=null;
function initHls(){
 if(audio.canPlayType('application/vnd.apple.mpegurl')){
  audio.src=STREAM_URL+'?t='+Date.now();
 }else if(window.Hls && Hls.isSupported()){
  hls=new Hls({liveSyncDurationCount:1,maxLiveSyncPlaybackRate:1.5});
  hls.loadSource(STREAM_URL); hls.attachMedia(audio);
 }else{
  document.querySelector('.muted').textContent='このブラウザは HLS 再生に対応していません。';
 }
}
async function updateMeta(){
 try{
  const r=await fetch(DATA_URL+'?t='+Date.now(),{cache:'no-store'});
  if(!r.ok)return; const d=await r.json();
  const n=d.nowPlaying||{};
  document.getElementById('title').textContent=n.title||'準備中';
  document.getElementById('artist').textContent=n.artist||'-';
  document.getElementById('album').textContent='Album: '+(n.album||'-');
  document.getElementById('composer').textContent='Composer: '+(n.composer||'-');
  const list=document.getElementById('playlist'); list.innerHTML='';
  (d.playlist||[]).forEach(x=>{
   const li=document.createElement('li');
   li.textContent=(x.title||'-')+' / '+(x.artist||'-'); list.appendChild(li);
  });
 }catch(e){}
}
async function updateTc(){
 const el=document.getElementById('tc');
 try{
  const r=await fetch('tc_status.php?t='+Date.now(),{cache:'no-store'});
  const d=await r.json();
  if(d.is_live){el.textContent='TWITCAST: LIVE';el.className='live';return;}
 }catch(e){}
 el.textContent='TWITCAST: OFFLINE';el.className='off';
}
initHls(); updateMeta(); updateTc();
setInterval(updateMeta,2500); setInterval(updateTc,8000);
if('serviceWorker' in navigator){navigator.serviceWorker.register('sw.js').catch(()=>{});}
</script>
</body>
</html>
```

17.9 metadata_publisher.ps1 - Now Playing を自動更新する参考例

現在の実サイトの data.php 実装は本資料の入力ソースに含まれていないため、下記は第三者が同じ体験を再現するための参考実装です。iTunes COM から曲名を読み、metadata.json を作り、scp でサーバーへ送ります。3 秒ごとに SSH 接続する簡易方式なので、小規模・検証用です。本番では WEB Core の Uploader へ統合するか、persistent SFTP へ改善してください。

```
$HostName = '<SFTP_HOST>'
$Port = 22
$User = '<SFTP_USER>'
$Key = 'C:\Users\<WINDOWS_USER>\.ssh\id_ed25519'
$Remote = '<REMOTE_WEBROOT>/data/metadata.json'
$Local = Join-Path $env:TEMP 'nativecast_metadata.json'
$iTunes = New-Object -ComObject iTunes.Application
while ($true) {
 try {
  $t = $iTunes.CurrentTrack
  $obj = [ordered]@{
   viewers = 0
   nowPlaying = [ordered]@{
    title = [string]$t.Name
    artist = [string]$t.Artist
    album = [string]$t.Album
    composer = [string]$t.Composer
   }
   playlist = @()
  }
  $obj | ConvertTo-Json -Depth 5 | Set-Content $Local -Encoding UTF8
  & scp.exe -q -P $Port -i $Key $Local ("$User@$HostName`:$Remote")
 } catch {}
 Start-Sleep -Seconds 3
}
```

17.10 Web server の最低設定

- HTTPS を使用
- m3u8: application/vnd.apple.mpegurl
- ts: video/mp2t
- m3u8 は no-cache または非常に短い TTL
- live/は Service Worker 長期 cache から除外
- SETTINGS/LOGS/source/key は WEB root へ置かない

18. 配布パッケージの作り方

18.1 推奨 ZIP 構成

```
NativeCast_Distribution/
 01_WebCore/
  ... WEB Core files ...
 02_TwitCore/
  ... Twit Core files ...
 03_SiteTemplate/
  player.html
  data.php
  tc_status.php
  link_map.json
  manifest.json
  sw.js
  assets/player-art.png
  data/metadata.json
  data/twit_status.json
 04_ControlCenter/
  NativeCast_ControlCenter.hta
  INSTALL_TO_DESKTOP.vbs
  README.txt
 05_Docs/
  NativeCast_Complete_Guide.pdf
  QUICK_START.txt
  CHANGELOG.txt
 VERSION.json
 MANIFEST_SHA256.json
```

18.2 配布前 Sanitize チェック

- [ ] 実ドメインが 0 件
- [ ] 実 SSH host/user が 0 件
- [ ] 実 RTMP URL が 0 件
- [ ] Stream Key が 0 件
- [ ] Windows ユーザー名が 0 件
- [ ] 秘密鍵絶対パスが 0 件
- [ ] ログの楽曲名/個人名が 0 件
- [ ] source default は<PLACEHOLDER>
- [ ] SETTINGS フォルダに実設定がない
- [ ] 公開可能な画像だけ

18.3 ファイル名ルール

Windows ZIP の文字化け事故を避けるため、第三者配布版は ASCII ファイル名を基本にします。日本語説明は README/PDF 本文で行い、スクリプト名は WEB_START_SILENT.vbs、TWIT_STATUS.ps1 のように役割が分かる英数字へ統一すると安全です。

19. 最終チェックリスト

- [ ] WEB Core BUILD PASS
- [ ] WEB 3 分試験 PASS
- [ ] WEB 1 時間試験 PASS
- [ ] ffmpeg.exe 不使用を確認

- [ ] HLS 再生 PC
- [ ] HLS 再生 Android
- [ ] HLS 再生 iPhone/iPad
- [ ] Now Playing 追従
- [ ] SFTP error なし
- [ ] Twit Core BUILD PASS
- [ ] Twit 3 分試験 PASS
- [ ] RTMP publish/unpublish 正常
- [ ] Stream Key 非露出
- [ ] Control Center 両 Core 操作 OK
- [ ] 旧安定版 Backup あり
- [ ] 実設定/鍵を配布 ZIP へ入れていない
- [ ] PDF/README が匿名化されている
- [ ] 配布先ユーザーが SETUP→BUILD→TEST→START を再現できる

付録 A. 設定ファイルの丸写しテンプレート

A.1 WEB Core - NativeCastSiteCore.json

```
{
  "work_dir": "%TEMP%\\NativeCastWeb-HLS",
  "ssh_host": "<SFTP_HOST>",
  "ssh_port": 22,
  "ssh_user": "<SFTP_USER>",
  "ssh_key": "C:\\Users\\<WINDOWS_USER>\\.ssh\\id_ed25519",
  "remote_dir": "<REMOTE_WEBROOT>/live",
  "volume": 1.0,
  "segment_seconds": 2,
  "list_size": 8,
  "audio_bitrate": "256k",
  "sample_rate": 48000,
  "capture_mode": "nativecast_wasapi_aac_h264_native_hls",
  "transfer_mode": "sftp",
  "diagnostic_logging": false
}
```

初心者は手書きでこの JSON を作るより、SETUP.bat の GUI から保存する方が安全です。上記は内容を理解するためのテンプレートです。

A.2 TwitCasting Core - 設定は手書きしない

TwitCasting Core の secure.clixml は Windows DPAPI を使って暗号化保存するため、丸写し用の平文ファイルは作りません。必ず SETUP スクリプトを実行し、RTMP URL と Stream Key をその PC 上で入力してください。

項目	推奨入力
RTMP URL	<TWITCAST_RTMP_URL>
Stream Key	<TWITCAST_STREAM_KEY>
Session Minutes	サービス制限より少し短い値
Restart Delay	5 秒程度から開始
Poll Seconds	2 秒
Fallback Data URL	https://<YOUR_DOMAIN>/data.php
Logo URL	https://<YOUR_DOMAIN>/assets/player-art.png

付録 B. WEB サイト補助ファイルの丸写しテンプレート

第 17 章の内容をそのまま各ファイル名で保存してください。文字コードは UTF-8 を推奨します。PHP ファイルの末尾には不要な ?> を書かない方が、余計な空白出力を防げます。

保存するファイル	文字コード	最初に確認すること
data.php	UTF-8	ブラウザで開き JSON が出る
tc_status.php	UTF-8	{"is_live":false}等が出る
link_map.json	UTF-8	JSON 構文エラーなし
manifest.json	UTF-8	アイコンパスが存在

sw.js	UTF-8	live/が cache 除外
data/metadata.json	UTF-8	nowPlaying/playlist キーあり
data/twit_status.json	UTF-8	is_live boolean
assets/player-art.png	PNG	著作権/利用許可を確認

付録 C. ログ判定早見表

ログ語	意味	判断
gap=0	入力音声の時間欠落なし	良好
dropPCM=0	PCM queue drop なし	良好
dropA=0	AAC/media audio drop なし	良好
dropV=0	video drop なし	良好
overrun=False	queue overrun なし	良好
slowMux=0	HLS mux 遅延しきい値超過なし	良好
slowMux=1 で固定	単発遅延の可能性	経過観察
slowMux が増え続ける	Mux/I/O 負荷	要調査
slowSend=0	RTMP 送信遅延なし	良好
HLS headers ready	SPS/PPS/AAC 等準備完了	良好
handshake PASS	RTMP handshake 成功	良好
publish START	配信開始	良好
Unpublish.Success	正常終了通知	良好
SFTP ERROR	Upload 失敗	要調査
BUILD FAILED	C# compile 失敗	本番起動しない
NativeCast	本プロジェクトで制作した配信システム全体の呼び名。市販製品名ではありません。Windows 上で音声取得、映像生成、エンコード、配信、監視を行う仕組みの総称です。	
Core（コア）	配信の中心で動く実行プログラム。 WEB Core と TwitCasting Core を別々に動かし、片方の問題をもう片方へ広げにくくしています。	
Native（ネイティブ）	この資料では「主要な配信経路を OBS/FFmpeg という別プロセスに任せない」という意味。 Windows API、Media Foundation、C#内部処理などを使って Core 側で直接処理します。すべてをゼロから自作した、という意味ではありません。	
B1.html	LOVE'S21 の視聴用 WEB ページのファイル名。 Now Playing、HLS 再生、プレイリスト、ツイキャス状態などを表示します。	
B1-style	B1.html の情報設計を元にした配信用固定映像レイアウトという本プロジェクト固有	

	名称。 854×480 画面に曲名、アーティスト、アルバム、作曲者、回転ロゴ、LIVE AUDIO、loves21.net 表記などを描画します。	
HLS	HTTP Live Streaming。WEB でライブ音声・映像を配る方式。 長い 1 本の動画ではなく、短い TS ファイルと m3u8 の一覧表を順番に配ります。	
m3u8	HLS で「次にどの短いファイルを再生するか」を書いたプレイリストファイル。 stream.m3u8 をブラウザが読み、そこに書かれた streamXXXX.ts を追いかけます。	
Segment（セグメント）	HLS を数秒単位に分けた短いメディアファイル。 本構成では基本 2 秒ごとの TS を生成します。	
MPEG-TS / TS	映像と音声を運ぶためのコンテナ形式。 HLS の各 2 秒ファイルとして使います。1 パケット 188 byte という決まりがあります。	
RTMP	ライブ配信サービスへ映像と音声をリアルタイム送信する通信方式。 TwitCasting Core から配信サービスへ直接 publish する経路で使います。	
SFTP	SSH の暗号化通信を使って安全にファイルを転送する方法。 WEB Core が m3u8/TS を WEB サーバーへ送るために使います。	
SSH	離れたサーバーへ暗号化して接続する仕組み。 SFTP の土台。秘密鍵やユーザー名を使って認証します。	
Stream Key	配信サービスが発行する秘密の配信用キー。 パスワードと同じ扱いです。PDF、HTML、ログ、配布 ZIP へ入れてはいけません。	
WASAPI Loopback	Windows がスピーカーへ出している音そのものを取り込む仕組み。 iTunes 等で再生している音を仮想マイクのように Core へ渡します。	
FIR	デジタル音声を整えるフィルタ方式の一つ。 この構成では音声処理チェーンの一部として使います。利用者が係数を触る必要はありません。	
PCM	圧縮前の生のデジタル音声データ。 WASAPI で取り込んだ音が AAC になる前の段階です。	
AAC-LC	広く使われる音声圧縮形式 AAC の標準的な方式。 本構成では 48 kHz stereo / 256 kbps を基本としています。	
H.264	映像を小さなデータ量に圧縮する代表的な動画形式。 B1-style 映像を 854×480 / 30fps で配信用に圧縮します。	
Codec（コーデック）	音や映像を圧縮・復元する方式や部品。 AAC は音声 Codec、H.264 は映像 Codec です。	
Hardware Encoder	CPU だけでなく GPU 等の専用機能を使う	

	映像エンコーダー。 PC 負荷を抑えながら H.264 を作るために利用します。	
Media Foundation / MFT	Windows 標準の音声・映像処理基盤と、その変換部品。 H.264 ハードウェアエンコーダーの利用などに使います。	
Encode（エンコード）	生の音や映像を AAC/H.264 などへ圧縮する処理。 音質・画質・ビットレートに直接関係します。	
Mux / Muxer	圧縮済みの音声と映像を、時刻を合わせて配信用の器へ詰める処理。 Native HLS では AAC/H.264 を MPEG-TS へまとめます。 再エンコードとは別処理です。	
GOP	H.264 で、あるキーフレームから次のキーフレームまでのまとまり。 本構成では約 2 秒単位を意識し、HLS segment を安全に切りやすくします。	
Keyframe / I-frame	それ単体から映像を復元し始められる基準フレーム。 HLS segment は基本的にキーフレーム境界から始めます。	
AVCC / Annex-B	H.264 の NAL データの並べ方の違い。 エンコーダー出力 AVCC を、MPEG-TS で扱いやすい Annex-B へ変換します。	
SPS / PPS	H.264 映像を解釈するための設定情報。 新しい segment の先頭でも正しくデコードできるようキーフレーム前へ付けます。	
ADTS	AAC 音声フレームに付ける小さなヘッダー形式。 MPEG-TS へ AAC を入れる時に必要な情報を付加します。	
PAT / PMT	MPEG-TS の中に「どの番号が映像で、どの番号が音声か」を書く案内表。 各 segment 先頭で送出し、再生機がストリーム構成を理解できるようにします。	
PTS / DTS / PCR	映像・音声を「いつ再生・処理するか」を合わせる時刻情報。 口と音がずれたり、時間が飛んだりしないための時計です。	
Continuity Counter	MPEG-TS パケットの順番確認用カウンタ。 パケット抜けや順序異常を受信側が判断する手掛かりになります。	
Now Playing	現在流れている曲名やアーティスト等の表示。 iTunes COM 等から取得し、映像・WEB ページ・ログへ反映します。	
COM / iTunes COM	Windows アプリ同士が情報をやり取りする仕組みの一つ。 iTunes から現在曲のタイトル等を取得するために使います。	
hls.js	HLS を JavaScript で再生するための一般的なライブラリ。 PC/Android など、ブラウザが HLS を直接扱わない場合の再生に使えます。	
WebKit Native HLS	iPhone/iPad の Safari 系ブラウザが持つ HLS の標準再生機能。 iOS では hls.js よりブラウザ内蔵 HLS を優先します。	
PWA	WEB サイトをホーム画面からアプリのように開ける仕組み。 通常 Safari と音声挙動が違う場合があるため、別途試験しま	

	す。	
Service Worker	PWA 等でキャッシュや通信を制御するバックグラウンド機能。 ライブ HLS を古い状態で固定しないよう、m3u8/TS の長期キャッシュを避けます。	
HTA	HTML/CSS/JavaScript で作る Windows ローカルアプリ形式。 NativeCast Control Center の GUI に使っています。普通の WEB ページより PC 操作権限があります。	
DPAPI	Windows ユーザーにひも付けて秘密情報を暗号化する仕組み。 TwitCasting の RTMP URL や Stream Key 保存に利用します。	
Health Log	CPU、メモリ、キュー、drop、遅延などを定期記録した運転状態。「音が出ているように見える」だけでなく内部が健康か判断します。	
Queue（キュー）	処理待ちデータを一時的に並べる場所。ネットワークが少し遅れても音声取得を止めないために使います。満杯が続くと危険です。	
drop	処理が追いつかずデータを捨てた回数。dropPCM/dropA/dropV は 0 を保つのが基本です。	
overrun	入力や処理が追いつかずバッファがあふれる状態。 Health で False を維持するのが望ましいです。	
Staging / Snapshot	配信中ファイルを直接アップロードせず、いったんコピーした安全な写し。 FFmpeg 版で playlist 更新競合を避けるために導入し、SFTP 安定化に役立ちました。	
PID	Windows 上で動く各プロセスに付く番号。 Control Center が Core の稼働状態を確認する時に表示します。	
Working Set / RAM	そのプログラムが実際に使っているメモリ量の目安。 長時間で増え続けないかを見ることでメモリリークの兆候を探します。	
MIME Type	WEB サーバーが「このファイルは何種類か」をブラウザへ伝える情報。 m3u8 と ts を正しい MIME で返さないと再生できない環境があります。	
Cache / TTL	ブラウザや中継がファイルをどのくらい保存して再利用するかの設定。 ライブ m3u8 を長くキャッシュすると古い曲を追い続けるため短くします。	
JSON	設定や Now Playing 情報を、人とプログラムの両方が読みやすい形で書く形式。metadata.json や設定ファイルで使います。	
PHP / Endpoint	WEB サーバー上で動く小さなプログラムと、そのアクセス窓口。 data.php や tc_status.php が必要な JSON だけ返します。	
API	プログラム同士が決められた形式で情報を	

	やり取りする窓口。 配信状態取得等で使う場合があります。秘密情報をブラウザ側へ返さないようにします。	
Rollback	新版に問題が出た時、直前の安定版へ戻すこと。 本番更新前に必ず 1 世代前を残します。	
Sanitize	人へ渡す前に接続先・鍵・ユーザー名・個人情報等を削除すること。 配布 ZIP/PDF/ログは必ず Sanitize してから共有します。	

付録 D. 用語集

用語	やさしい意味
WASAPI Loopback	Windows が実際に出力している音を取り込む仕組み。
FIR	音声サンプル処理に使うフィルタ方式。
AAC-LC	一般的な AAC 音声方式。
H.264	映像圧縮方式。
MFT	Windows Media Foundation の変換/エンコード部品。
RTMP	ライブ配信サービスへ映像・音声を送るプロトコル。
HLS	m3u8 playlist と短い media segment で配信する方式。
MPEG-TS	HLS の segment でよく使われる 188 byte packet ベースの container。
PAT/PMT	MPEG-TS 内で Program と stream PID を知らせる表。
PCR/PTS/DTS	再生タイミングを合わせる時刻情報。
GOP	H.264 の keyframe 間隔のまとまり。
SFTP	SSH 上で安全にファイル転送する方式。
DPAPI	Windows ユーザーに紐づけて秘密データを保護する仕組み。
HTA	Windows で HTML/CSS/JavaScript をローカルアプリのように動かす仕組み。
Health	drop や queue、CPU/RAM 等の運転状態をまとめたログ。
Rollback	新版に問題があった時、直前安定版へ戻すこと。
Sanitize	公開前に秘密情報や個人情報を取り除くこと。

最後に

このシステムで最も重要なのは、特定のツール名ではなく「役割を分ける」「秘密情報を分ける」「試験を段階化する」「正常停止とロールバックを必ず用意する」という設計です。これを守れば、配信先やサイトデザインを変えても同じ運用思想を再利用できます。

END OF NATIVECAST COMPLETE GUIDE v3 / loves21.net

20. 現行仕様追補 - Distribution Complete v2.0

この追補の読み方

第1〜40ページは、前回版の本文をそのまま残しています。設計の経緯、用語、導入手順、試験思想を「当時の完成形」として保存するためです。2026-10-07時点で変更された箇所だけを、この追補で追加します。

したがって、旧本文とこの追補で数値・名称・運用方法が異なる場合は、現行運用については本追補を優先してください。旧本文は誤りとして削除するのではなく、NativeCastがどのように安定化されたかを残す履歴として扱います。

20.1 前回版から現行版へ切り替えた理由

項目	前回版で見えた課題	現行v2.0での扱い
WEBアップロード	短い周期でsftp.exeを何度も起動する方式は、長時間運用でプロセス/TCP churnが大きかった。	1本のssh.exeを維持し、そのstdin/stdout上でNative SFTP v3を継続。
HLS公開	書込み途中のTSやm3u8を読まれる余地があった。	一時名 .uploading へ全量送信し、size確認後にrenameして正式公開。
破損切分け	「生成時に壊れたのか」「転送で壊れたのか」を判定しにくかった。	全TSのsize照合、ローカルTS構造検証、20本に1本のSHA-256 readbackを追加。
playlist snapshot	ごく稀にローカル側Access Deniedが発生し、SSH再接続まで行っていた。	ローカルsnapshot処理を最大3回、150ms間隔で再試行。それでも失敗した時のみ既存RECOVER。
表示クレジット	「作曲」「作曲者」表記では、楽曲提供者の意向と合わないケースが出た。	B1/Web映像/Twit映像の見える表記を「製作者」へ統一。内部Composerキーは互換性のため維持。
運用自動化	iTunesプレイリストの時刻切替は手動だった。	Playlist Timer v1.0.1 VERIFIEDを追加。現在曲終了後にPAUSE→SELECT→PLAY。

20.2 旧本文の読み替え

- 旧本文中の「作曲者」は、現行配布版では表示上「製作者」と読み替えます。内部iTunes COMのComposer取得は変更していません。
- 旧本文の「今後の改善」には、現在すでに実装済みの項目があります。完了状況は本追補の27章で整理します。
- 旧本文のHLS保持数に関する例より、本追補の「2秒TS x 20本」を現行値として優先します。

21. WEB Core 現行仕様 - 2秒TS / 20本 / 8秒遅延

5秒TSも実運用試験しましたが、途切れが発生した時に「5〜6秒の無音」として聞こえるため、最終的に2秒TSへ戻しました。短いTSにすることで、1断片に問題が起きた場合の可聴欠損を小さくし、8秒程度の再生余裕の中で再送・復旧できる設計を優先しています。

項目	現行値	理由
HLS segment	2秒	1断片の欠損時間を短くし、キーフレーム周期GOP60と一致。
playlist保持	20本（約40秒）	8秒再生余裕に対し十分な履歴を確保しつつ、長時間TSを残さない。
再生目標	ライブ端から約8秒後ろ	低遅延と回線揺らぎ吸収の妥協点。
遅延上限目安	約20秒	遅れ過ぎた場合はプレイヤー側で追いつく。
音声	AAC-LC 256 kbps / 48 kHz	従来の音質方針を維持。
映像	H.264 854x480 / 30fps / GOP60	Hardware Encoderを利用しCPU負荷を抑える。

21.1 Atomic Publish と転送検証

TSは正式ファイル名へ直接書かず、まず一時名へ書き切ります。その後リモートsizeを確認し、完全なファイルだけ正式名へrenameします。m3u8も同様に、書込み途中の状態を視聴者へ見せないことを優先します。

> 処理順: local TS structure check → upload to temporary name → remote size verify → rename to final name → playlist publish → old TS cleanup

21.2 ローカルTS構造検証

送信前に188 byte境界、sync byte 0x47、PAT、TEI、adaptation field、PIDごとのcontinuity counterを検査します。これにより、PC内部で生成した時点から壊れていた場合と、転送後に異常が出た場合を分離できます。

22. Persistent SFTP と playlist snapshot HOTFIX

初期版では2秒ごとにsftp.exeを起動・接続・切断していました。Windows Firewall上では通信はALLOWでしたが、長時間運用で「アクセスが拒否されました。」が発生し、Uploaderが停止する弱点がありました。現行版では接続方式そのものを変更しています。

22.1 現行転送経路

1本のWindows標準 ssh.exe を -s sftp で起動し、そのバイナリstdin/stdout上でSFTP v3のINIT / OPEN / WRITE / CLOSE / REMOVE / RENAME等をPowerShell側から直接処理します。毎segmentごとにsftp.exeを起動しません。

保護	現行動作
SFTP応答待ち	5秒。1回の停止で20秒以上待ち続けない。
初回再接続	1秒から再試行。
正式公開	一時名→size確認→rename。
毎TS検証	local sizeとremote sizeを照合。
内容検証	20本に1本をSFTPで読み戻し、SHA-256比較。
再接続後	remote reconcileを行い、古いTSや残骸を整理。

22.2 playlist snapshotのAccess Denied対策

実運用では、SFTPそのものではなくローカルのplaylist snapshot処理で、数時間に1回程度のUnauthorizedAccessExceptionが観測されました。現行HOTFIXでは、この例外を即SSH障害として扱いません。

> 現行: snapshot処理を最大3回、150ms間隔で再実行 → 3回とも失敗した時だけ従来のSFTP RECOVERへ移行。

これにより、一瞬のファイルロックやウイルス対策ソフト等による局所的なアクセス競合で、正常なSSHセッションまで切断することを減らします。原因を特定の製品へ断定するものではありません。

23. 長時間検証結果 - 生成・転送破損は確認されず

現行構成では、短時間試験だけでなく複数日にわたる実運転ログを使って検証しました。

検証項目	観測結果
連続運転	2026-10-05 10:53頃から10-07 04:56頃まで約42時間。
Core Health	gap=0 / dropPCM=0 / dropA=0 / dropV=0 / overrun=Falseを維持。
最終付近CPU/RAM	CPUおおむね2〜3%、Working Set約90MB台。
TS upload	75,758本以上。
remote size	sizeOKがupload本数と一致。サイズ不一致なし。
local structure	localStructNG=0。ローカルTS構造異常なし。
SHA-256 readback	3,787回以上、hashRetry=0。内容不一致なし。
slowMux	約42時間で累計13回。最大mux約1.17秒だがdropは増加せず。

23.1 この結果から分かること

以前疑っていた「PCでTSが生成された時点から壊れる」「SFTP転送中にTSの内容が壊れる」という仮説は、今回の大量検証では支持されませんでした。もちろん全環境で絶対に起きないことを証明するものではありませんが、現行実運用での主原因としては優先度を下げられます。

23.2 slowMuxの扱い

slowMuxはゼロではありませんが、発生頻度は非常に低く、発生時も音声・映像dropは0のままでした。このため、PC/Androidで体感した頻繁な無音欠損を説明する主因とは考えていません。

24. 端末別再生方針とB1の位置付け

実機確認では、iPhone/iPadのWebKit Native HLSは安定している一方、PC/AndroidのHLSでは環境により無音欠損を体感することがありました。生成・転送側の検証が正常であることから、現行運用では「すべての端末で同じ経路を強制する」より、安定する再生経路を使い分けます。

端末	推奨経路	補足
iPhone / iPad	Native HLS	Apple系ブラウザ内蔵HLSを優先。現行実機では安定。
PC	TwitCastingを安定再生経路として推奨	B1 HLSも利用可能だが、環境によって無音欠損が出る場合がある。
Android	TwitCastingを安定再生経路として推奨	ブラウザ/hls.js/端末差が大きいため、無理にHLS一本化しない。

24.1 B1の音量操作

B1の音量操作はHLS再生時に有効です。PC/AndroidでTwitCastingを選ぶ場合は、その再生側の音量操作を使います。B1の操作パネルはiOS HLSやPC/AndroidでHLSを利用する場合のために残しています。

24.2 HLSはDRMではない

2秒TSを20本だけ保持する設計は、古い断片を長くサーバーへ残さないための対策です。しかしHLSを連続受信して録音する行為そのものを防ぐDRMではありません。元WAV/FLAC等をWEB rootへ置かないことが最優先です。

25. クレジット表記・TwitCore互換・Control Center

25.1 「作曲者」から「製作者」へ

一部の楽曲提供者からの要望を受け、現行配布版では、利用者から見えるクレジットを「製作者」へ統一しました。対象はB1、WEB配信用B1-style映像、TwitCasting配信用映像です。

一方、iTunes COMが持つComposerプロパティや内部JSONのcomposerキーは変更していません。表示名だけを変更することで、既存コードやAPI互換性を壊さず要望へ対応しています。

25.2 TwitCoreのファイル名は変更しない

TwitCasting
Coreは、安定版の日本語実行名・スクリプト名をそのまま維持します。genericな英字名への改名は、Control Centerのプロセス検出や既存運用との互換性を壊すため行いません。

再パッケージ時に文字化けが発生した経験から、TwitCoreは元ZIPの文字コード・日本語ファイル名を保持し、必要な表示文字列だけを変更する方針に固定しました。

25.3 Control Center

Control Center v1.0はWEB/TwitのSTATUS / START / STOP / LOGだけを担当し、配信データ経路には介入しません。現行TwitCoreの実行名を前提にしているため、TwitCore側の名前固定はControl Center互換の一部です。

26. iTunes Playlist Timer v1.0.1 VERIFIED

時刻によるプレイリスト変更を追加しました。ただし「指定時刻になったら再生中の曲を途中で切る」動作は禁止し、現在曲が自然に終了した後だけ切り替えます。

```
厳守順序: 指定時刻到達 → 現在曲終了待ち → PAUSE → PLAYLIST SELECT → PLAY
```

26.1 v1.0からv1.0.1へ修正した理由

- 初版ではPowerShellの自動変数 $PID と同名の $pid

を使い、現在曲取得が失敗する可能性がありました。v1.0.1では変数名衝突を除去しています。

- PlayerStateだけを見て即「曲終了」とみなさず、対象トラックの遷移を監視する方向へ修正しました。
- 一時停止中は勝手にプレイリスト変更せず、再生が進み対象曲が終了するまで待つ設計です。

26.2 運用上の注意

iTunesのクロスフェードを有効にすると「曲が完全に終了した瞬間」が曖昧になるため、厳密な曲境界で切り替えたい場合はクロスフェードOFFを推奨します。

27. Distribution Complete v2.0 - 現行配布構造

前回版の「WEB Core / Twit Core / Site / Control Center」という分離思想はそのままです。v2.0ではそこへPlaylist Timerと、現行仕様に合わせた配布・検証資料を追加しました。

フォルダ	内容
01_WebCore	現行WEB Core、Persistent SFTP、Atomic Publish、検証機能、playlist retry HOTFIX。
02_TwitCore	日本語名を維持した安定版Native RTMP Core。「製作者」表示対応。
03_Site_B1	現行B1、Now Playing、HLS、Twit状態、補助ファイル。
04_ControlCenter	WEB/Twitの状態・開始・停止・ログをまとめるHTA。
05_iTunes_Playlist_Timer	v1.0.1 VERIFIED。曲終了後のPlaylist変更を自動化。
06_Docs	本ガイド、移行理由、CHANGELOG、QUICK START、検証要約。
07_UnitPackages	各ユニットを個別再配布できるZIP/HTML。

27.1 旧版から移行する時の最小手順

- 旧安定版を別フォルダへ残してから、新しい配布ZIPを展開する。
- 実SFTP設定、秘密鍵、RTMP URL/Stream Key等は配布ZIPへコピーせず、各PCで安全に設定する。
- WEB設定は segment_seconds=2 / list_size=20 を確認する。
- B1、Web/Twit映像のクレジット表示が「製作者」になっていることを確認する。
- 3分試験 → 1時間試験 → 実運用の順で昇格する。

27.2 現行版として優先する値

2秒TS / 20本 / 約8秒再生余裕 / AAC-LC 256kbps 48kHz / H.264 854x480 30fps GOP60 / Persistent Native SFTP v3 / Atomic Publish / size verify EVERY_TS / SHA-256 readback 1/20 / localStructNG監視。

28. 2026年10月10日 現行構成と検証実績

追補の読み方

本追補は、既存48ページを残したうえで、2026年10月10日の実PC作業と確認結果を追加したものです。旧本文は開発履歴として参照し、設定や端末別再生方針が異なる場合は本追補を優先してください。今回の構成は「NativeCast v3.0 AutoMaster RC1 - JUN FIELD TEST」です。公開済み配布ZIPすべての内容を、この試験構成へ更新したという意味ではありません。

運用期間と連続運転を区別する

今回の基になったコードは、7日間連続配信したエンジンコードです。この実績はJUN氏の運用確認に基づくものです。ChatGPTが今回参照できた旧ツイキャスログは10月3日から10月10日までの約7日間で、導入作業に伴う停止・再起動も含みます。基となったエンジンの連続配信実績と、今回参照したログの範囲・導入後の試験期間は区別します。

既存の検証実績

旧本文44ページの約42時間連続運転の記録は、その時点の構成の検証結果として残します。今日導入したAutoMaster RC1の連続運転実績へ置き換えて数えることはしません。RC1の長期安定性の確認は10月10日の導入時点から始まります。

10月10日の観測結果

導入時のWeb公開監視（約3分）でプレイリスト更新が継続し、ローカルから公開までの観測上の遅れは0〜2秒でした。ツイキャスは利用者が正常な再生を確認しました。10:12頃のツイキャスHEALTHログではgap/dropPCM/dropA/dropVが0、overrun=False、CPU約3%、ワーキングセット約117MBでした。いずれも観測した範囲の結果です。

未検証の範囲

全曲の音量補正、全端末・全回線での320kbps再生、数日間のRC1連続運転まで保証したものではありません。内部処理に短い遅延が記録されたこともあり、欠落が0だった事実と処理時間の変動を分けて評価します。

29. なぜ変更が必要になったのか

端末で無音・音切れ・表示停止が発生

B1.htmlでAndroidの表示遅延や無音、iPhoneの再生開始遅延と音切れが報告されました。その後の端末向け修正で、両端末とも表示と音切れが改善したと利用者が確認しています。旧本文45ページの「Androidはツイキャスを推奨」という当時の方針だけでは、現在のWeb再生とアプリ配布の状態を説明できなくなりました。

一つの原因に断定しない

問題は、生成側の音声・映像時刻、HLSプレイリストの連続性、転送・公開のタイミング、端末のバッファや再接続処理、PC側の資源使用を合わせて検討する必要がありました。PC負荷だけ、あるいはブラウザだけを根本原因として断定する証拠はありません。生成・転送が正常でも端末で再生が止まる場合があります。

長時間運転で資源が残らないようにする

ツイキャスのセッション更新でエンコーダー関連資源を適切に解放する修正を引き継ぎました。再接続を重ねた際のメモリ増加を抑える目的です。PCの不要な常駐処理も整理しましたが、PCの緩慢さの全原因が解消した、または音切れの原因が常駐ソフトだったとは断定しません。

RC1をそのまま上書きしなかった理由

提供されたRC1パッケージには、実PCで蓄積した現在の修正がすべて含まれていませんでした。そこで、7日間連続配信したエンジンコードを基礎とし、実PCの修正済みソースへAutoMaster・曲識別・320kbps設定を組み込みました。既存の安定化処理を失わずに新機能を試すための導入方法です。

運用上の見落としを減らす

異常停止や再開に気付く仕組み、統一された操作画面、日本語の説明、曜日の選択を求められました。配信機能だけでなく、運用する人が状態を理解し、設定を間違えにくい環境を整える必要がありました。

30. 音声エンジンとAutoMasterの変更

320kbps試験構成

Webとツイキャスの音声設定をAAC-LC、48kHzステレオ、320kbpsへ変更しました。旧本文の256kbpsは旧構成の設定です。320kbps化で音声データ量は約25%増えますが、元音源の情報が増えることや、すべての回線・端末で音切れがなくなることは保証しません。

曲別の自動音量補正

AutoMasterはautoモード、目標program RMS -16dBFS、サンプルピーク上限-1dBFS、最大補正は増幅+6dB／減衰-12dBです。iTunesの曲識別に基づいて、曲ごとの測定と補正値を保存します。未知の曲はまず測定・学習し、保存された補正値を次回再生から曲頭へ適用します。

従来と大きく変わる点

曲ごとの音量差を、毎回の手動調整だけに頼らず、学習済みの補正値で軽減できます。学習前の曲へ完成済みの補正が最初からかかるわけではありません。ピーク抑制は別に働きます。これはRMSとサンプルピークを用いる機能であり、正式なLUFS／True Peak測定器ではありません。

保持した安定化処理

Webは音声サンプル基準の映像時刻、HLSの継続番号とプレイリスト更新、検証付きSFTP公開処理を保持しました。Twitはセッション終了時のエンコーダー解放修正を保持しました。HLSは2秒TS／20本、ツイキャスは29分セッション設定を継承しています。

導入時の不備と修正も記録する

ツイキャス初回切り替え時は起動スクリプトの文字コードで失敗し、旧版へ復旧しました。その後Windows PowerShell向けのUTF-8 BOMへ修正して正常に切り替えました。新ランチャーには二重起動防止を追加しました。旧版は別フォルダで保持し、復旧用ショートカットを用意しています。

31. 端末向け再生と操作画面

AndroidアプリとWebページ

画面ロック中も音楽を継続できるよう、Android用アプリにバックグラウンド再生機能を用意しました。利用者から画面ロック時の再生継続の確認がありました。Webのプレイヤー下にはAndroidだけにインストール案内を表示し、配布ページにインストール手順とGoogle Playプロテクトの警告画面の説明を追加しています。

iOS用Webアプリ

通常ブラウザでは「ホームへ戻る」を表示し、ホーム画面から起動したWebアプリでは隠す方針です。iOSブラウザ用のインストール案内も用意しました。製作者・アルバム等の外部リンクは別画面で開く設定へ変更し、ツイキャスへのリンクも追加しました。OSによる音声制御があるため、外部アプリ切り替え時の再生継続は実機で確認します。

日本語のAutoMaster Console

開始・停止、動作モード、学習状態、適用ゲインなどを日本語で表示するよう変更しました。状態ファイルはUTF-8で読み込みます。Webとツイキャスを個別に操作し、片方の操作ともう片方の状態を区別できる構成です。

iTunes COMタイマーを統合

既に稼働しているiTunesプレイリストタイマーをConsoleから設定・開始・停止できるようにしました。予約一覧と状態を表示し、時刻・プレイリスト・曜日を設定できます。曜日は「毎日・日曜日～土曜日」のプルダウンです。複数曜日は曜日ごとに行を追加できます。

タイマーの動作原則

設定時刻になっても現在曲の途中では切り替えず、曲の終了後に対象プレイリストへ切り替えます。保存した設定はタイマーが自動で読み直します。利用者はConsoleからの設定とタイマー動作を確認しました。例として毎日8:30と21:00の予約を継承していますが、これは利用者個別の設定です。

32. メール通知と今後の監視

独立したメール監視

配信エンジンとは別の監視処理で、プロセス停止・再開・再起動、内部エラー後のセッション再開、ローカルHLSの更新停止・再開を通知します。件名と本文は日本語です。GmailのSMTPでテスト送信がサーバーに受理されたことを確認しました。受理は受信箱への到着確認とは区別します。

認証と再送

送信認証にはGmailアプリパスワードを使用し、このWindowsユーザー専用の暗号化保存としています。パスワードを配信ログや公開資料に記載しません。送信失敗時は通知をローカルに保存して再送します。Windowsログイン時にメール監視を起動する設定を追加しました。

通知が意味すること

プロセス停止通知は手動停止も含み、停止だけでクラッシュとは断定しません。通常の29分再接続は異常と区別し、通知は選択式です。ローカルHLS更新は約10秒間隔で監視し、30秒超の停止を続けて検知した際に通知します。これは全端末で音が聞こえることの保証ではありません。

長時間監視の重点

セッション再接続後のメモリ推移、CPUとキュー滞留、gap/dropPCM/dropA/dropV、overrun、RTMP送信遅延、SFTP再試行と公開遅延を確認します。AutoMasterでは学習値・適用ゲイン・ピーク抑制量・補正解除エラーを確認し、曲ごとの不自然な音量差や歪みを実際に聴いて評価します。

端末確認と復旧

Android/iPhoneの再生開始、長時間再生、画面ロック、回線切り替え、外部リンク後の復帰を確認します。PC再起動後の起動状態、通知の再送、旧版への復旧も検証対象です。現在のメール監視は状態通知用であり、配信エンジンの自動復旧や外部Watchdogの稼働を保証するものではありません。PCや回線全体が止まれば、そのPCから即時送信はできません。

33. ChatGPTの大幅な介入と検証の限界

今回の制作・修正体制

今回は不備を減らすため、ChatGPT（Codex）が大幅に介入しています。提案や文章作成にとどまらず、実PC上のコード・設定・ログの確認、修正版の統合とビルド、起動不備の修正、旧版への復旧、公開ページとAndroidアプリの改善、操作画面の日本語化、タイマー統合、メール通知の実装・設定確認に関与しました。

利用者の実機確認を重視

ChatGPT側のログ・構文・生成物の確認に加え、利用者によるAndroid/iPhoneの表示・再生改善、画面ロック中の音楽継続、ツイキャスの正常再生、タイマー操作の確認を判断材料としています。実機での聴感確認と、機械的な正常ログの両方が必要です。

介入は無欠陥の保証ではない

ChatGPTが大幅に関与したことは、バグが存在しないことや、すべての環境で同じ結果になることを保証しません。起動文字コードの失敗も隠さず記録し、確認済み・未確認・今後の監視項目を明確に分けています。現時点の判断は「観測範囲では正常稼働、RC1は長期試験を継続」です。

参照した根拠

既存ガイド48ページ、10月10日の設置記録、Web公開約3分の監視結果、Web/TwitのHEALTH・セッションログ、AutoMasterの学習・適用状態、メール送信ログ、利用者からの実機確認を参照しました。基となったエンジンコードの7日間連続配信実績はJUN氏の運用確認によるものです。今回の追補は実際に導入した構成の記録であり、受領RC1 ZIPそのものの機能一覧ではありません。

今後の記録方法

「稼働開始からの期間」「計画停止を含む運用期間」「中断のない連続運転時間」を別々に記録します。安定記録を更新する際は、日時・構成・再接続の種類・欠落やエラーの有無・実機再生結果を残します。AIの推測だけで安定実績を延長しません。