令和２年度 メディア芸術連携基盤等整備推進事業
連携基盤整備推進事業 連携基盤強化事業

リストＤＢからメディア芸術データベース（ベータ版）への登録、及び、全録サーバのテキスト抽出の検証　実施報告書

一般社団法人日本アニメーター・演出協会

令和３年２月

目次

目次

第 1 章　事業概要

1.1　目的（目的・背景）

アニメ制作従事者に関する記録であるクレジットは、アニメ鑑賞においてはより深い理解をもたらし、アニメ研究では一次資料として重用される。また、原画等の中間制作物や関連資料を保存し活用するアーカイブ活動において、クレジットはそれらの価値を理解し、利活用方法を検討するための基礎情報である。さらに、今後、様々な利活用が期待される中間制作物や関連資料に関する権利などの処理を円滑に進めるためにも、重要な情報となる。しかし、アニメ産業が隆盛な一方で、この業界ではどのような職種の人材がどの程度の規模存在し、どのような作品を作っているのか、といった最も基本的な状況すら把握が難しい。そもそも制作されている作品タイトルの全体像すら把握が困難である状況は改善されていない。

令和元年度メディア芸術連携促進事業にて、特定非営利活動法人アニメ特撮アーカイブ機構（以下 ATAC）が実施した「アニメスタッフデータベースの持続的な構築・整備に向けての調査・検証」では、アニメのタイトル、スタッフに関しての網羅的なデータベースを強化し発展させていく方策の一環として、関東広域圏地上波デジタルテレビ局で放送されるアニメを網羅的に録画するシステムを構築したことを背景として、＜テーマ 1＞①全映像ファイルからスタッフクレジット画面の抽出、②効率的なスタッフクレジット画面とそれ以外の分類、③スタッフクレジット画面のテキスト化、④前述①～③の自動化の試行、＜テーマ 2＞原口正宏氏が率いるリスト制作委員会にて蓄積されている「リスト DB」から「メディア芸術データベース（β版）」（以下「MADB（β版）」）へのデータ反映の仕組みを構築した。

1.2　実施内容（概要）

令和元年度に構築済みのアニメ全録サーバに蓄積された映像ファイルと EPG ファイルを元に①EPG ファイルの統合機能の作成、②全映像ファイルからスタッフクレジット画面の抽出機能の作成、③抽出された画像の OCR 機能の作成、④データベースに投入するための各種検討、⑤「リスト DB」の分析、⑥「リスト DB」から MADB（β版）へのデータ移行についての検討を行った。

「クレジット」採集の有力な方式として、番組情報として配信される①EPG データ、放送された番組の CSV（Comma Separated Value）ファイルへの統合処理と、OP や ED に表示される文字情報の OCR（Optical Character Recognition/Reader）処理の簡便化を行った。具体的には、地上波、BS、CS 等で放送されるアニメを東京都千代田区、同杉並区に設置したサーバですべて録画した上で（昨年度実施済み）、①は番組に付随して配信される EPG データから放送局、放送日時、番組名、番組情報を CSV ファイルに統合化して管理可能にした。②は録画映像から OP、ED を 1 秒ごとに画像として切り出してファイルに一時的に保存し、この画像を OCR 処理してテキストとして取得できるようにした。

日本国内で公開されたアニメ作品の、とりわけその歴史に関する研究の第一人者であるリスト制

作委員会の原口正宏氏が長年にわたって蓄積している「リスト DB」は、DOS 時代のカード型データベース上にて管理されており、当時のコンピュータの能力的な制約や、「リスト DB」の運用思想などの理由から、現状のリレーショナル DB（以下 RDB）や MADB（β版）へのデータ移行については高い障壁がある。また、「リスト DB」は私的な目的で長年蓄積された知的財産でもあるため、本年中の取扱いでは細心の注意を払い、将来の MADB を介しての公開には多少の制約が発生することが確認できた。

1.3　実施体制

本事業の実施体制は以下のとおりとなる。

主催：文化庁
　　一般社団法人 日本アニメーター・演出協会（JAniCA）

コーディネーター：高橋 望（ATAC）
オブザーバー：大坪英之（JAniCA）
担当者：三好 寛（ATAC）
　　辻 壮一（ATAC）
　　須山大介（合同会社 Prod.）
　　原口正宏（リスト制作委員会）
　　磯部正義（リスト制作委員会）
　　谷口隆一（ライター）

1.4　実施スケジュール

本事業の実施スケジュールは以下のとおりである。

令和 2 年 4 月 1 日　記録作成のための方針会議
令和 2 年 5 月 19 日　内部会議（第 1 回）
令和 2 年 6 月 19 日　内部会議（第 2 回）
令和 2 年 7 月 10 日　OCR 会議①
令和 2 年 7 月 14 日　リスト制作委員会・磯部氏ヒアリング
令和 2 年 7 月 21 日　内部会議（第 3 回）
令和 2 年 8 月 20 日　リスト制作委員会・原口氏ヒアリング①
令和 2 年 8 月 28 日　内部会議（第 4 回）
令和 2 年 9 月 29 日　内部会議（第 5 回）
令和 2 年 10 月 5 日　OCR 会議②
令和 2 年 10 月 15 日 リスト制作委員会・原口氏ヒアリング②

令和 2 年 10 月 22 日 内部会議（第 6 回）
令和 2 年 11 月 12 日 リスト制作委員会・原口氏ヒアリング③
令和 2 年 11 月 19 日 内部会議（第 7 回）
令和 2 年 11 月 20 日 中間報告会
令和 2 年 12 月 3 日　OCR 会議③
令和 2 年 12 月 17 日 リスト制作委員会・データ検証会議
令和 2 年 12 月 25 日 内部会議（第 8 回）
令和 3 年 1 月 20 日　内部会議（第 9 回）
令和 3 年 2 月 9 日　　報告書執筆会議
令和 3 年 2 月 10 日　内部会議（第 10 回）
令和 3 年 2 月 19 日　最終報告会

第 2 章　成果・課題

2.1　実施結果

昨年の事業においてアニメ作品の全録を行うことでクレジット部分の表示内容を映像として確認する体制は整ったが、最終的なメタデータ化に至る道のりとして「クレジット画面の抽出（動画から静止画）」「OCR 処理（静止画からシンプルテキスト）」「メタデータ化（シンプルテキストから意味付けされた語句の整理）」を想定し、本年度事業においては「クレジット画面の抽出（動画から静止画）」「OCR 処理（静止画からシンプルテキスト）」を試行した。また、全録サーバに映像と併せて記録された EPG ファイルも利用価値があることが判明したため「EPG 統合」機能も作成した。

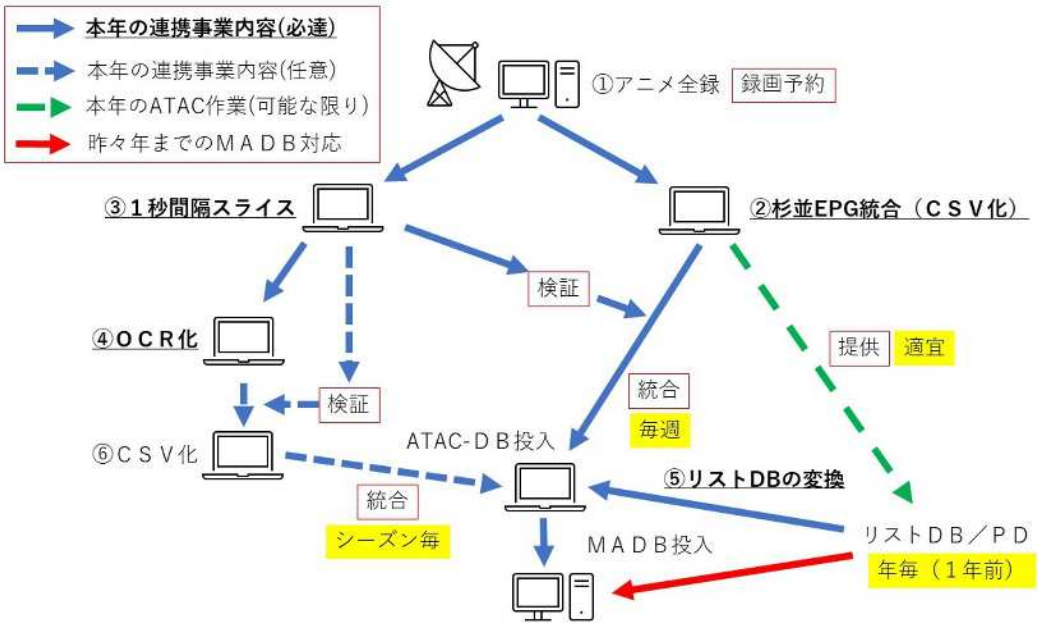


図 2-1　本事業のデータの流れ

リスト DB については、過去の事業においてファイル構造やフォルダ構成についての事前調査が済んでいたこともあり、詳細なデータ構造や、格納されているデータの意味、運用上の留意点、dBASE に起因した制約などについてヒアリングを重ねて、MADB（β版）へのマッピングを行い、接続するデータ濃度（範囲、粒度）について検討を行った。

2.1.1　1 秒キャプチャ

アニメ作品のクレジット画面を OCR にてテキスト化する前処理として、映像全部（最初から最後まで）について、抽出間隔については 2 秒ないし 1 秒間隔、対象映像ファイル形式としては TS ファイルないし MP4 ファイル、抽出画像の保存形式としては JPEG ないし PNG 形式の選択式とし、画面インターフェースを持つモード（UI モード）と画面インターフェースを持たない実行モード（UI レスモード）を選択できるプログラムを開発した。抽出される画像ファイルは「映像ファイルのフォルダを作成し、連番のみのファイル名」となる。

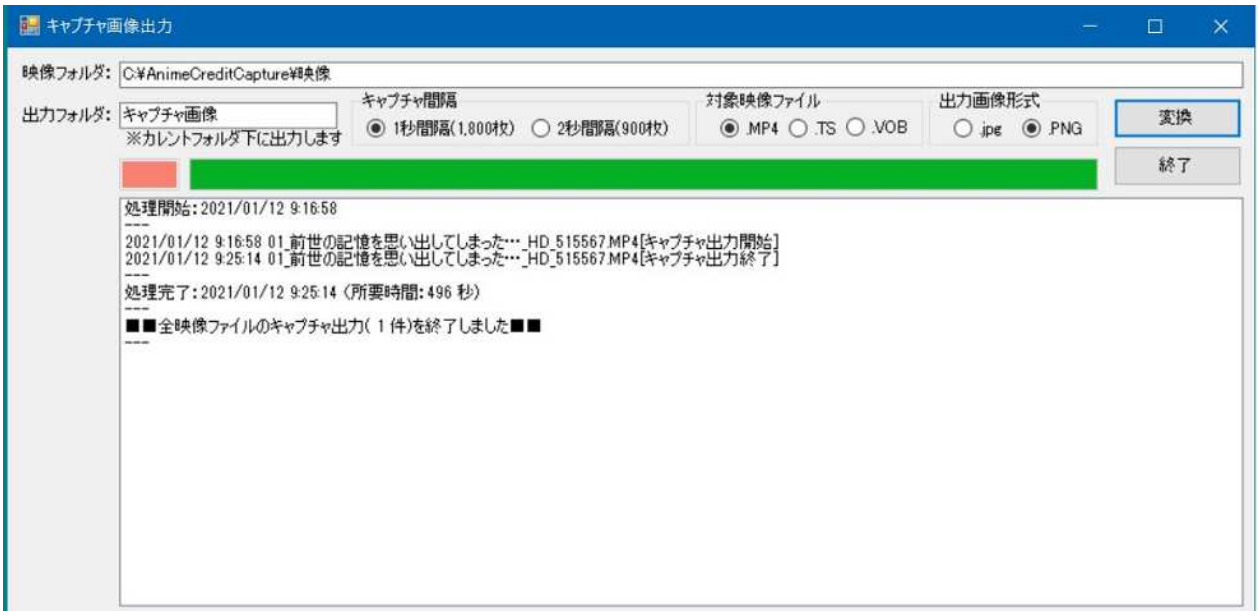


図 2-2　1 秒キャプチャの動作画面

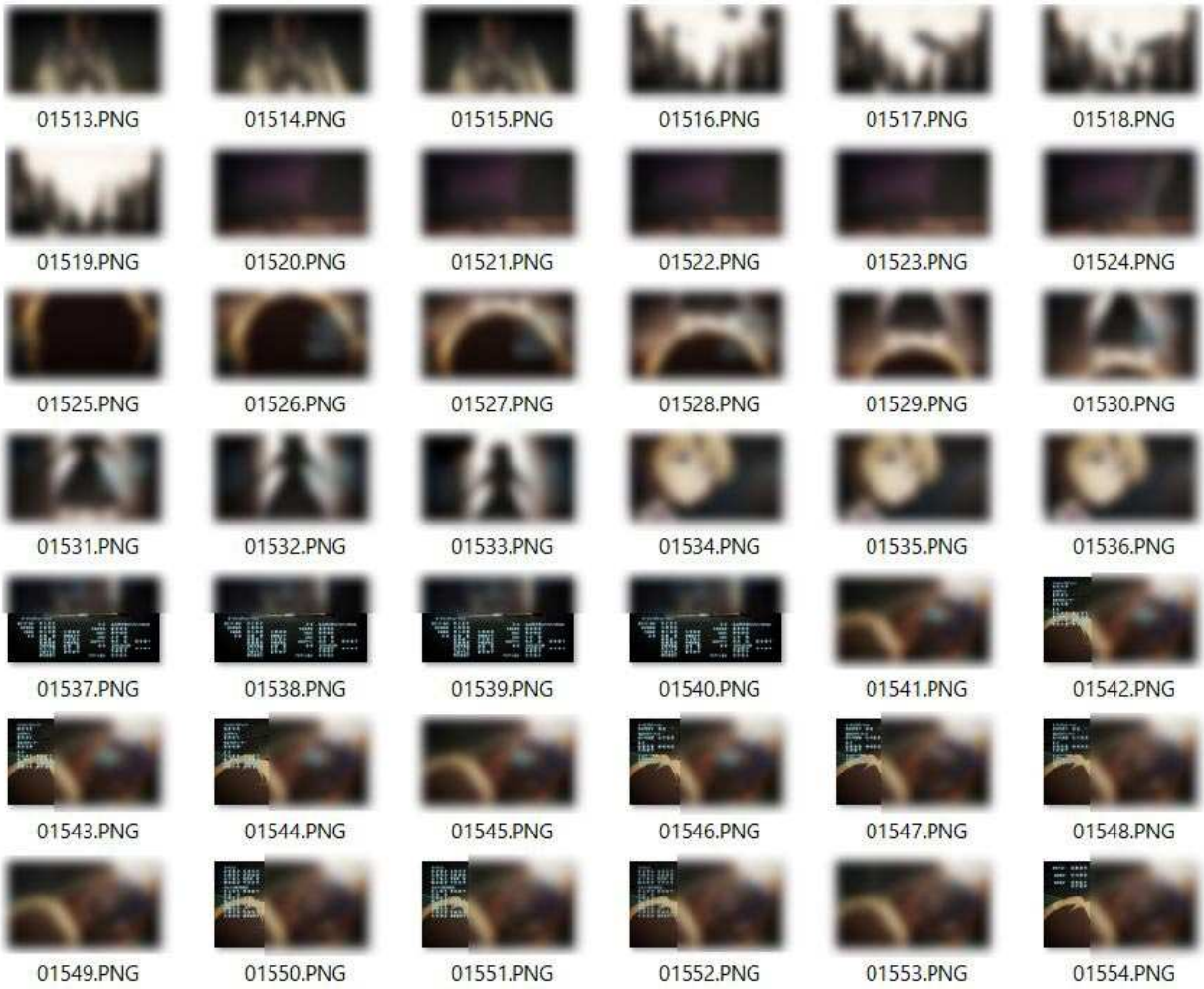


図 2-3　1 秒キャプチャの出力例

原口正宏氏に対するヒアリングにおいて、「全部の映像をスライスして切り出し、順番に保存しておくと、クレジットを採集していく際に前の方から順に行えて便利」「絶対時刻があればガイドとして分かりやすく、クレジットが出る画面を時間からピンポイントで探し、映像を確認する使い方もできるため、時刻表示があった方が良いが、付けられなくても 00 秒からのカウントアップで判断できる」という評価を頂いた。

2.1.2　OCR

前項で抽出した画面のうち「OCR に向いた画面（複数）」であることを前提として、まず OCR サービスの比較検討を行い、GoogleCloudVision の OCR サービスを選択した。次に GoogleCloudVision の OCR サービスを利用して、テキスト化を行う仕組みを構築した。

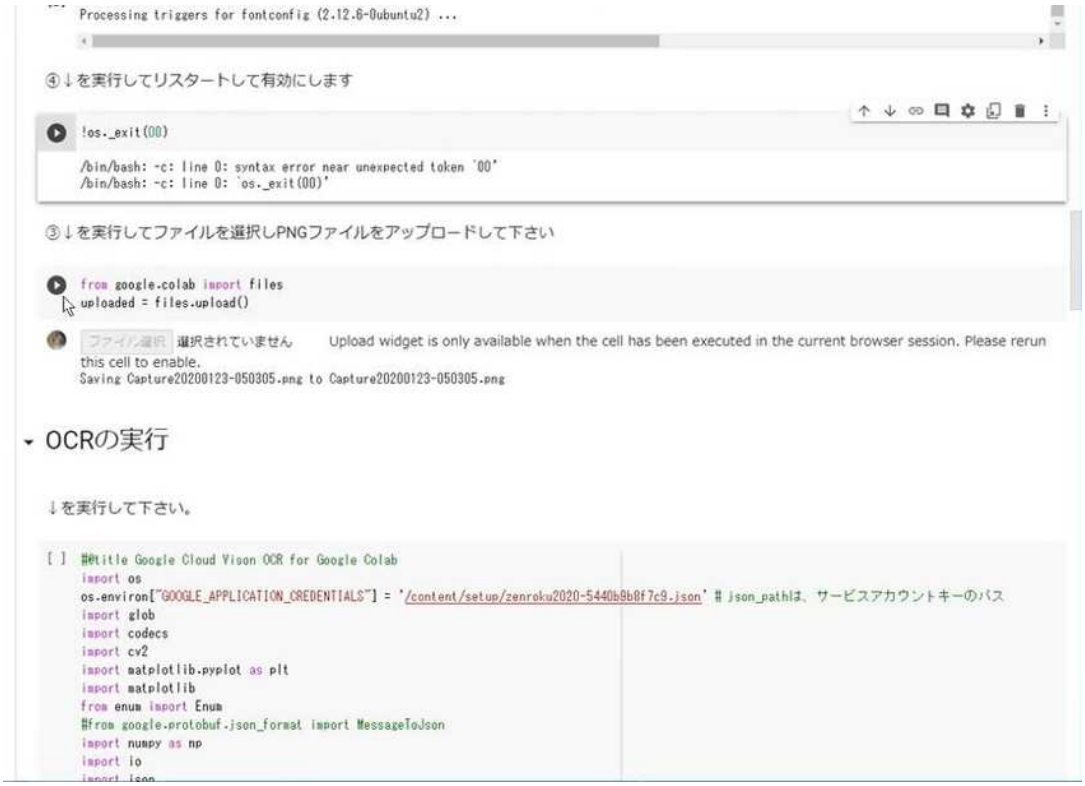


図 2-4　OCR 処理の動作画面（1）

1 番組（1 話）のクレジット画面を適切に用意した場合、数分程度でテキストを取得することができるようになった。

図 2-5　OCR 処理の動作画面（2）

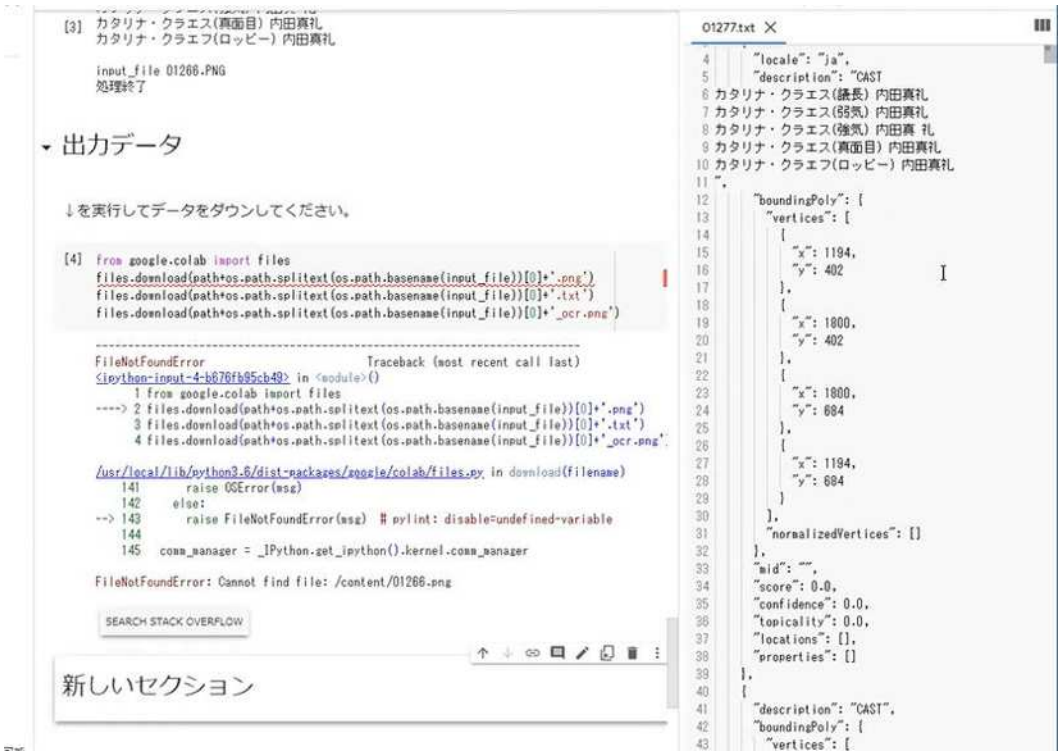


図 2-6　OCR 処理の動作画面（3）

2.1.3　EPG 統合

全録サーバには番組録画（映像ファイル化）と同時に、番組に紐［ひも］付く EPG ファイルも記録される。映像ファイルや OCR 処理を行って取得したテキストを管理する台帳として録画一覧が必要であることに加えて、映像そのものには含まれない放送情報などのテキスト情報としてこの EPG ファイルについて有用性があるため、このファイル群を統合する機能を作成した。

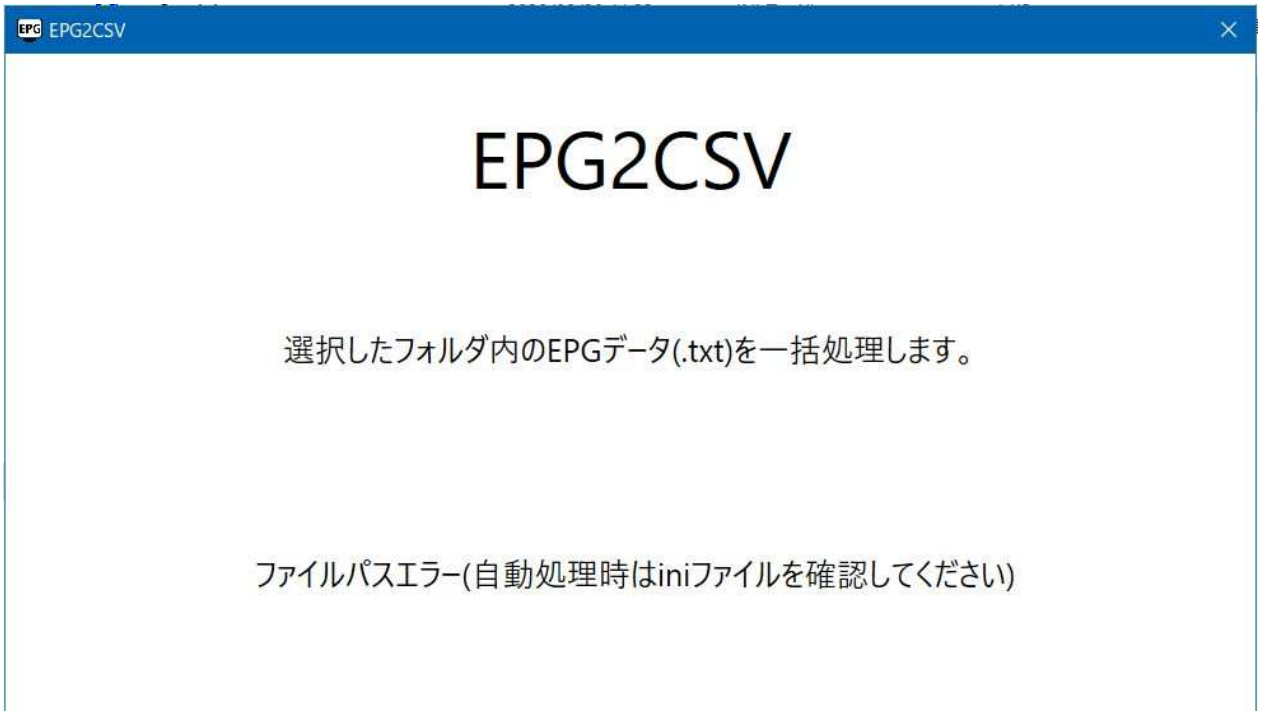


図 2-7　EPG 統合の動作画面

ファイルの統合には約 1,000 ファイルを数分程度で統合し、番組名については定型化した振り分けルールにて枠タイトル、作品タイトル、話数表記、各話タイトル項目への簡易なメタデータ処理を行い、手作業での負荷を激減できた。

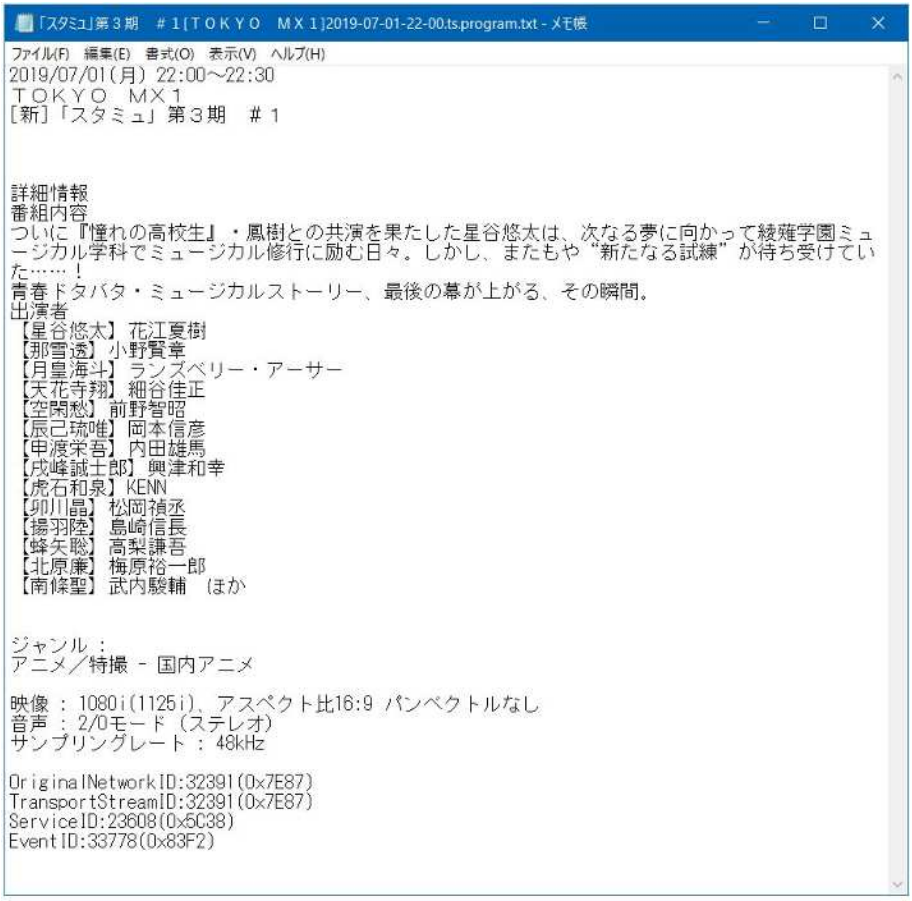


図 2-8　EPG ファイルのサンプル

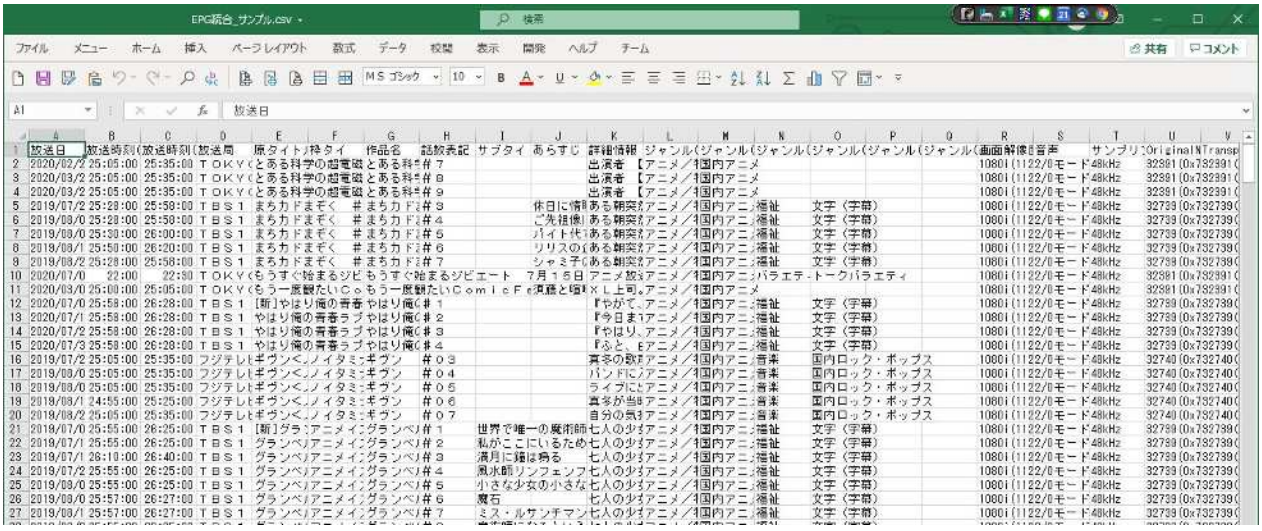


図 2-9　EPG 統合の処理結果

2.1.4　リスト DB 変換

まず、リスト制作委員会の原口氏、磯部氏に対して、「リスト DB」の設計思想、運用上の留意点についてヒアリングを行った。次に、ヒアリングに基づき各フォルダ構成やテーブルについての解析を行い、疑問点などを整理して、改めて格納された値などについて詳細なヒアリングを行った。

解析の結果、「リスト DB の設計思想」（①リスト DB が画面表示を記録するシステムであり、人名の完全な正規化がなされていない）、「カード型データベースに起因する問題」（②格納順番に意味を持つ、③カラムの格納量によるレコード分割）、「共通スタッフと各話スタッフの分離」（④各話の総スタッフを列挙する際の特徴）、「表示以外の値を持つ」（⑤クレジット表記上の省略表記の補完、⑥事後の調査研究の反映）、「営業上の理由による提供不可領域」について確認できた。

解析を踏まえて、リスト DB と MADB（β版）を仲介する中間 DB（ATAC-DB［仮称］）を作成できた。

2.2　結果からの推測、課題

2.2.1　OCR

EPG に比べて誤字が多い。誤字が発生する条件は、解像度が足りず画面上の文字が潰れている場合、文字が小さすぎる場合、手書きのようなフォントで書かれている場合、文字の背景に模様がある場合、文字は止まっていても背景画が動いている場合などがある。これらの外乱要素がなければ、OCR は 90%の確率で認識できる。ただし、異体字が多くある「渡辺・斉藤」のような文字の場合は、機械でも人でもどの文字が正しいかを確定できない場合がある。

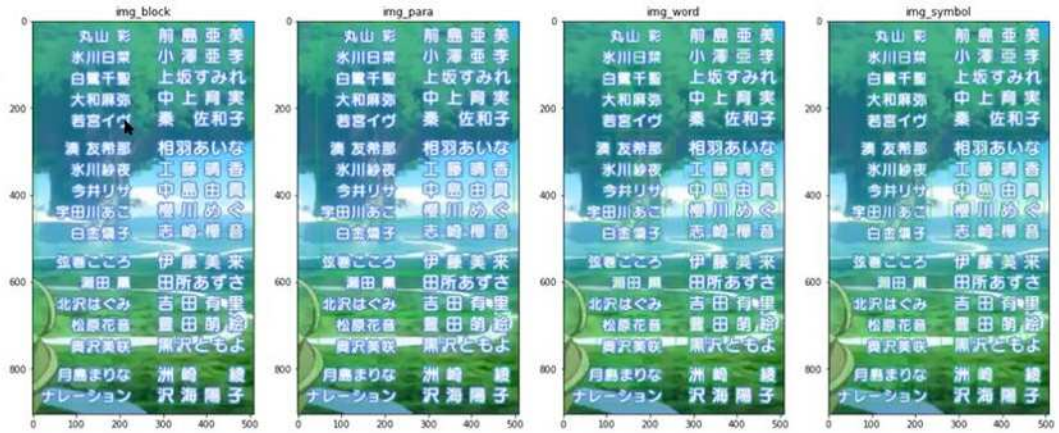


図 2-10　クレジット画面の表記例

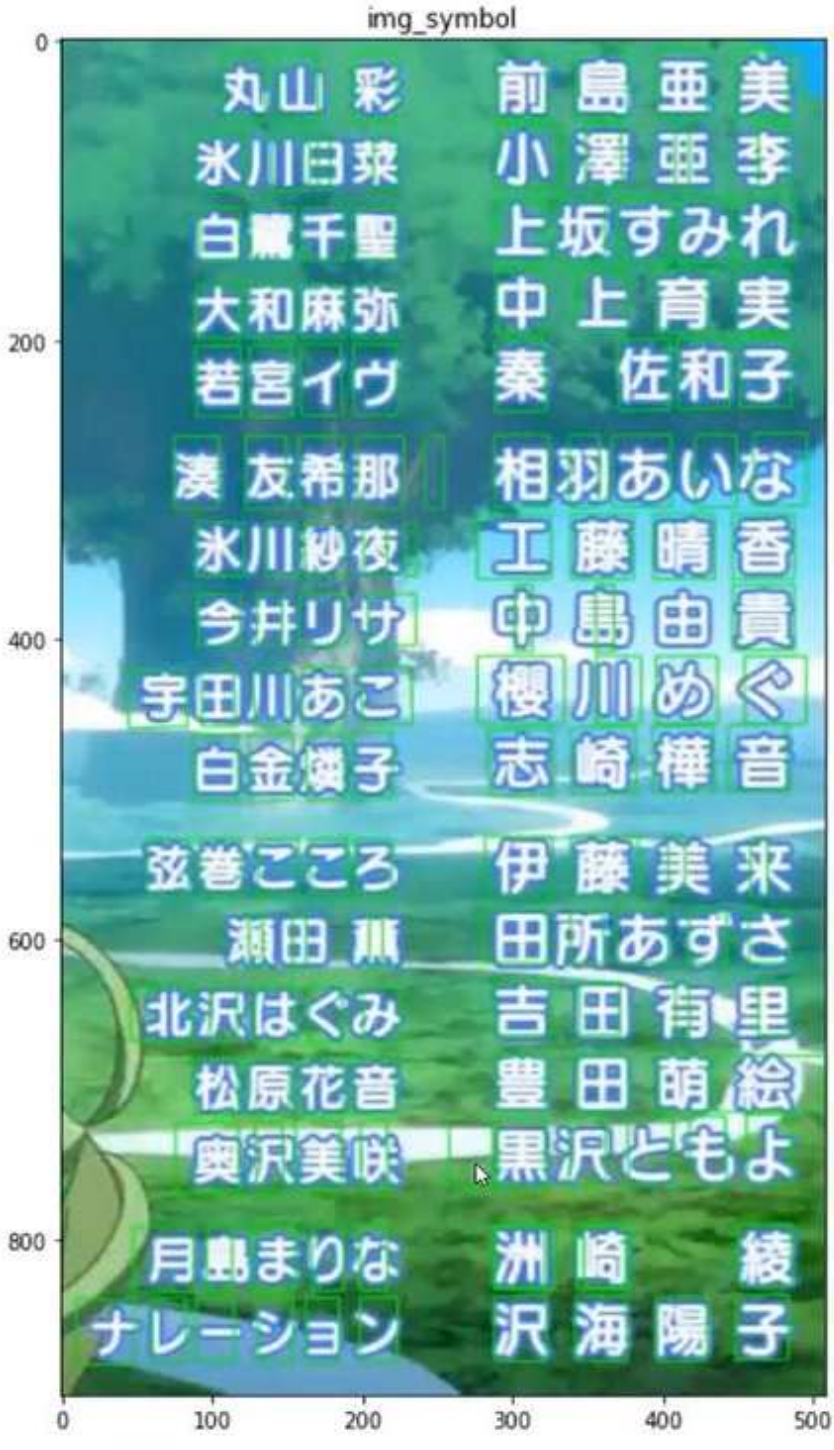


図 2-11　クレジット画面での文字認識領域

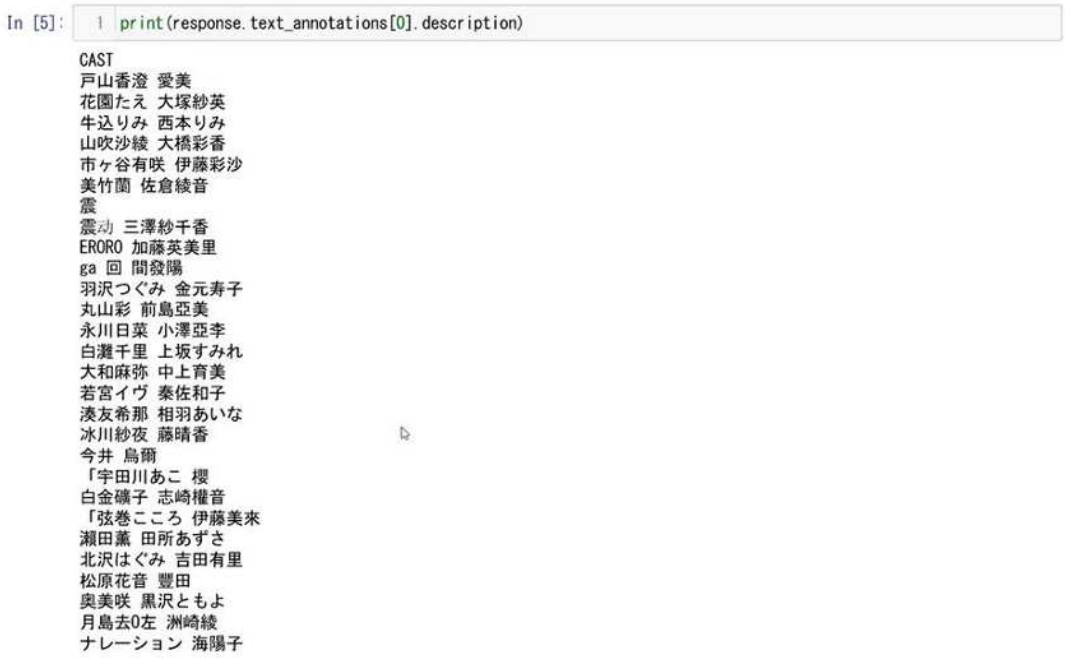


図 2-12　テキスト化されたクレジット画面

いかに認識率を向上させるかが課題である。画面全体を読み込ませるのではなく、文字が出ている領域を限定すると誤字が減ることがある。背景の色を変えるなど、前処理によっても誤字が減る。AI の性能が日進月歩のため、前処理を突き詰めるよりも AI 技術の発展を待った方が良いかもしれないと考えている。

誤字を低減する方法としては、修正用のデータをあらかじめ用意するケースも考えられる。手動で選別するために使う、あからさまな間違いは自動修正する―といった方法で誤字を極力減らすアプローチも試行した。

汎用 OCR は文字組みの認識精度が低く、スペース（空白）が存在すると、右と左で別の単語・語句であると解釈してつながらなくなってしまう。対策として、文字の集団を認識するための機械学習 （クラスタリング、まとめて処理する） を行った。この処理を行うと、汎用 OCR ではできなかった部分、苦手だった部分が正しく認識されるケースが増えた。

汎用 OCR は、1 画面単位での課金が発生し変換コストがかさんでしまう。映像中に表出した文字をどうテキスト化するかも課題である。

人名は、肩書を分離し意味付けしなければ、有用なデータとして使えない。画面組みの位置情報から類推し、今までの DB からこの人はこれだろうという推定を行う必要がある。また、表記揺らぎや、スペース（空白）の有無など。クレジット画面を制作する担当者がどの文字を基本としているのかは明確ではない。

現在、OCR 処理はクラウドサービスを用いているが、クラウドサービスを使わない方が、いろいろと調整して特徴を把握し、認識性能を上げられる。他方で、クラウドサービスは膨大なデータで学習をしているという強い面がある。それを組み合わせるのも良いが、すべてのデータをクラウドに上げていてはコストが合わない。クレジット文字有無の場所を問わずに文字認識で、1 番組全部の 1,800 枚をアップロードすると 290 円となる。1 秒スライスだけに限定した場合は 145 円。これが 1 シリーズで 1 シーズン、1 年でトータル幾らになるのか。実際は、これだけの作業工数ではないため、諸費用を含めるともっと高くなる。ビジネスモデルとして考えた場合、クラウドサービスを基準にする場合は資金面での永続性に問題がある。

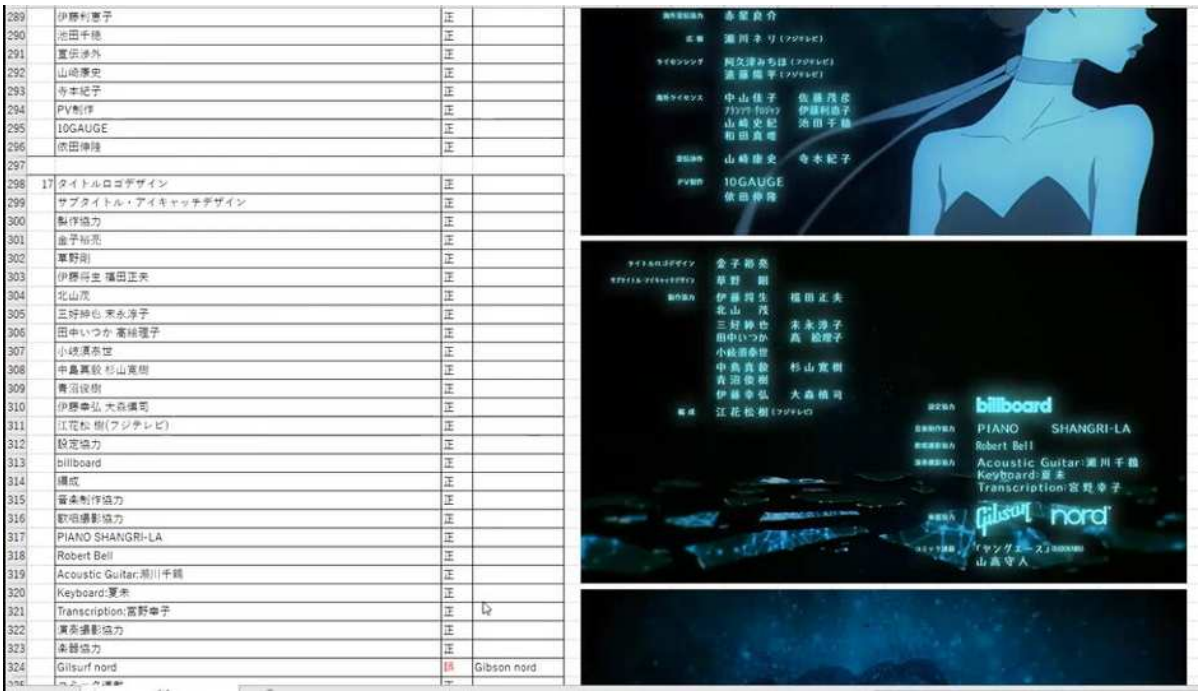
クレジットのテキスト化と簡単に言っても、実は簡単ではないことが、改めて認識された。前処理、事業化、OCR の学習など次年度以降にも取り組むことによって、より精度が高く処理時間も早い OCR からのテキスト化によるクレジット採集が行えるようになるだろう。

原口氏から「間違っていてもいいから、ある一定のルールで全部拾ってくれる。そこに 1 秒スキャンのような確認できる画像データがあれば、照らし合わせは人間がやれば良い。1 から入力する手間が、2 割でも 3 割でも減って、誤差修正するだけになれば格段に作業量は減る。そのための一歩だと考えているので、間違いをするにもその間違え方を統一してほしい。」という御意見を頂いた。

また、「リスト DB」は人名や役職名についての学習に使える。OCR は作品クレジットに特化しておらず、一般の日本語に準じている（汎用の）ため、これに「リスト DB」から抽出したアニメに特有の用語や人名を反映させて、辞書として登録すると認識精度が上がるかもしれないと考えている。

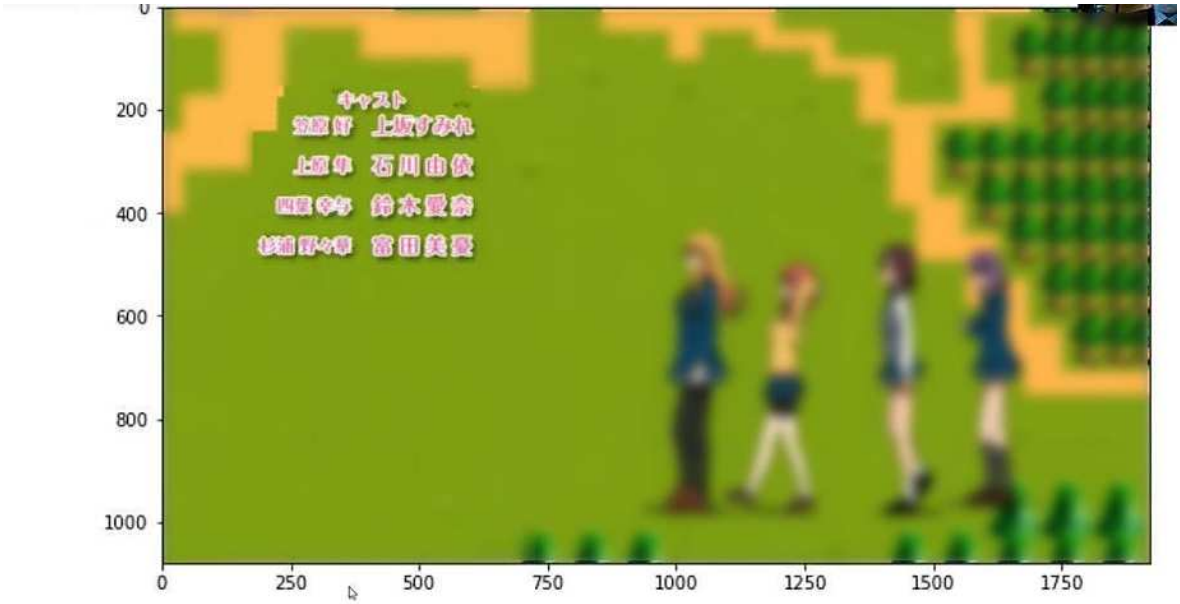
他方で、「人間の頭にある知識は便利だが危険でもある。経験がなまじあるため、こう書いてあるものだと思い込みで読んでしまうからだ。違和感を抱いても、そこで待てよとならないと、本当に字を観察していることにならない。機械は感情を抜きにして厳密に字を見るから良さがある。これに学習させると人間に近付いてしまわないか。曖昧であったり、矛盾を持ったりすることが、AI が人間っぽくなる道だが、この作業、テロップを拾うことに関しては、冷酷なままで良い。この場に何があるかを正確に拾う方向は捨ててはいけない。その上で、取り込んだものを意味として理解する方が本当は良い。」と原口氏からの指摘もあり、さじ加減が非常に難しいことを感じた。

原口氏よりあった示唆と要望として、「頭の中で、こういうプログラムならこのように検出するだろうと推測する拾い方に近いのが、『キャロル＆チューズデイ』のギブソンなどが表示されているテロップ。大きく分けて左側と右側 にテロップのブロックが分かれている。それぞれが左右に 2 項目ずつ並んでいる。制作協力の中に「伊藤」と「福田」がいる。この場合、人間の頭なら、左から拾い、それから「タイトルロゴ」「サブタイトル/アイキャッチデザイン」と来て「制作協力」を 「伊藤」「福田」「北山」……と拾う。」



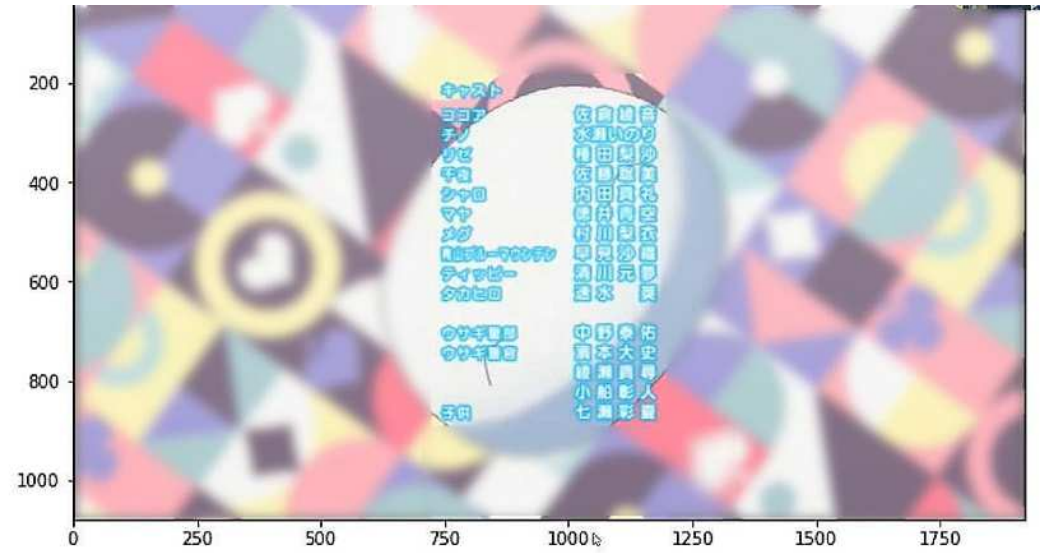
289		伊藤利恵子	正	
290		池田千穂	正	
291		夏伝涉外	正	
292		山崎康史	正	
293		寺本紀子	正	
294		PV制作	正	
295		10GAUGE	正	
296		依田伸隆	正	
297				
298	17	タイトルロゴデザイン	正	
299		サブタイトル・アイキャッチデザイン	正	
300		制作協力	正	
301		金子裕亮	正	
302		草野剛	正	
303		伊藤将生 福田正夫	正	
304		北山茂	正	
305		三好伸也 末永浄子	正	
306		田中いつか 高綾理子	正	
307		小峻須奈世	正	
308		中島真哉 杉山寛樹	正	
309		青沼俊樹	正	
310		伊藤幸弘 大森慎司	正	
311		江花松樹(フジテレビ)	正	
312		設定協力	正	
313		billboard	正	
314		構成	正	
315		音楽制作協力	正	
316		歌唱撮影協力	正	
317		PIANO SHANGRI-LA	正	
318		Robert Bell	正	
319		Acoustic Guitar:瀬川千鶴	正	
320		Keyboard:夏未	正	
321		Transcription:宮野幸子	正	
322		演奏撮影協力	正	
323		楽器協力	正	
324		Gilsurf nord	誤	Gibson nord

図 2-13　クレジット画面とテキストの正誤比較



キャスト
原好 上坂すみれ
原隼石川由依
葉鞘 鈴木愛奈
杉浦野簿富田美憂

図 2-14　クレジット画面の処理例（1）



キャスト
ココア
佐倉綾音
チノ
水瀬いのり

図 2-15　クレジット画面の処理例（2）

ユナ
カイ
キャスト
希

図 2-16　クレジット画面の処理例（3）

OCR を行うと、機械的に「伊藤」と「福田」を 1 列と考えるパターンと、「伊藤」「北山」「三好」と拾って「福田」と「末永」が後回しになるパターンがある。 逆の発想としては、「伊藤」と「福田」の間にスペースがある。大きく分けると左側と右側のブロックの間にも、大きなスペースがある。遠くから見たら大きくブロックを分けるスペースが確認できるときに、先にそこに境界線を引いてしまう。あるいは、横の行で 1 行目 2 行目 3 行目とスライスを引き、碁盤目状の線を引いてから、線で引いた一つ一つが文字列であると認識した上でナンバリングができれば良いかもしれない。「タイトルロゴデザイン」が 1、「金子裕亮」が 2、「サブタイトル／アイキャッチ」が 3、「草野剛」が 4 とナンバリングする。その前段階として、役名だろうが人名だろうが、関係なくそこが一区切りであるだろうと見極めるため、空いているスペースを頼りに線を引いて、幾何学的にさいの目にすることができれば有益である。

図 2-17　段組されたクレジット画面

並び順は、統一的に学習はしづらいため、並び順はある段階までは人が補助をして、左に左にと拾ってほしい場合、左のブロックが終わってから右のブロックを拾ってほしいというパターンがある。そのような識別方向の大きなパターンを分類する方法も考えられる。

OCR は、現時点では「夢の機能」で、無調整で自動的にテキスト化して全部が正しくテキスト化されることにはならない。また、テキスト化に際しても、人間なら役職と人名で意味を分けるが、テキストにしてしまうと意味を判別することなく等価に扱われてしまう。これは、画面の文字（図形情報）が文字コード化されただけで、本質的な意味付けがされておらず、この分類作業の自動化は今後

必要になる。原口氏が現在行っている作業は極みの先にあって到達するには多くの障壁があるが、いかに省力化・高精度化できるかが鍵だと考えている。

2.2.2　リスト DB 変換

MADB（β版）への連携に際しては、幾つかの障壁がある。まず、カード型 DB とリレーショナル型 DB の DB マネージメント機能に起因した点であり、特に、リスト DB が画面表示を記録するシステムとして設計・運用されているため、人名が大きく 8 つの役職ごとに並列して記載されていて完全な正規化がなされていない。例として、動画→原画→作画監督→監督とキャリアを移った場合、「動画→原画→作画監督」については同一テーブル内で共有の人物 ID を使用するが、「監督」は別テーブルにも情報が格納されているため、自然人としては同一でも複数テーブルに人物情報が同時に存在してしまう。同様に制作→演出→監督や、仕上げ→動画等のケースもある。人名が完全一致した場合に同一 ID に名寄せする方法も検討したが、明らかな同姓同名の人物が存在することと、明らかではない同姓同名の人物の可能性も考えたために機械的な統合が不可能であると判断した。

リスト DB は、クレジット表示を記録するだけでなく、原口氏の取材や調査の結果をも記録していることが判明したが、そのために採録根拠について外部からの検証性が困難である。そのため、検証性と信用性を担保しつつ MADB（β版）へ連携するには、画面上にクレジット表示された情報のみに限定してデータを抽出する必要がある。また、クレジット表記上で省略された役職についても補完登録されている（例えば、原画が 5 名列記される場合に、最初の人にだけ原画という役職名が記載され、残りの 4 名には役職名が表記されないケース）。そのため、役職名＋人物情報は ID 化された（正規化された）状態で連携できず、文字列として連携する必要がある。

表 2-1　MADB（β版）への連携例（1）

No	OP/ED	ページ	種類	職名	役名・会社名	氏名	備考	メモ
1	OP	1		原作		山口　悟	（一迅社文庫アイリス／一迅社刊）	
2	OP	1		キャラクター原案		ひだかなみ		
3	OP	2		企画		遊佐　和彦		
4	OP	2		企画		丸山　博雄		
5	OP	2		企画		金子　逸人		
6	OP	2		企画		染谷　誓一		
7	OP	2		企画		今井　克幸		
8	OP	2		企画		三嶋　章夫		
9	OP	2		企画		新井　重人		
10	OP	2		企画		武智　恒雄		
11	OP	2		企画		高橋　和也		
12	OP	3		シリーズ構成		清水　恵		
13	OP	3		キャラクターデザイン		大島　美和		
14	OP	3		プロップデザイン		澤入　祐樹		
15	OP	4		美術監督		込山　明日香		
16	OP	4		色彩設定		重富　英里		
17	OP	4		ＧＤディレクター		小笠原　努		
18	OP	4		撮影監督		衛藤英毅		
19	OP	4		編集		瀧川　三智	（ＲＥＡＬ－Ｔ）	
20	OP	4		音響監督		亀山　俊樹	（ｇｒｏｏｏｖｅ）	
21	OP	5		音楽		田淵　夏海		
22	OP	5		音楽		中村　巴奈重		
23	OP	5		音楽		高木　達彦		
24	OP	5		音楽		櫻井　美希		
25	OP	5		音楽		兼松　衆		
26	OP	5		音楽プロデューサー		水田　大介		
27	OP	5		音楽制作		日音		
28	OP	5		音楽協力		ミリカ・ミュージック		
29	OP	5		音楽協力		日音		
30	OP	6	オープニングテーマ			乙女ルートはひとつじゃない！		
31	OP	6				angela		
32	OP	6		作詞		atsuko		
33	OP	6		作曲		atsuko/KATSU		
34	OP	6		編曲		KATSU		
35	OP	7		原作協力		ノベル編集部		
36	OP	7		原作協力		根上　つかさ		
37	OP	7		エグゼクティブプロデューサー		竹内　文恵		
38	OP	7		エグゼクティブプロデューサー		中西　豪		
39	OP	8		プロデューサー		西　啓		
40	OP	8		プロデューサー		青井　宏之		
41	OP	8		プロデューサー		神宮司　剛史		
42	OP	8		プロデューサー		諏訪　豊		
43	OP	8		アニメーション制作プロデューサー		金子　逸人		
44	OP	9		アニメーション制作		ＳＩＬＶＥＲ　ＬＩＮＫ．		
45	OP	10		監督		井上　圭介		
46	OP	11		製作		はめふら製作委員会		
47	OP	11		©		山口悟・一迅社／はめふら製作委員会		

原作 山口悟（一迅社文庫アイリス／一迅社刊），キャラクター原案 ひだかなみ，企画 遊佐和彦，丸山博雄，金子逸人，染谷誓一，今井克幸，三嶋章夫，新井重人，武智恒雄，高橋和也，シリーズ構成 清水恵，キャラクターデザイン 大島美和，プロップデザイン 澤入祐樹，美術監督 込山明日香，色彩設定 重富英里，３Ｄディレクター 小笠原努，撮影監督 衛藤英毅，編集 瀧川三智（ＲＥＡＬ－Ｔ），音響監督 亀山俊樹（ｇｒｏｏｏｖｅ），音楽 田淵夏海，中村巴奈重，高木達彦，櫻井美希，兼松衆，音楽プロデューサー 水田大介，音楽制作 日音，音楽協力 ミリカ・ミュージック，日音，オープニングテーマ 乙女ルートはひとつじゃない！，angela，作詞 atsuko，作曲 atsuko/KATSU，編曲 KATSU，原作協力 ノベル編集部，原作協力 根上つかさ，エグゼクティブプロデューサー 竹内文恵，中西豪，プロデューサー 西啓，青井宏之，神宮司剛史，諏訪豊，アニメーション制作プロデューサー 金子逸人，アニメーション制作 ＳＩＬＶＥＲＬＩＮＫ．，監督 井上圭介，製作 はめふら製作委員会，© 山口悟・一迅社／はめふら製作委員会

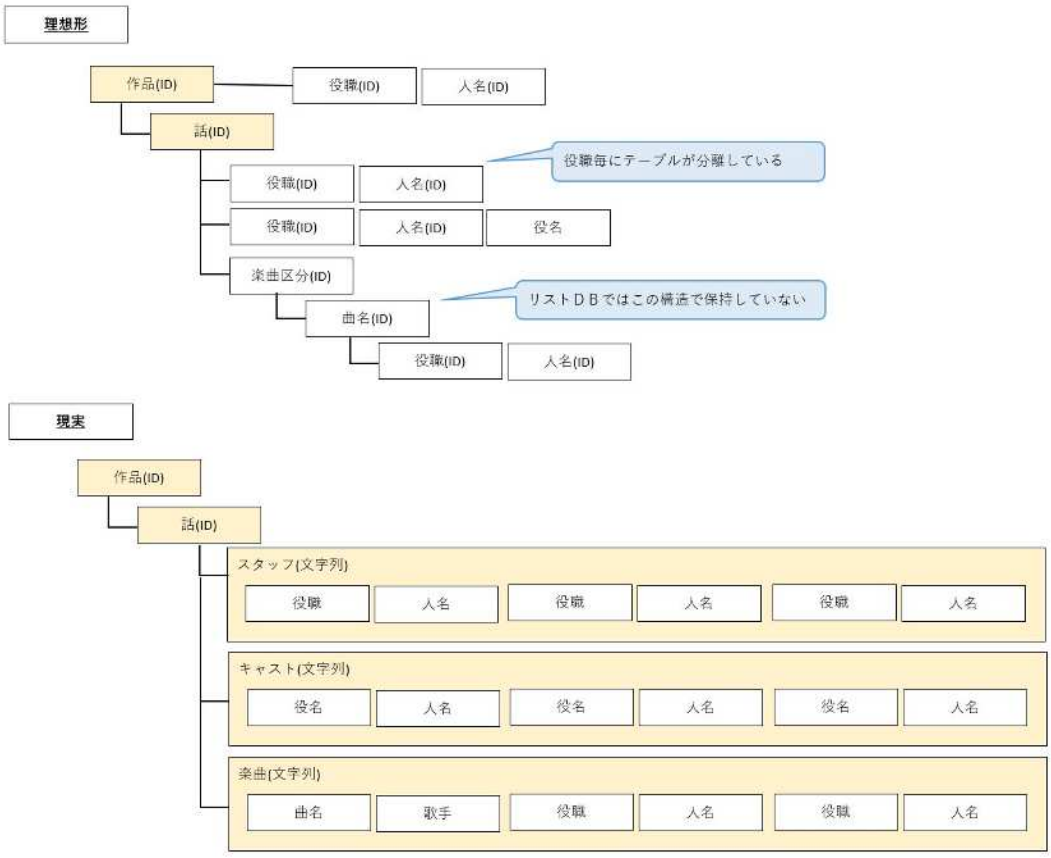


図 2-18　MADB（β版）への連携例（2）

リスト DB は、原口氏の民業との兼ね合いで詳細な情報を提供しない作品が幾つか存在する（例：『ゲゲゲの鬼太郎』等）

また、実務的な困難点としては、既存の MADB（β版）に格納済みのレコードとリスト DB との ID の紐付け作業がある。本質的には、作品本数 10,968 件、各話本数約 47,000 件を 1 件ずつ目視で両 DB のレコードに対して双方向の ID を付与する必要がある。ただし、MADB（β版）の前身である MADB（開発版）が間接的にリスト DB を基盤に構築された経緯を持つために、ある程度は名称の完全一致での機械的な紐付け作業が期待できる。

将来的な問題点としては、今後の「リスト DB」への変更が発生した場合の、次回以降の差分反映の手法について課題がある。

2.3　その他の特記事項

地上波、BS、CS で放送されるアニメの全番組を把握し全録サーバで録画する大変さがある。本年期間中においてもリスト制作委員会、杉並サーバ、千代田サーバともに異なるタイミング、異なる理由で時々受信障害や機材トラブルで録画ができない場面が発生した。工事や天候等による停電などを考慮しつつ全録サーバを 24 時間 365 日稼働した状態に置く大変さがある。しかし、異なる場所、異なる方式で録画を行っているために相互に補完できたために堅牢［けんろう］性は著しく向上した。

映像からキャプチャによって画像を 1 秒スライスで切り抜いたあと、クレジットが入っている画像を選び取って下処理する大変さがある。キャプチャ間隔を密にすることで画面の欠落を防ぐことはできたが、そのために抽出された画面数が多くなり、必要なクレジット画面の選抜には手間を要してしまう。また、最終的には、目視での誤字や誤認識を修正する作業があるため、不要な画像を除去したクレジット画面だけをセットで残す必要がある。

近年、OCR 化サービスはインフラ化しつつある。API などは各サービスでバラバラだが、もはや特別なものではない。例えば AWS（アマゾンウエブサービス）はストレージで先行しており、EC2 のフォーマットが確立されていて、どこのクラウドからも使える。これらのサービス動向を眺めていると、ある程度フォーマットが固まってくると、吸収され普遍化、一般化されていく傾向がある。したがって、つなぐ場所を変えるだけで、どのサービスも選べるようになる。それらの技術は自動運転技術への適用を目的に開発されているのもあり、近年は画像をそのまま投入しても、的確な文字を抽出する OCR が増えてきた。情景から文字を抽出して認識する技術が OCR に入ってきたことにより、文字認識能力が大いに向上しているからだ。こうした進化の速度も、OCR の選定や利用の上

で考慮する必要がある。

EPG データ、OCR ともにテキスト化のめどが立ち、現状の目視確認の補助的な機能としての手応えを感じた。しかし、氏名や役職をどのように分離するか、数多くの要素技術が発展途上にあり、これらを組み合わせることで、近い将来に人が分類できるくらいの精度で AI が推論して分類・メタデータ化できると考えている。

アニメ分野においては、現在研究のために採録を行う公的な機関や試みがない。本事業のように、現在動いている作品に関して、ほぼリアルタイムにデータを取れる仕組みを構築できれば、いずれ学術機関も乗ってくれるようになるだろう。また、自動化できるようになっているので、在野においても追従する人はいるだろうと考える。そういった人のために道をならす意味で、EPG のテキスト化、1 秒スライス化、OCR によるテキスト化を行えるようにしておく意義がある。

第 3 章　実施内容

3.1　会議開催

3.1.1　定例会議

本年は新型コロナウイルス感染症の感染防止の観点から、実際に集まっての会合は避け、ネット会議ソフトの Zoom を利用して参加メンバーによる月次の定例報告会議を開催した。

令和 2 年 5 月 19 日　内部会議（第 1 回）
令和 2 年 6 月 19 日　内部会議（第 2 回）
令和 2 年 7 月 21 日　内部会議（第 3 回）
令和 2 年 8 月 28 日　内部会議（第 4 回）
令和 2 年 9 月 29 日　内部会議（第 5 回）
令和 2 年 10 月 22 日 内部会議（第 6 回）
令和 2 年 11 月 19 日 内部会議（第 7 回）
令和 2 年 12 月 25 日 内部会議（第 8 回）
令和 3 年 1 月 20 日　内部会議（第 9 回）
令和 3 年 2 月 10 日　内部会議（第 10 回）

3.1.2　リスト制作委員会の採録手法のヒアリング

リスト制作委員会の作業方法把握や OCR 化、EPG の CSV 化によるテキストデータの精度確認を目的に、原口氏、磯部正義氏らを交えた Zoom でのミーティングを随時開催した。

令和 2 年 7 月 14 日　リスト制作委員会・磯部氏ヒアリング
令和 2 年 8 月 20 日　リスト制作委員会・原口氏ヒアリング①
令和 2 年 10 月 15 日 リスト制作委員会・原口氏ヒアリング②
令和 2 年 11 月 12 日 リスト制作委員会・原口氏ヒアリング③
令和 2 年 12 月 17 日 リスト制作委員会・データ検証会議

3.1.3　その他会議

上記以外の会議については適宜行った。

令和 2 年 7 月 10 日 OCR 会議①
令和 2 年 10 月 5 日 OCR 会議②
令和 2 年 12 月 3 日 OCR 会議③

令和 2 年 4 月 1 日　記録作成のための方針会議
令和 3 年 2 月 9 日　報告書執筆会議

令和 2 年 11 月 20 日 中間報告会
令和 3 年 2 月 19 日　最終報告会

個々の作業においては、ネットワークを接続して、千代田サーバ、杉並サーバで録画した番組映像を基に、付随する EPG データのテキスト化に必要なプログラムの作成、OCR 化の手順の策定や必要なプログラムの作成、それぞれに採集できたデータの検証を行い、付随する相談、検討については随時行った。

3.2　クレジット画面抽出方法の確立（1 秒キャプチャ）

アニメ作品のクレジット画面を OCR にてテキスト化する際に、人間は文字を文字として容易に、無意識に認識できるものの、機械的な意味付けや分類を行うには、まずは動画からクレジットが表示されている画面を抽出することが必要と考えた。

地上デジタル放送のアニメ番組は、大まかに「OP」「前提供表示」「CM」「本編映像（A パート）」「CM」「本編映像（B パート）」「ED」「次回予告」「後提供表示」「CM」のような順番（フォーマット）で表示される。そこで「OP」「ED」に部分の映像に限定して抽出することで、抽出作業負荷の低減ができるのではないかと考え、幾つかの作品をサンプルとしてフォーマットの解析を行った。解析を行ってみたところ、同一作品であってもフォーマットが異なる場合や、フォーマットそのものは同一でも所要時間が異なるなど、実質的に録画番組数分の違いがあることが判明した。

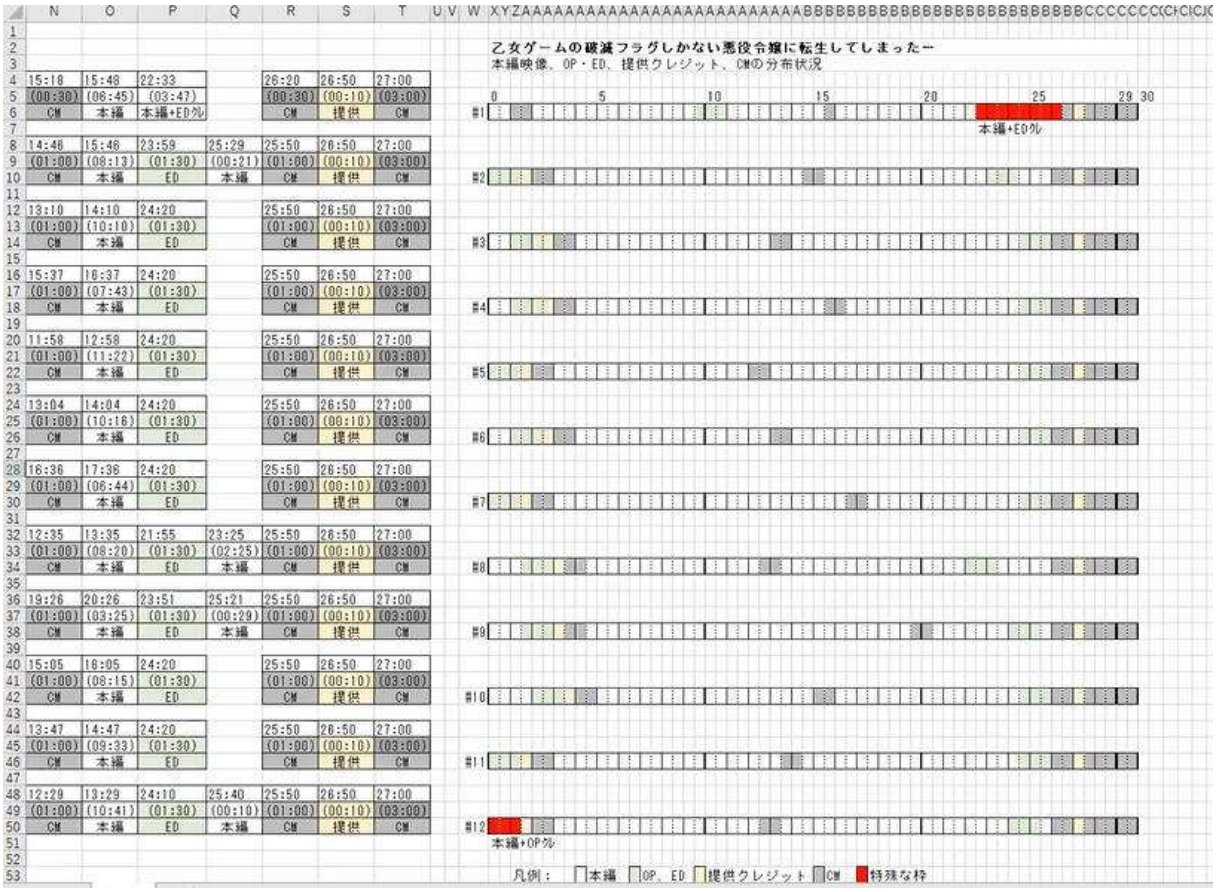


図 3-1　番組フォーマットの分析例

地上デジタル放送で放送されるアニメ作品は概算で年間 200 作品、話数換算では約 3,000 話、放送時間数では約 9 万時間にも及ぶため、効率的にクレジット画面を抽出するために番組ごとにフォーマットを調べて OP や ED 部分を指定することはかえって非効率であることが判明した。

そこで、映像全部（最初から最後まで）について、プロトタイプとして 2 秒間隔での画像抽出の機能を作成し、抽出された画像について評価を行った。抽出間隔については、クレジット画面の欠落が散見できたため 2 秒ないし 1 秒間隔の選択式とした。対象映像形式としては、圧縮効率に優れた MP4 形式を当初は想定していたが、復元時の圧縮ノイズ（画面内、時間軸方向ともに）が発生し、後段の OCR 処理において悪影響が想定されたため、TS ファイル（DR モード録画ファイル）と MP4 の選択式とした。抽出画像の保存形式としては、当初は JPEG 画像を想定していたものの、保存時の圧縮ノイズが想定できたため、JPEG と可逆圧縮の PNG 形式の選択式とした。

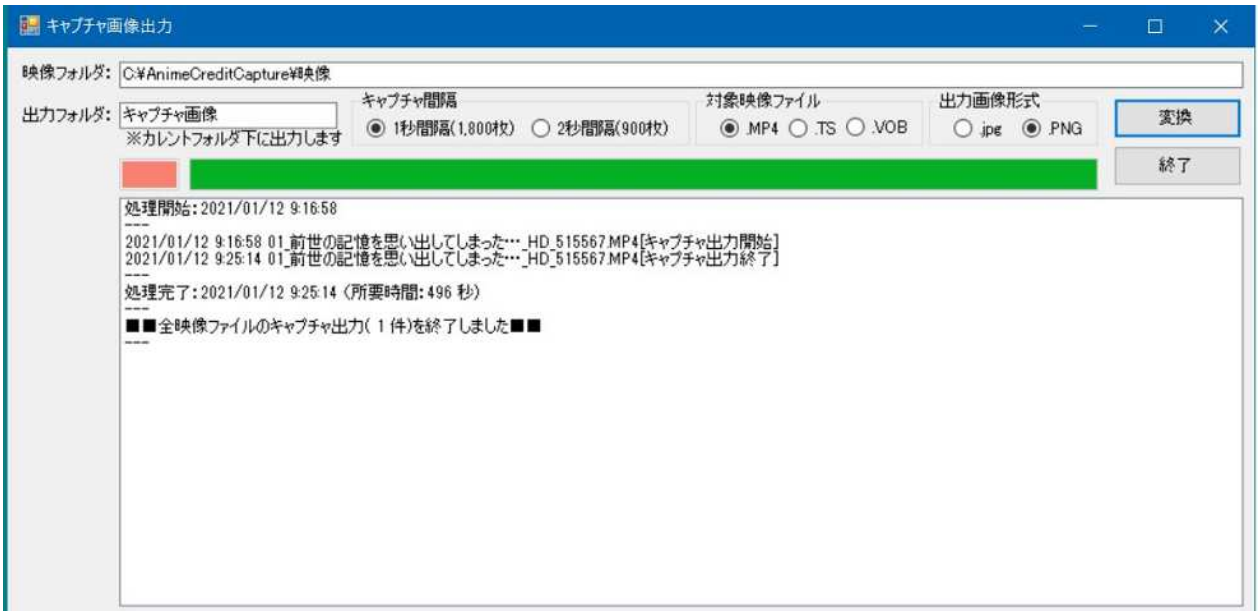


図 3-2　1 秒キャプチャの動作画面

また、杉並サーバは、映像ファイルが同一フォルダに蓄積されるため、当初の設計で機能を十分に満たしていた。さらに、下位フォルダに格納された構造に対しても処理できるよう改良を加えた。

抽出される画像ファイルは「映像ファイルのフォルダを作成し、連番のみのファイル名」となる。正確なラップタイムではないものの、1 秒間隔で出力されるために、ほぼラップタイムに近い秒数になっている。正確なラップタイムとならない理由は、デジタル放送の録画システムの仕組み上で、規定時刻よりも 15～30 秒程度手前から録画開始されてしまうことと、番組フォーマットの理由から映像のみの本編時刻と放送時刻とのズレが生じるためである。

3.3　クレジット画面からテキスト抽出の試行

クレジット画面からテキスト抽出を行うに当たって、確実に必要となるのが OCR のシステムである。本事業では、アニメ映像から文字情報を抽出するのに適した OCR を、パッケージソフトや WEB で提供されている OCR サービスなどに求めた。

●WEB 系

- Google（Google Cloud Vision API）
- Amazon（Amazon Textract）
- Microsoft（Computer Vision API）
- LINE（CLOVA OCR）

ほか

●パッケージ系
・Panasonic（『読取革命』→ソースネクストへ開発・販売を移管）
ほか

これらからどのサービスを選択するか検討を行い、OCR 選定のポイントとして下記要件をピックアップした。

●選定ポイント
・画像内文字認識力
・文字認識精度
・API レベル
・ライセンス形態
・コスト

アニメ映像から画像を切り出す場合、クレジットの文字が映像の上に乗っているケースが通常となる。そのため帳票や名刺など、無地の上に書かれた文字の抽出を目的とした通常の OCR 化とは異なる能力が求められる。提供されるサービスモデルが、性能は高いものの大量のテロップを読む作業とは合わない場合もあるため、選択に当たっては考慮が必要となる。

動画は時間とともに情報が変化していく。背景が動き、表示されている文字の位置も動き背景も動くため、ピンポイントでどの画面のどの部分を狙って読み取ったら良いのかを決めることが作品ごとに異なるため、一律な定義（設定）が難しい。

文字の背景がノイズとして認識され、フォントも作品によって異なるため、文字だけの抽出に特化した、一般的な OCR では能力的に弱いところがある。少しでもノイズがあると、それを文字として認識してしまい認識エラーが多発してしまう。文字そのものにも装飾があったり、デザイン上で埋め込んであったりといったものがある。テレビ画面をキャプチャして静止画にしたあと、画像処理ソフトで前処理を行った方が良い場合がある。文字が書かれている部分だけを枠で囲んで抽出したり、文字と誤認識されそうなゴミを取ったりすれば、より高い精度で読み取れるようになる。これを 1 枚 1 枚、手入力で行うのは手間がかかる。

文字の読み取り精度も重要なポイントとなる。人名など固有名詞には異体字も多くあり、これらを正確に読み取り記録していく必要がある。

画像に対応した文字がそのまま帰ってきても、意味のあるテキストにはなってはいないため、メタデータ化（正しい意味付け）が必要となる。独特な文字組みが行われているものもあって、文字とし

て読み込めたものの、メタデータ化した場合、右と左で独立した別の語として認識し、一文にならないものもある。

こうした場合、パラメータを操作して、必要な情報を読むようにカスタマイズしなければならない。読み取る範囲や、読み取ったあとのデータの表示など、OCR の利用で読み取りたい情報を指定するパラメータの設定などについて、API（Application Programming Interface）がどこまで対応しているかも、選定のポイントになる。コントロールできるレベルがどの程度まであるか、プログラムを組んだとしてもしっかりコントロールできるかなどの条件により、使用感が変わってくる。WEB 系 OCR の API では、こうしたカスタマイズのサービスに対応していない場合が多く、自分たちでカスタマイズした学習記憶をほぼ持てない。AI によって追加学習させる仕組みを独自に作り、追加できれば良いが、複雑で投資も必要になるため、今回の事業範囲ではコスト的に難しい。パブリッククラウドのサービスなら、ライセンスは簡単に受けられるが、映像からのキャプチャ画像に特化した利用となると、パブリッククラウドが提供している一般的なサービスとは、用途が違ってくる。ライセンスを受けることで、API の利用やカスタマイズに必要な情報が得られれば良いが、ほとんどが公開されていない。スタートを切るまでのハードルが高い上に、スタート後も価格が莫大［ばくだい］になると予想される。時間的な制約、見積りの不透明さ、契約プロセスの問題、公共性などを勘案すると、ライセンスが必要なものは利用しづらい。

料金の請求形態には買い切り、サブスクリプション、読んだ画面の枚数や解析した文字量による従量制などがある。ソフトのインストールパッケージは買い切りとなる。機能が高い一方で価格は安いものは安いが、性能的に全録サーバ事業における OCR 化で要求しているレベルには至っていない。パブリッククラウドは請求形態に従量制、サブスクリプションなどがある。多いのは従量制で、料金は文字数やページ数による。イニシャルコストによって契約金が必要になるもの、初期費用を要求するサービスなどがある。ライセンスで契約を結んでも、使い続けていると保守費用が必要となる。サブスクリプションではランニングコストがかかる。部分的に識別性能が高いものがあるが、プロバイダ側が用意しているサービスプランと、本事業のようなキャプチャ画面からクレジットを採取するような作業とはマッチしない。クレジット採取はそれだけ異質な作業と言える。プロバイダ側にサービスメニューがない場合、値付けが先方任せになっていて、採算が合いづらくなる。個別契約を求めても良いが、契約の問題、ライセンスの問題が浮上してくる。クオリティを上げてもらう場合も、費用を 1 回払えば済むものではない。放っておいても性能はどんどんと上がっていく。それに見合った額になるのかを検討する必要がある。

クレジット画面のテキスト化は長く続ける活動となるため、永続的にサービスが提供されるのかも重要になる。従量料金もネックになる。過去のタイトルと現在のタイトルを合わせて 1 タイトルのデータとして考えた場合、これを従量制の中でデータを投入した場合、コストがどれくらいかかるか見えなくなる。どういった作業フローで、効率良くサービスを利用し、文字データ化していくの

か、1 回の利用ではなく、自動化を含め永続的に利用できるのか、性能バランスが良いのか、スケーラビリティを持っているかが、OCR 選択上での要点となる。

次々と機能が拡張されていくサービスがある一方、発展がないサービス（製品）もある。今後のスピードアップ、機能面のスケーラビリティが見込めるかどうかも選択の上で重要となる。今は 10 分かかっているものが、スピード化によって時間が半分になれば効率が上がる。これらを勘案して選ぶ必要がある。

これらの検討を経て、全録サーバ事業における OCR 化では、Google Cloud Vision API を選択した。理由は、API 情報が取りやすいこと、利用までの契約ハードルが低くスピードが速いこと、圧倒的に手軽であったことが挙げられる。スケーラビリティも能力もかなり幅が広い。文字だけでなく画像に対する情報も得られる。OCR に限らず、様々なものからデータが取れる。メタ化に対してあらゆる手法を試すため、検討する上の材料を得られる。従量性でコストもそれなりにかかるが、下げる方法も試していける。サービス選択に当たってのテストでは、平均的なサービスを従量性で使った場合を各種サービスで比べ、発生コストの低いサービスを選択した。

3.4　抽出されたテキストの検証

Google Vision API テスト条件

・元データはアニメ番組（1,440×1,080dpi）をデコードした画面を PNG 保存、表示したものを入力ソースとしている。

・今回はトリミングによる精度向上性の検証テストも行った。

・素材の質の向上テストの一環として、0.25 秒スライスでの時間軸による同一多素材での差を行った。

Google Vision API テスト結果について

・優れている点

　・識精度の向上スピード

　・縦書き誤認識の少なさ

　・性能バランス

・弱点

　・文字認識の欠落

性能向上のスピードは目を見張るものがあり、癖のあるデザインフォントや背景干渉等の不利な条件下でも極めて良好な結果で、現時点では LINE OCR にアドバンテージはあるが徐々にその差は縮まりつつある。認識精度は、事業開始直後よりもどんどんと上がっている。当初の理念にはまだ追いついていないが、時間の問題だろう。LINE と違い、縦書きを誤認識せず性能バランスが良い。弱

点として、ゴミを文字として認識してしまうこと以外に、文字を文字として認識せず欠落が出てしまうことがある。形がシンプルなものが欠落する傾向にある。欠落に関しては、文字を文字として認識せず図形として認識する。『100 万の命の上に俺は立っている』のエンディングを例に挙げると、使用されている直線で組まれたような独特のフォントに対して、欠落及び誤字が度々見られた。これをどうなくすかが課題だ。旧字体については人間よりうまく拾う傾向にあった。

トリミングした場合、時間で変化しているため、2 秒スライスでは 1 枚しか取れないところを 0.25 秒間隔で取り、クレジットが表示されている部分を 10 枚弱、すべて入力した。トリミングにより誤字・欠落部分の認識、精度の向上が見られた。鮮明になる反面、それらをゴミと認識し S/N 比が低下した。トレードオフ的なものがあるが、拾えていなかった部分が拾えるようになった。トリミングで精度が向上する一方で、S/N 比が低下するといった事例は、OCR が簡単な話ではないことを表す。OCR が一般化していない現時点の段階では、こうしたノウハウは理解されない。現時点では、何となく使えるという話が落としどころになる。

3.5　自動化検証

3.5.1　クレジット画面抽出方法の確立（1 秒キャプチャ）

日時バッチ処理での動作を想定して、画面インターフェースを持たない実行モード（UI レスモード）について機能拡張を行った。各種の設定値については INI ファイルに記載することで、処理そのものは自動処理を行うようにすることができるようした。初期値については、多くの映像ファイルを参考にして「キャプチャ間隔 : 1 秒、対象映像形式 : TS ファイル形式、出力形式 : PNG 形式」を採用した。クレジットの表示傾向によりパラメータを変更する映像ファイルについては、UI モードにて個別に作業するような操作を想定している。

日時バッチの起動については Windows のタスク管理画面から設定し、番組録画の少ない時間帯（10～15 時）に稼働することで、アニメ全録処理を行う PC の負荷に配慮した。

3.5.2　OCR

OCR 処理については、クラウド上で実行するため、操作する PC は Windows、Mac を問わない。OCR 化のプログラムは Python で組んでいる。Python が動作しインターネットに接続可能な環境なら実行できる。今回の OCR 化で使用している Python のバージョンは 3 以上であった。

【Python3 の構築方法】
Windows　https://www.python.jp/install/windows/install.html
Mac　https://qiita.com/ms-rock/items/72b8f1abc661c539bb09

プログラムを走らせる Python に重ねるインターフェースとして、Jupyter Notebook を使ってい

る。Jupyter Notebook とはブラウザ上で動くデータ分析ツールで、マスの中にプログラムや文字を入れると、結果がブラウザ上で返ってくる。

【Jupyter Notebook】
https://www.javadrive.jp/python/jupyter-notebook/index1.html

OCR 化作業の環境構築として、PC に Python と Jupyter Notebook を入れていく。今回は各マシンでローカルに作業環境を立てたが、ネットワーク上で利用するならどこに作業環境を立ててもかまわない。ただし、ストレージに Google Drive を利用しているため、サーバの容量を必要とする今回の作業で、個人が持つアカウントと同じものを使用すると容量が厳しくなる。

Google のアカウントを持っていれば、自前で環境を構築しなくても、Google Colaboratory で稼働させるのが手軽で早い。すべて Google 上で解決する。環境構築のハードルは高くない。ただし、Google のアカウントを取った上で、Google Cloud Platform（GCP）で Colaboratory を使う、OCR を使うといった設定が必要になる。

【Google Colaboratory】
https://www.tdi.co.jp/miso/google-colaboratory-gpu

利用に当たってプロジェクトを作ったり、API を有効にしたりする必要がある。サービスアカウントキーは API を使用するときに必ず必要となる。利用者が誰かを把握するもので、それを基に課金が行われる。ただし、普通に（私的に）使う程度の分量であれば無料の範囲で行うことができる。クラウド環境を使えば、手元の PC も高機能なものは必要ない。ウェブ上で見積りも行える。ただ、Google の仕組み上、アカウントをふだんから自分で使っているものと、作業で使うもので分けられない。学校などで使う際には対応が必要である。

OCR 処理を実行するスクリプトそのものについては、別紙を参照。

「Credits OCR for Google Cloud Vison colab version」

初期設定（セッションごとに 1 回だけ実行）
①Google Vison OCR Setup の実行
②Vision API サービス アカウント キー ファイルのアップロード

画像指定方法の選択
③画像ファイルをアップロード、又は、Google Drive に格納されたフォルダを指定する

OCR の実行
④解析の実行

出力データ
⑤画像ファイルのアップロードを選んだ場合はデータファイルの自動ダウンロードが開始される
Google Drive を利用した場合は、該当フォルダに出力結果が出力される

3.5.3　EPG 結合

EPG ファイルは 1 番組ずつ 1 ファイルとして生成される。レコーダーやテレビの番組情報のボタンを押すと、映る画面に出てくる情報と同じものである。項目としては日付、時刻、開始時刻、放送局、番組タイトルがある。EPG の詳細に出てくる項目が全部入っている。EPG ジャンルは仕組み的には三つまで付けられる。それぞれ大と小のジャンルが取れる。画面解像度・音声情報も取得できる。

これらのファイルを一覧表としてまとめる場合、EPG ファイルは書誌に近い意味を持つために、これらの項目を転記する形でデータを保存していくことになるが、これを 1 個ずつコピー&ペーストして一つの CSV ファイルにまとめる作業は大変なものとなる。

取得した EPG テキストは、そのままではメタ化されておらず、使いづらいので細かく修正をかける必要がある。番組のタイトルについては、一例では、TBS の「スーパーアニメイズム」という枠の中に入っている『アルゴナビス』という番組、その第 1 話は、EPG では枠の情報、作品タイトル、サブタイトル、話数表記、字幕有無など重畳するときのデータの形状が入ってくる。話数表記にサブタイトルが入ってくるものもある。人間であれば、これは作品タイトルでこれはサブタイトルだと認識できるが、機械的に扱おうとすると面倒である。枠が決まっている番組は、自動的に切り分けるようにしている。『ボス・ベイビー』では、枠タイトルはないが、作品タイトルとサブタイトルと補足情報（2 か国語、字幕）が入る。放送局から送出される大元の EPG データは人間がデータを作って入力しているものなので、不定型なデータが入ってくる場合がある。閉じカッコがない、事前に見ていた枠の情報と、実際に収集された情報が微妙に違う、といったことが起こる。しかし、EPG は（テレビ局の編成／放送部の）人間が作るものなので、ある程度類型化できているため、過去（令和元年 10 月以降）に蓄積した EPG を参考にして、プログラムによるコーディングで条件分岐処理を行い、メタ項目に振り分ける機能を実装した。

また、特殊な留意事項として、開始終了時刻については、テレビ局の運用に合わせ、1 日の表示を 4 時スタート、30 時終わりとしている。これは、火曜日の 25 時放送のアニメと言われた場合、水曜日午前 1 時とは認識しないためである。そのため実際の絶対日時とは多少のズレが生じる。

第 3 章　実施内容

EPG からのテキストデータ化で、リスト制作は十分のように思われるが問題がある。放送データは二重三重にチェックされているが、ラテ覧（テレビ欄の意）のデータは多重チェックが徹底されておらず信用性が薄い。収集することに意義はあるが、研究機関が右から左に流して良いかと言うと、信頼性が担保できない。それでも、EPG からのデータだけでも DB 化されて自由に使えるようになれば、アニメ放送の全体像をつかめるようになる。また、EPG からのデータは、実際に放送された記録という意味では役に立つ。EPG を使えば、最低限のデータがそれなりのスピードで取れる。

プログラムについては、所定のフォルダ直下に存在する EPG ファイルについて CSV ファイルに統合を行う。その際、番組タイトルについては、以下のような解析処理を行い、メタ化を行っている。

①[新][字][終][再][デ][二][多]削除

②アニメ（全角スペース）ミニアニメ（全角スペースを削除）
③枠抽出
【】か＜＞か（）に囲まれた下記の文字
フジバラナイト SAT
アニメイズム
スーパーアニメイズム
アニおび
ノイタミナ
天てれアニメ
＋Ultra

ただし、2 つの枠が書かれた場合は全角スペースで連結
例、フジバラナイト SAT　ノイタミナ

④枠抽出 2
AnichU

ただし、既に枠が入っていた場合は全角スペースで連結

⑤話数抽出（複数話できるだけ対応）
数字として認識する文字
０-９0-9 一-十．零壱弐参拾

話数として認識する形
　総集編
　＃か＃に続く数字
　第から始まって数字、話と続く文字
　（）に囲まれた数字

〇〇～〇〇、ないし〇〇・〇〇の表記があれば使う

なければ、話数をできるだけ捜し、全角スペースで連結

⑥上記抜き出し後、空になった「」【】『』＜＞があれば削除

⑦サブタイトル抽出（複数話できるだけ対応）

⑧文末に、「スペース★,、・」があれば削除

⑨一般名称テレビアニメ、アニメを一旦消して
「〇〇」だけになってしまうときは処理を行わない。

⑩文末から「〇〇」を探してサブタイトルに追加
既にサブタイトルがある場合は、全角スペースで連結

⑪また文末の「スペース★,、・」があれば削除
以降、サブタイトルがなくなるまで抜き出し

上記のアルゴリズムを使用して、番組名を以下のように変換することができる。

EPG 番組名：グランベルム　＃2「私がここにいるために」【アニメイズム】[字]

枠タイトル：アニメイズム
作品名：グランベルム
話数表記：＃2
サブタイトル：私がここにいるために

3.6　リスト DB の変換

3.6.1　リスト DB の概要

「リスト DB」とは、原口正宏氏を代表とするリスト制作委員会が、テレビ番組を録画し、クレジットデータを抽出して記録してきた「dBASE」上に構築したカード型データベースのことを指す。ATAC 及び JAniCA では、全録サーバで録画した番組からクレジットデータを抽出する方法を検討している。「dBASE」は相当に古いシステムで、現状のグラフ型 DB には形が合わないため、それを接続するのが目標である。

DB の概要をつかむ情報として作品本数と各話タイトル数、人名などがある。人名も自然人や法人が混在する。それが役職ごとに違うこともある。製作として関わる人、スタッフとして関わる人、声優、音響などがある。役名のレコード件数が非常に多いが、これは作品×話数×登場人物となるために延べ人数として件数が多くなる。そのため 22 万件にも達していた。他の人名は、延べ人数と実質の人数が乖離［かいり］していく。曲情報のレコード件数が意外と多かった。オープニング（OP）やエンディング（ED）、挿入歌まで入力されているからである。他の DB では曲情報を採録しているところが少ない。

最古作品：大正 6 年（1917 年）『芋川椋三 玄関番の巻』
最新作品：当時放送中の『ゲゲゲの鬼太郎[第 6 作]』など
作品本数：約 10,968 作品
各話（サブタイトル）数：約 47,000 件　※140,990 件から推計
人名（製作等）：約 22,298 名　※法人、個人、重複、表記揺らぎ等を含む
人名（スタッフ等）：約 71,300 名　※法人、個人、重複、表記揺らぎ等を含む
人名（声優等）：約 14,971 名　※重複、表記揺らぎ等を含む
役名：約 221,259 名
曲情報：約 5,714 件　※重複、表記揺らぎ等を含む

DBF というファイルは、作った順に作られているファイルのために、整理された公開順ではないことに注意が必要である。一見『白蛇伝』から並んでいるよう見えるが、この作業をする当時に入れやすい順に登録したためにこの辺りは時代順になっている。大正や昭和の初期の作品は後日に登録している。その時点で判明したものをその都度入れていったものとなる。

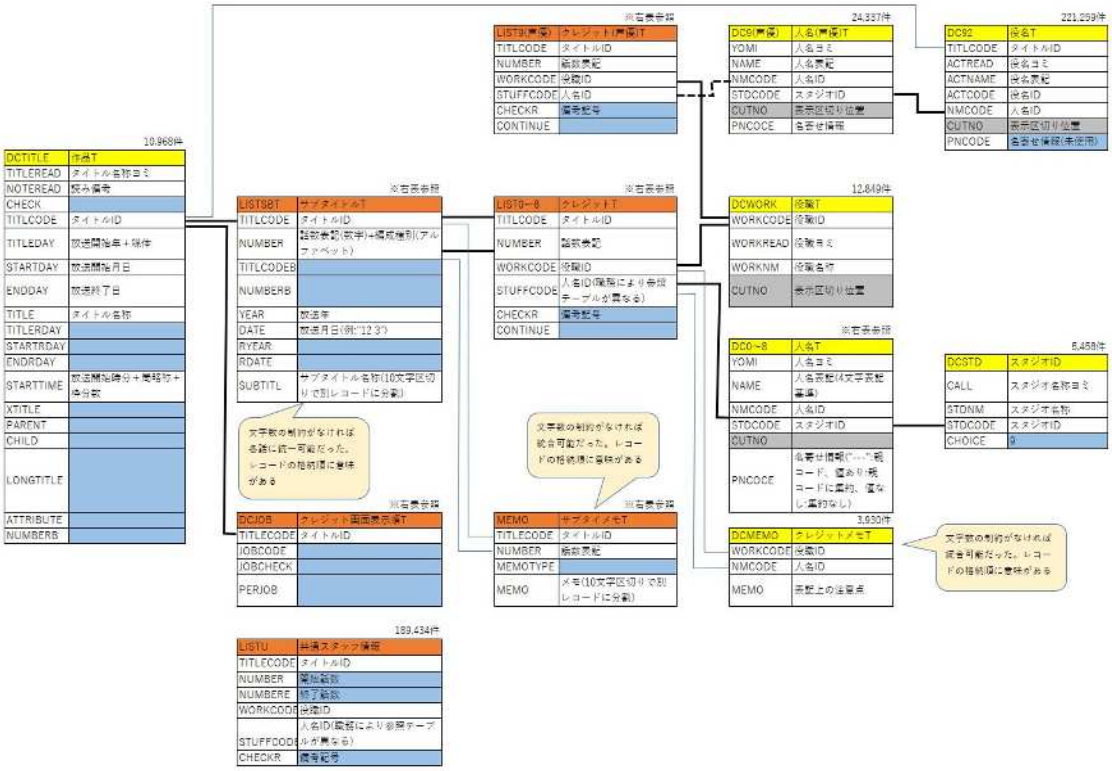


図 3-3　リスト DB のテーブル関連図

表 3-1　リスト DB の各テーブルの格納件数

フォルダ	ファイル名	格納データ	件数	重複除去	備考
ルート	DC0	脚本	11,753	8,909	
ルート	DC1	演出、監督	12,717	8,069	
ルート	DC2	デザイン	0	0	未使用
ルート	DC3	作画	47,167	32,019	
ルート	DC4	美術	10,213	6,815	
ルート	DC5	仕上	12,513	10,453	
ルート	DC6	撮影	6,560	5,035	
ルート	DC7	音響	30,320	20,871	
ルート	DC8	制作・製作	27,914	22,298	
ルート	DC9	声優	24,337	14,971	
ルート	DC92	役名	221,259	-	
ルート	DCTITLE	作品名	10,968	-	
ルート	DCWORK	職名	12,849	-	
ルート	DCSTD	スタジオ	5,458	-	
ルート	DCMEMO	登録上のメモ	8,464	-	
ルート	LISTU	シリーズ共通スタッフ	189,434	-	

フォルダ	ファイル	職務	96	95	94	93	92	91	90	89	00	小計
各年代	LIST0	各話の脚本	102,906	2,844	3,041	2,550	3,243	3,296	13,629	0	0	131,509
各年代	LIST1	各話の演出、監督	196,363	5,179	5,121	4,620	5,719	6,079	20,735	0	0	243,816
各年代	LIST2	デザイン	0	0	0	0	0	0	0	0	0	0
各年代	LIST3	各話の作画	403,012	12,385	23,214	14,261	18,674	20,629	76,077	0	702	568,954
各年代	LIST4	各話の美術	155,016	4,726	9,312	5,753	8,050	9,076	29,919	0	0	221,852
各年代	LIST5	各話の仕上	32,729	4,114	11,240	6,517	8,676	8,849	36,142	0	697	108,964
各年代	LIST6	各話の撮影	29,019	3,024	8,735	5,047	6,650	5,925	23,838	0	0	82,238
各年代	LIST7	各話の音響	25,282	984	2,121	1,387	937	482	6,932	0	0	38,125
各年代	LIST8	各話の制作・製作	78,319	3,568	9,201	5,228	7,183	6,938	16,953	0	6	127,396
各年代	LIST9	各話の声優	554,231	36,509	28,620	13,172	20,145	19,642	97,604	513,490	0	1,283,413
各年代	LISTSBT	サブタイトル	114,443	3,660	3,123	2,808	2,894	2,921	11,124	0	17	140,990
各年代	MEMO	サブタイメモ	1,547	1,222	979	607	893	789	1,906	0	0	7,943
各年代	DCIOB	クレジット画面表示順	200,952	7,282	7,918	4,936	5,154	4,792	19,642	8	28	250,712

3.6.2　リスト制作委員会へのヒアリング

本事業ではリスト制作委員会の原口氏、磯部氏に対して、4 回のヒアリング会議を行った。また会議とは別に適宜メールや電話での補足調査を行い、リスト DB の詳細な分析を行った。

リスト DB は、フォルダの中のテーブルで、具体的にデータが入っている。テンポラリファイルを除き、有効な意味があるデータが格納されているファイルだけを抽出して解析したものが上記の表となる。DC0 や DC8、DC9 といったネーミングルールである。

DC0 は原作脚本系の人名を入れる辞書（マスタテーブルの意味）である。ここには、シナリオライターや漫画原作者、ライトノベル原作者といった、人のクレジットに出てくる人名を整理する。DC1 は演出家、監督、演出やチーフディレクターなど演出家に当たる人たちの人名が入る。DC2 は、初期にキャラクターデザインをする人、デザインをする人、作画をする人を分けようとして作ったものである。しかし現在は、デザインと作画を全部 DC3 に統合し、DC2 は空席になっている。近年 CG 関係のスタッフを分類する必要が出ている。作画に整理するか撮影か仕上げかグレーゾーンの人たちが増えた。空いている DC2 を使う検討をしている。DC3 はデザインや作画関係、DC4 は美術背景、DC5 は色彩設計・仕上げ、DC6 は撮影と編集関係、DC7 は音響関係、DC8 は制作、プロデュース関係、DC9 は声の出演。

順番はおおむねクレジットの表示順に連携している。アニメーションができあがっていく順番と考えて良い。0 番から 9 番まで順番にたどると一つの作品が完成するイメージとなっている。

DC92 には役名が入り、DC9 とリンクしている。最終的に DC0 から DC9 は存在している人名かスタジオ名だが、DC92 は現実に存在していない役名で、キャラクター名と言える。それと DC9 は 1 対 1 で対応する。サザエを加藤みどりが演じた場合、加藤みどりが含まれるのは DC9 だが、サザエという役名が含まれているのは DC92。サザエという役名のところに、演じているのは加藤みどりと分かるようなコードナンバーが記録されている。

ただし、サザエという役名がどの作品のどこに出たかは記録されていない。加藤みどりがどの作品のどの話数に出てきたかが実際には記録されている。それとサザエが 1 本の糸でつながっているので、自動的にサザエが加藤みどりの情報と一緒にリストデータに書き込まれる形でリンクしている。

DC1 桁の数字は人名集計をするときに必要となる。ある人の仕事を集計させるとき、まず脚本関係、演出関係といった分野ごとに分かれていると、集計のときに雑多な情報が紛れ込んでない。そういうメリットがあるため、大きくグループに分けている。

DC9 は声優の仕事歴を見なければいけない場合があるので、声優名が整理されている情報を検索できるようにしてある。そこに役名が混じってくると、現実に存在しない人間があいうえお順の間に紛れ込むため、役名は別（DC92）に用意した。

DCWORK は役職名。監督、チーフディレクター、シリーズディレクターと呼び方が変わるごとに

別の役職名になるので､役職名が DCWORK に入っている｡DCWORK の中に最終的に先ほどの DC0 から DC9 までが入り、どのカテゴリーに所属する役職名かが分かるような仕掛けになっている。

人名の DB を作るとき、全量でなければ意味がない。すべての期間を集めてきて、そこから手がけた仕事を引っ張り出してくる必要がある。

4 桁の西暦ナンバーのあとにアルファベットの A や E が付いている。E は映画の E、劇場作品。A はテレビ作品。V は OVA。ここを追加すれば配信作品もカバーできるだろうと考えている。配信作品だけ別の記号にすれば 4 番目のグループができる。カテゴリーが不明の作品にアルファベットを振ることも可能。あとからでも修正が簡単にできるよう、5 桁目のアルファベットが存在している。

D 列に半角 3 文字で「ﾜﾜﾚ」「ﾜﾜﾝ」という文字が出てくる。コードとして使えるものを洗いざらい使っている。その際に、プログラム的に誤動作しない記号を使っている。これらは自動的に作られる。プログラムが勝手に振っていく。機械が勝手に決める。新しい作品を登録したときに、前の登録を見て勝手に採番される。

各話タイトルには、全角「／」が入っているが、これは改行を意味する。文字数も 1 カラムに全角 10 文字しか入っていない。サブタイトルは何行になっても良いような仕組みになっている。長いサブタイトルは長いだけ行数を取る。

同時に、表組みとして打ち出すときのことも考えている。横 10 文字は、余りに横に広いと表として見づらくなる。表は横書きで 1 話分が縦の帯になっているもの。以前の ATAC の打合せで『ハッスルパンチ』のリストを見せたときに作ったものと同じ。現状で最も見やすい記録方式として、横に 8 話分並べている。縦は役職が長く伸びていく表組みを作った。

『サザエさん』の「メガネをとったパパ」に関して言うなら、サブタイトル、「メガネを／」のあとに「とった」とつなげても良いが、見栄えを画面に近付けられたら、それに越したことはない。だから 2 行目に「とったパパ」を入れたが、その間に何もなければ、連結されたときに改行されたという情報が失われてしまう。力ずくで改行マークが残るよう「／」を入れた。DB が 1 行目、2 行目、3 行目､「メガネ」をとったパパ」「作品ナンバー」「サザエさんの 1,350 回の A パートサブタイトル」を示しているように出力する。今の「年間 PD」に当てはめるとき、自動的に「／」の次、の右のスペースはトリミングして、2 行目に「とった」がすぐにつながるようになっている。プログラムを変えるだけで 1 行になったり 3 行になったりする。

なるべく詳細な状態で最初のデータを作り、後にそこから省くことは簡単にできるが、最初に省いていたデータを後に精細にするのは大変という思想となっている。[縦書] と入れているのは､「メガネをとったパパ」が縦書きで表示されたという印である。何もないときは基本的には横書き。世の中のサブタイトルは圧倒的に横書きなので、特に説明がない限りは横書きが前提となる。[縦書] という文字は必要に応じて省いて出力できる。

「メガネをとったパパ」が 1 行になっているデータは巷［ちまた］にはある。それで世の中は問題ないと思っているが、原口氏は、やがてこの改行自体に意味があると思われるかもしれないと考えており、こだわって記録していく。こだわって作業をすることで、ほかの箇所にも目が行くようになる効果もある。ここで改行するのだと神経をとがらせていると、誤字率も減る。自分の中で負荷をかけることで、普通の人が見逃しそうな部分に関して、精度を高められる。それによって他のミスも救おうとしている。

リスト DB を作るとき、元から改行が必要だと思い作った。改行マークを入れようとした。これの前に手書きでデータを作っている時代があり、これの 1 個手前の、dBASE 体系ができる以前にプロトタイプデータとして作っていた時期が 2 年間くらいあった。dBASE に連結はされていないデータが実はある。当時は、入力していく方法が未完成で、そのときは改行マークとか考えていなかった。単に入力するだけでなく、そのあとにどういう出力をする可能性があるかを考えたとき、あるいはテキスト、文字をベタ打ちするだけでなく、見栄えとして表形式にすることを念頭に置いたときなど、様々な必要性を鑑み、横の文字数をどれくらい取るかも考えて行ったのがこの段階に当たる。同時に改行マークは入れることにした。

DOS レベル PC98M2 の稼働速度とデータのキャパシティで、どれくらいの文字量だったら回せるかを考えたとき、コードナンバーは半角 3 桁にした方がいいと入れ知恵してくれた人がいた。コンピュータオタクで同時に作画マニアの人で、その人に相談して基本的なプログラムを作ってもらった。表にしたいと無理難題を言い、データのレコードのフィールド幅をどうしたら良いかを設定してもらった。1 行のデータの中に必要なデータ、これとこれとこれを入れれば最終的に表にするときに無駄な作業が減るだろうと入れ知恵をしてくれた。

DCWORK について。時代によって役職は変わってくる。今は現像という職業はないだろう。

CUTNO（カットナンバー）という項目がある。これは表組みを作るときに大事で、チーフアニメーターとかオープニングの役職名で長すぎて 7 文字だったか 5 文字で折り返さなくてはならず、表組みのときにチーフアニメーターが機械的に分断され、「チーフアニ」のように切られると気持ち悪いため、「チーフ」を 1 行目、「アニメーター」を 2 行目すると見栄えが良いので、「フ」と「ア」の間で切ってくださいという情報を入れている。カッコの手前で 2 行目に送ってくれと命令するのが「ュ」とか「ォ」。自動的にどこで折るかと聞いてくる機能があり、長すぎるので折りたいと要望を言うと、声の出演だけを消して、第 1 画面を消してみせると、ここから 2 行目にしたいのだろうとコンピュータが自動的に考えて、「ュ」とか「オ」という文字を書き込む。

役職名について。監督の役職コードはどの作品も「1!5」で同じ。監督と入れば「1!5」。人名のスペースの空きにはこだわっているが、監督のような 2 文字の場合は、詰まっている文字が開いているだけ。「監　督」と間が空いていたとしても、画面上は監督しか出ていない。前提として、人名に関して言えば、日本人の人名に多いのは 4 文字で、4 文字を最低限の文字数と考え、配分がどうなっ

ているかを見ながら、画面に忠実に拾おうとしている。だが役職名は、監督なら監督だという意味だと分かる。それ自体、間が空いているかには意味がない。普通に監督として拾うが、中には例外もある。

「美術構成」という役職が『白蛇伝』であるが、「美術」で改行され「構成」が次の行にくる。「美術」と「構成」の 2 つの役職だったのか、「美術構成」という役職が単に改行されただけなのかが分からない。恣意的に決め付けるのも嫌なので、間違いないとの別の根拠がなければ、基本的に 2 行連結されている怪しい役職は、美術構成と拾う。ただし、美術のあとに改行があることは示さなくてはいけない。改行が分かるようにしてある。役職の中で文字空けとか改行に関して、気を付けなければいけないと感じる場合は、忠実に拾っておく。

3 文字が役職だが、2 文字目と 3 文字目の間だけが非常に開いていたりする役職名、そういう場合は開いている部分は、一連の文字のつながりの中で空くので厳密にする。基本的の文字は均等に並んでいる、という状態を大前提と考える。均等の文字幅が何かしら作為的に変えられている場合は忠実に拾う。

「チーフアニメーター」の語が、「チーフ」で改行され「アニメーター」が 2 行目になると、別の役職コードで拾わせている。「チーフ（半角空き）アニメーター」も別コード。「チーフ・アニメーター」も別のコードナンバーになる。ナカグロ（・）があるかないかと同じように、「チーフ（半角空き）アニメーター」も 2 語の間を少し空けたかったのだから、そこはしっかりと拾うようにしている。

人名について、基本は 4 文字としている。クレジットの表記方法で一番標準的なパターンが、4 文字の人間の幅を基本的に抑えているものだった。4 文字より狭いもの、3 文字とか 2 文字の幅を基準にして押し込めるのでなく、4 文字を普通にして名前が 1 文字の人は名字と名前の間をけるというクレジットの作り方を、東映アニメでもどこでも行っていた。

「會川　昇」や「出崎　哲」などで 1 文字、全角文字を開ける拾い方をしているのは、クレジットに普通に出ていたからである。逆に言うと、4 文字基準と言っているのは、高屋敷英雄、高橋留美子、富野由悠季が入ったとしても 4 文字幅にむりやり 5 文字が入るはずである。それがコンピュータの性能上できなかった。最終的に 4 文字幅に富野由悠季を押し込める出力の方法ができれば、それが実際のクレジットに近くなるのだと理解した上で、現行の方式で行っている。

竹書房の頭にハイフン（半角の長音記号）が入っているのは、スタジオ名や出版社名の頭には音引きマークを入れて、あいうえお順で整理巣時に人名の中に会社が混ざらないようにする。会社名だけ集計するときは、頭に音引きがあるものを拾う。

PNCODE。ハイフンで 3 文字の「---」を入れている。これがペンネームの親を表す。ここに記入がないものは揺らぎがない名前。この表記しかクレジット上にないもの。揺らぎがあるのもには「-

--」を入れる。PNCODE にコードナンバーが書かれている人がいる。例えば富野由悠季については、昔の「喜幸」というものがある。その F 欄には、本来の富野由悠季の項の C 欄にある、MNCODE「ヲヲカ」とい文字が入っている。）延べ表記人数ではなく自然人として集計しようとすると、PNCODE に「---」に入っているものと空欄のものを集計すれば、表記揺らぎは排除できる

D 列の STDCODE はスタジオコード。所属スタジオになる。スタジオ名とスタジオのコードナンバーが記入されているテーブルが別にある。人名を登録するときにスタジオ名のコードを記入する。それが人名コードとリンクする。

撮影会社の名前は、撮影のところの 1 行目にデータとしてある。2 行目に所属している人名が表示してあるが、人名と 1 行目との間に連関はない。1 行目のデータと 2 行目のデータというところだけ。その代わり人名のところに、分かる限りは何とかプロダクションという名称を STDCODE としている

楽曲の登録方法については撮影と全く同じである。1 行目 2 行目 3 行目 4 行目に連関関係は入れずに入力しているが、この入力方法は長年の課題である。曲名に対して作詞と作曲と編曲と歌を構造化しても良いかもしれない。役名と同じような関係性を、他の役職で自信を持ってつなぐことができるとすれば歌だ。この曲に対して作詞は結び付けやすい。100%ではない場合もあるが、結び付けやすい。「セント・ジュエルを探せ!」／作詞・森由里子／作曲・タケカワユキヒデ／編曲・山本健司では、森由里子が「セント・ジュエルを探せ!」の作詞家であると本当は分かった方が良いが、現状は作詞・森由里子が『新ビックリマン』のオープニングを何話から何話まで担当したかしか分からない。

声の出演で DC92 があったことと同じように、DC72 のようなものを作り、下にぶら下げるのが理想だが、そこに一概にそこに踏み切れないのは、役名と声優名の関係と違って、作詞や作曲、編曲は曲にまずつながっていることがある。そこからもう 1 回つながって、森由里子、タケカワユキヒデ、山本健司と 3 段構造になる。この 3 段階をつなげる表形式のようなものを構築すればきれいに記録できる。そういったイメージはあるが、今はまだ行っていない。現状の形式で入力しているのは、表形式で打ち出すときに簡単だったからである。歌のデータを、1 行目 2 行目 3 行目で打ち出せば、この形で自動的に表示してくれる。「セント・ジュエルを探せ!」の作詞が森由里子だと分かる仕掛けになっているが、森由里子のお仕事リストを作るのはもうひと手間が必要となる。

3.6.3　変換指針

「リスト DB」（dBASE というカード型 DB に起因）が順番という情報を持っておらず、レコードが登録された順に意味を持っていることが解析によって分かった。格納順番が意味を持っているので、データを統合する際に、現代の一般的な DB に読み込むと順番が崩れてしまう。RDB は格納順に意味を持っていない。「リスト DB」から MADB（β版）に投入する前に、データを整形し、文字コードであったり順番であったりを前処理しないといけない。

半角カナをコードで使っており扱いづらい。Access では小さい「ｧ」や濁音が、全部同じ扱いになっている。「ウ」でも大きい「ウ」と小さい「ｳ」が同じ。元々のデータとして半濁音や濁音は使っていないはずだが、記号の類を無視する。Access や Excel がそういう仕組みになっており、ここの変換手順に手間取った。

原口氏の DB に対する考え方、元々の作業の方向性が、画面のクレジットをそのまま転記し記録することであり、人の名前の表記揺らぎについて、非常に気を使って入力している。ただ、普通の人には余り意味を持たない部分であり、どうやって微細な違いを丸めて MADB に接続するかが課題である。

各テーブルの意味は以下のとおりである。それぞれの件数は表 3-1 を参照。

DC0：脚本
DC1：演出、監督
DC2：デザイン　←実質的に未使用
DC3：作画
DC4：美術
DC5：仕上げ
DC6：撮影
DC7：音響
DC8：制作・製作
DC9：声優
DC92：役名

DCTITLE：作品名
DCWORK：職名
DCSTD：スタジオ　←判明しているもののみ
DCMEMO：登録上のメモ
LISTU：シリーズ共通スタッフ

LISTSBT：各話タイトル
MEMO：登録上のメモ
DCJOB：各話入力が必要なデータを入力する際に、
　　「役職名」の「画面表示順」を記録しておくファイル

LIST0：各話の脚本
LIST1：各話の演出、監督
LIST2：デザイン　←使用している年もある
LIST3：各話の作画
LIST4：各話の美術
LIST5：各話の仕上げ
LIST6：各話の撮影
LIST7：各話の音響
LIST8：各話の制作・製作
LIST9：各話の声優

データ登録上の留意点
・読みで濁点は除去　※ソートで邪魔のため
・法人は読みの冒頭に半角長音"ｰ"を付加　※ソートで自然人／法人を分けるため

各粒度でデータを移行する場合のテーブルの組合せ

作品（シリーズ）情報：DCTITLE
各話情報　　：DCTITLE＋LISTSBT
各話スタッフ情報　：DCTITLE＋LISTSBT＋LISTU＋DCJOB＋DCWORK＋DCSTD
　　＋LIST0～8＋DC0～8
各話キャスト情報　：DCTITLE＋LISTSBT＋LISTU＋DCJOB＋LIST9＋DC92
※メモ情報は除く

3.7 データ統合

「リスト DB」「EPG」「OCR」について統合するマッピングを行った。しかし、それぞれで人間系で項目を持っている（データがある）と目視で判断できても、適切に項目が分離していなかったり、紐付けに人間の判断が必要だったりする項目が散見され、DB として利用しやすい形に自動的に成形するにはまだ課題が残る。

表 3-2　MADB（β版）への「リスト DB」「OCR」「EPG」のマッピング

特徴			詳細・正確		バランス	早い
採録例	MADB		リストDB		全録OCR	全録EPG
	モデル	項目	ファイル	項目		
ガンダム	作品	シリーズ名	(なし)	(なし)	手作業	手作業
ガンダム	作品	シリーズ名(ヨミ)	(なし)	(なし)	手作業	手作業
機動戦士ガンダム 鉄血のオルフェンズ	作品	タイトル	DCTITLE	TITLE	手作業	手作業
キドウセンシガンダム　テッケツノオルフェンズ	作品	タイトル（ヨミ）	DCTITLE	TITLEREAD	手作業	手作業
長井龍雪	作品	作者名	LIST1-DC1	NAME	手作業	手作業
ナガイ　タツユキ	作品	作者名（ヨミ）	DC1	YOMI	手作業	手作業
MBS	版	発行者名	DCTITLE	STARTTIMEの一部	(なし)	(なし)
2015年10月4日	版	公開年月日	DCTITLE	STARTTIMEの一部	(なし)	(なし)
30	版	放送枠時間	DCTITLE	STARTTIMEの一部	(なし)	(なし)
矢立肇、富野由悠季	版	原作者	LIST8-DC8	NAME	手作業	手作業
機動戦士ガンダム	版	原作名	(なし)	(なし)	手作業	手作業
サンライズ	版	制作名	LIST8-DC8	NAME	手作業	手作業
創通・サンライズ・MBS	版	コピーライト	LIST8-DC8	NAME	手作業	手作業
【アニメイズム】機動戦士ガンダム 鉄血のオルフェンズ　#1 「鉄と血と」	具体形	原タイトル	(なし)	(なし)	○	○
キドウセンシガンダム　テッケツノオルフェンズ　テットチト	具体形	ゲンタイトルヨミ	(なし)	(なし)	(なし)	(なし)
アニメイズム	具体形	枠タイトル	(なし)	(なし)	手作業	○
アニメイズム	具体形	ワクタイトルヨミ	(なし)	(なし)	(なし)	(なし)
機動戦士ガンダム 鉄血のオルフェンズ	具体形	タイトル	DCTITLE	TITLE	○	○
キドウセンシガンダム　テッケツノオルフェンズ	具体形	タイトル（ヨミ）	DCTITLE	TITLEREAD	(なし)	(なし)
1	具体形	順序	LISTSBT	NUMBER	○	記載があれば
鉄と血と	具体形	サブタイトル	LISTSBT－MEMO	MEMOの結合	○	記載があれば
テットチト	具体形	サブタイトル（ヨミ）	(なし)	(なし)	(なし)	(なし)
TBS	具体形	発行者名	DCTITLE	STARTTIMEの一部	○	○
2015年10月4日	具体形	公開年月日	LISTSBT	YEARとDATEの結合	○	○
17:00	具体形	放送開始時刻	DCTITLE	STARTTIMEの一部	○	○
30	具体形	放送枠時間	DCTITLE	STARTTIMEの一部	(なし)	(なし)
企画:サンライズ、原作:矢立肇、富野由悠季、監督:長井龍雪、シリーズ構成:岡田麿里、キャラクターデザイン原案:伊藤悠、キャラクターデザイン:千葉道徳、メカデザイン:鷲尾直広、海老川兼武、形部一平、寺岡賢司、篠原保、美術:草薙、音楽:横山克、制作協力:創通・ADK、製作:サンライズ・MBS等	具体形	スタッフ名	LIST0～8－DC0～8	NAME	○	記載があれば
三日月・オーガス:河西健吾、オルガ・イツカ:細谷佳正、ビスケット・グリフォン:花江夏樹、ユージン・セブンスターク:梅原裕一郎、昭弘・アルトランド:内匠靖明、ノルバ・シノ:村田太志、タカキ・ウノ:天崎滉平、ライド・マッス:田村睦心、ヤマギ・ギルマトン:斉藤壮馬、クーデリア・藍那・バーンスタイン:寺崎裕香、アトラ・ミクスタ:金元寿子、マクギリス・ファリド:櫻井孝宏、ガエリオ・ボードウィン:松風雅也　等	具体形	キャスト名	LIST9－DC9-DC92	ACTNAMEとNAMEの結合	○	記載があれば
OP:「Raise your flag」MAN WITH A MISSION、ED:「オルフェンズの涙」MISIA	具体形	楽曲	LIST7－DC7	NAMEの一部	手作業	記載があれば
	具体形	各話情報備考	(なし)	(なし)	(なし)	(なし)

※表示例のサンプルのため実データと異なります

付録

1　1秒キャプチャ　ソースコード

```
'*********************************************************
'  Function: btnExecCapture_Click
'  Caution：映像ファイル群からのキャプチャ画像の切り出し
'  arg      ：なし
'  result   ：なし
'*********************************************************
Private Sub btnExecCapture_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles btnExecCapture.Click

        Dim strInputMovieFolderName As String = ""          '映像データ格納フォルダ名
        Dim strOutputCaptureFolderName As String = ""       'キャプチャ画像出力先フォルダ名(全体)
        Dim strMovieFileName As String = ""                 '映像ファイル名(ファイル名のみ)
        Dim strMoviesFileName() As String                   '個々の映像ファイル名(フルパス)
        Dim nCountMovies As Integer = 0                     '個々の映像ファイルの個数
        Dim strCaptuerFileName() As String                  '(あれば)出力済のファイル名　※存在チェックのみ
        Dim strArgs As String = ""                          'ffmpeg の引数
        Dim nInterval As Integer = 0                        'キャプチャ取得間隔(単位:秒)
        Dim strMessage As String = ""                       '終了メッセージ
        Dim strExt As String = ""                           '作業対象映像形式(.MP4 or .TS)
        Dim datTotalStartTime As Date                       '処理全体の開始時刻
        Dim datTotalEndTime As Date                         '処理全体の終了時刻
        Dim datLapStartTime As Date                         '個別映像ファイルの開始時刻
        Dim datLapalEndTime As Date                         '個別映像ファイルの終了時刻
        Dim strOutputExt As String                          '出力画像形式(.JPG or .PNG)
        Dim psInfo As New ProcessStartInfo()
        Dim strLog As String

        'ログリストのクリア
        LstLog.Items.Clear()
        'プログレスバーの初期化
        ProgressBar.Value = 0
```

```
        'インターバル間隔の設定
        If opt1.Checked = True Then
            nInterval = 1
        ElseIf opt2.Checked = True Then
            nInterval = 2
        End If

        '対象映像形式の選択
        If optMp4.Checked = True Then
            strExt = "MP4"
        ElseIf optTs.Checked = True Then
            strExt = "TS"
        ElseIf optVOB.Checked = True Then
            strExt = "VOB"
        End If

        '出力画像形式の選択
        If optJpg.Checked = True Then
            strOutputExt = "JPG"
        ElseIf optPng.Checked = True Then
            strOutputExt = "PNG"
        End If

        '読み込み映像フォルダ名の取得
        strInputMovieFolderName = txtMovieFolder.Text

        '出力先フォルダ名の設定()
        strOutputCaptureFolderName = My.Computer.FileSystem.CurrentDirectory & "¥" &
txtCaptureFolder.Text

        '出力フォルダが存在しない場合は作成する
        If Not (IO.Directory.Exists(strOutputCaptureFolderName)) Then
            IO.Directory.CreateDirectory(strOutputCaptureFolderName)
        End If
```

```
        '映像ファイルリストの取得
        strMoviesFileName = System.IO.Directory.GetFiles(strInputMovieFolderName, "*." &
strExt, System.IO.SearchOption.AllDirectories)

        'ファイルリストの件数の取得
        nCountMovies = UBound(strMoviesFileName) + 1

        '処理全体の開始時刻のセット
        datTotalStartTime = Now()

        '画面ログへ処理開始時刻の出力
        strMessage = "処理開始：" & datTotalStartTime
        LstLog.Items.Add(strMessage)
        LstLog.Items.Add("---")

        'ループ処理
        For i = 0 To nCountMovies - 1

            '個別映像ファイルの開始時刻のセット
            datLapStartTime = Now()

            '映像ファイル名のみの取得
            strMovieFileName = System.IO.Path.GetFileName(strMoviesFileName(i))

            '出力フォルダ(個別映像ファイル用)が存在しない場合は作成する
            If Not (IO.Directory.Exists(strOutputCaptureFolderName & "¥" &
strMovieFileName)) Then
                IO.Directory.CreateDirectory(strOutputCaptureFolderName & "¥" &
strMovieFileName)
            End If

            'フォルダ内にファイルが存在するか確認し、存在しない場合にファイルを切り出す
            strCaptuerFileName = System.IO.Directory.GetFiles(strOutputCaptureFolderName
& "¥" & strMovieFileName, "*", System.IO.SearchOption.AllDirectories)
            If UBound(strCaptuerFileName) < 1 Then
```

```
                '画面ログへの出力(開始)
                LstLog.Items.Add(Now() & " " & strMovieFileName & "[キャプチャ出力開始]")

                'ffmpeg への引数設定
                strArgs = ""
                strArgs = strArgs & " -i "                              '
                strArgs = strArgs & """" & strMoviesFileName(i) & """"    ' 映像ファイル名(フルパス)
                strArgs = strArgs & " -r " & 1 / nInterval & " "                    'キャプチャ間隔
                strArgs = strArgs & """" & strOutputCaptureFolderName & "¥" & strMovieFileName & "¥%05d." & strOutputExt & """"  '出力フォルダ＋ファイル名＋拡張子

                'キャプチャ出力[本体]
                psInfo.FileName = "ffmpeg.exe"          ' 実行するファイル(ffmpeg)
                psInfo.Arguments = strArgs              ' 引数
                If strVisibleFfmpeg = "ON" Then
                    txtIndicator.BackColor = Color.Azure      'DOS 窓表示
                    psInfo.CreateNoWindow = False           ' コンソール・ウィンドウを開く
                    psInfo.UseShellExecute = True           ' シェル機能を使用する
                Else
                    txtIndicator.BackColor = Color.Salmon     'DOS 窓非表示
                    psInfo.CreateNoWindow = True            ' コンソール・ウィンドウを開かない
                    psInfo.UseShellExecute = False          ' シェル機能を使用しない
                End If

                Using p As Process = Process.Start(psInfo)
                    Do
                        If p.WaitForExit(500) Then
                            'プロセスが終了したので待機を終了して抜ける
                            txtIndicator.Text = ""
                            Exit Do
                        Else
                            ' 待機がタイムアウトしたので待機を継続する
                            Select Case txtIndicator.Text
                                Case ""
                                    txtIndicator.Text = "－"
```

```
                        Case "／"
                            txtIndicator.Text = "｜"
                        Case "｜"
                            txtIndicator.Text = "＼"
                        Case "＼"
                            txtIndicator.Text = "－"
                        Case "－"
                            txtIndicator.Text = "／"
                    End Select
                End If
            Loop
        End Using

        '個別映像ファイルの終了時刻のセット
        datLapalEndTime = Now()

        '画面ログへの出力(終了)
        LstLog.Items.Add(Now() & " " & strMovieFileName & "[キャプチャ出力終了]")
        LstLog.Items.Add("---")
    Else
        '画面ログへの出力(スキップ)
        LstLog.Items.Add(Now() & " " & strMovieFileName & "[スキップしました]")
        LstLog.Items.Add("---")
    End If

    '画面ログの表示位置更新
    LstLog.TopIndex = LstLog.Items.Count - 1

    'プログレスバーへ値のセット
    ProgressBar.Value = i / nCountMovies * 100

    'フォームの再描画
    Me.Refresh()
Next

'処理全体の終了時刻のセット()
```

```
        datTotalEndTime = Now()

        '画面ログへ作業終了時刻&経過秒数の出力
        strMessage = "処理完了：" & datTotalEndTime & "（所要時間：" &
DateDiff(DateInterval.Second, datTotalStartTime, datTotalEndTime) & " 秒)"
        LstLog.Items.Add(strMessage)
        LstLog.Items.Add("---")

        '画面ログの表示位置更新
        LstLog.TopIndex = LstLog.Items.Count - 1

        'プログレスバーへ値のセット
        ProgressBar.Value = 100

        '画面ログへ作業完了の出力
        If nCountMovies > 0 Then
            strMessage = "■■全映像ファイルのキャプチャ出力(" & nCountMovies & " 件)を終
了しました■■"
        Else
            strMessage = "□□対象ファイルが存在しません□□"
        End If
        LstLog.Items.Add(strMessage)
        LstLog.Items.Add("---")

        '画面ログの表示位置更新
        LstLog.TopIndex = LstLog.Items.Count - 1


        'ファイルログの出力
        strLog = ""
        For i = 0 To LstLog.Items.Count - 1
            strLog = strLog & LstLog.Items(i) & vbCrLf
        Next

        Dim info As Byte() = New UTF8Encoding(True).GetBytes(Now() & strLog)
        Dim fs As FileStream = File.Create(My.Computer.FileSystem.CurrentDirectory & "¥log"
```

```
& Now.ToString("yyyyMMddHHmmss") & ".txt")
            fs.Write(info, 0, info.Length)
            fs.Close()

        End Sub
```

2　OCR　変換スクリプト

```
#@title ③解析を開始します
import os
os.environ["GOOGLE_APPLICATION_CREDENTIALS"] = '/content/setup/'+keyfile # json_path
は、サービスアカウントキーのパス
import glob
import re
import time
import codecs
import cv2
import matplotlib.pyplot as plt
import matplotlib
from enum import Enum
import numpy as np
import io
import json
from google.colab import files
from google.cloud import vision
from google.cloud.vision_v1 import types
import japanize_matplotlib
import seaborn as sns
sns.set(font="IPAexGothic")

%matplotlib inline

files = os.listdir(path)
file = [f for f in files if os.path.isfile(os.path.join(path, f))]
os.chdir(path)

class FeatureType(Enum):
    PAGE = 1
    BLOCK = 2
    PARA = 3
    WORD = 4
```

```
    SYMBOL = 5

def draw_boxes(input_file, bounds):
    img = imread(input_file, cv2.IMREAD_COLOR)
    col = 0
    for bound in bounds:
      if type(bound) is float:
          cf = (bound)
          col = cf *255
          continue
      else:
          green = int(col)
          p1 = (bound.vertices[0].x, bound.vertices[0].y) # top left
          p2 = (bound.vertices[1].x, bound.vertices[1].y) # top right
          p3 = (bound.vertices[2].x, bound.vertices[2].y) # bottom right
          p4 = (bound.vertices[3].x, bound.vertices[3].y) # bottom left
          cv2.line(img, p1, p2, (0, green, 0), thickness=1, lineType=cv2.LINE_AA)
          cv2.line(img, p2, p3, (0, green, 0), thickness=1, lineType=cv2.LINE_AA)
          cv2.line(img, p3, p4, (0, green, 0), thickness=1, lineType=cv2.LINE_AA)
          cv2.line(img, p4, p1, (0, green, 0), thickness=1, lineType=cv2.LINE_AA)
    return img

def get_document_bounds(response, feature):
    document = response.full_text_annotation
    bounds = []
    confid = []
    for page in document.pages:
        for block in page.blocks:
            for paragraph in block.paragraphs:
                for word in paragraph.words:
                    for symbol in word.symbols:
                        if (feature == FeatureType.SYMBOL):
                            bounds.append(symbol.confidence)
                            bounds.append(symbol.bounding_box)

                    if (feature == FeatureType.WORD):
```

```
                        bounds.append(word.bounding_box)
                if (feature == FeatureType.PARA):
                    bounds.append(paragraph.bounding_box)
            if (feature == FeatureType.BLOCK):
                bounds.append(block.bounding_box)
    return bounds,confid

client = vision.ImageAnnotatorClient()

def imread(filename, flags=cv2.IMREAD_COLOR, dtype=np.uint8):
    try:
        n = np.fromfile(filename, dtype)
        img = cv2.imdecode(n, flags)
        return img
    except Exception as e:
        print(e)
        return None

def outfile(input_file, jsonObj, response):
    jsons = jsonObj.encode('utf-8').decode('unicode-escape')
    with codecs.open(os.path.splitext(os.path.basename(input_file))[0]+'.json', 'w', 'utf-8') as f:
        f.write(jsons)
    with codecs.open(os.path.splitext(os.path.basename(input_file))[0]+'.txt', 'w', 'utf-8') as f:
        print( response.full_text_annotation.text, file=f)



def detect_text(input_file):
    client = vision.ImageAnnotatorClient()
    with io.open(input_file,'rb') as image_file:
        content = image_file.read()
    image = types.Image(content=content)
    response  =  client.document_text_detection(image=image,image_context={"language_hints":
["ja"]},   )
    return response

def analyse(input_file):
```

```
    print(input_file,"解析開始")
    #解析用イメージ読み込み
    response = detect_text(input_file)
    bounds,confid = get_document_bounds(response, FeatureType.SYMBOL)
    img_block = draw_boxes(input_file, bounds)
    sns.set(font="IPAexGothic")
    plt.figure(figsize=[25,25])
    plt.grid(False)
    plt.imshow(img_block[:,:,::-1])
    plt.title(input_file[:-4], fontsize = 48)
    plt.savefig(os.path.splitext(os.path.basename(input_file))[0]+'_ocr.png')
    plt.show()
    print("解析結果：")
    print(response.full_text_annotation.text)
    jsonObj = response.__class__.to_json(response)
    outfile(input_file,jsonObj,response)


for input_file in file:

    gl = glob.glob('*.txt') #テキストファイル一覧取得
    pattern = ".*¥.(jpg|png|bmp)"
    files = re.search(pattern, input_file, re.IGNORECASE)
    if files:
        if input_file.endswith('_ocr.png'): #ocrpng ファイルチェック
          continue
        else:
            fb = (os.path.basename(input_file).split('.', 1)[0]+".txt") #同名ファイルの txt 版
            if fb in gl: #同名ファイルの txt 版チェック
                print('json file found')
                time.sleep(4)
                print(input_file,'を再解析しますか？(y/n)')
                input_text=input()
                if input_text == "n":
                    continue
                elif input_text == "y":
```

```
                    analyse(input_file)
                    continue
                else:
                    continue
            else:
                analyse(input_file)
                continue
    else:
        pass
print('処理終了')
```

3　EPG 統合　ソースコード

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;

using Microsoft.WindowsAPICodePack.Dialogs;

using System.IO;

using System.Text.RegularExpressions;//正規表現
using System.Reflection.Emit;
using System.Data.SqlTypes;



namespace epg2csv
{
    /// <summary>
    /// MainWindow.xaml の相互作用ロジック
    /// </summary>
    public partial class MainWindow：Window
    {
        enum AutoType
        {
            all, last, normal
```

```
};
enum DataType
{
    Info, Story, Last
};

const string iniName = "epg2csv.ini";
List<string> iniLine = new List<string>();

AutoType AutoMode = AutoType.normal;
string SetPath = "";
string lastdays = "";

string LastdaysTemp = "";//いま処理してるもの
string NowLastdays = "";//処理した最新

public MainWindow()
{
    InitializeComponent();
    Encoding.RegisterProvider(CodePagesEncodingProvider.Instance);

    //ini ファイルを読み込む(テキストとして処理してます)
    iniRead();
}

private async void iniRead()
{
    try
    {
        FileStream fs = new FileStream(iniName, FileMode.Open);
        StreamReader sr = new StreamReader(fs,
System.Text.Encoding.GetEncoding("shift-jis"));
        string onceline = null;


        while ((onceline = await sr.ReadLineAsync()) != null)
```

```
            {
                iniLine.Add(onceline);
                onceline = onceline.Replace("¥r", "").Replace("¥n", "");

                Match IniMacth = Regex.Match(onceline, "(.*)=(.*)");
                if (IniMacth.Success)
                {
                    Console.WriteLine(IniMacth.Groups[1].ToString());
                    Console.WriteLine(IniMacth.Groups[2].ToString());


                    if (IniMacth.Groups[1].ToString() == "mode" &&
IniMacth.Groups[2].ToString() == "all")
                        AutoMode = AutoType.all;
                    else if (IniMacth.Groups[1].ToString() == "mode" &&
IniMacth.Groups[2].ToString() == "last")
                        AutoMode = AutoType.last;

                    if (IniMacth.Groups[1].ToString() == "path")
                        SetPath = IniMacth.Groups[2].ToString();

                    if (IniMacth.Groups[1].ToString() == "lastdays")
                        lastdays = IniMacth.Groups[2].ToString();

                }

            }

            if (AutoMode != AutoType.normal)
            {
                ReadProcess(SetPath);
            }



        }
```

```
        catch (Exception ex)
        {
            await iniWriteAsync();//ファイルがなさそうなら書いておく
        }
    }


    private async Task iniWriteAsync()
    {

        string ModeText = "normal";
        if (AutoMode == AutoType.all)
            ModeText = "all";
        else if (AutoMode == AutoType.last)
            ModeText = "last";

        //新しい日付いれる
        //lastdays = DateTime.Now.ToString("yyyyMMdd");
        lastdays = NowLastdays;

        string IniText = Resource.DefIni.Replace("{{mode}}", ModeText).Replace("{{path}}",
SetPath).Replace("{{lastdays}}", lastdays);


        Encoding sjisEnc = Encoding.GetEncoding("Shift_JIS");
        byte[] encodedText = sjisEnc.GetBytes(IniText);


        using (FileStream sourceStream = new FileStream(iniName,
            FileMode.OpenOrCreate, FileAccess.Write, FileShare.None,
            bufferSize: 4096, useAsync: true))
        {
            await sourceStream.WriteAsync(encodedText, 0, encodedText.Length);
        };
    }
```

```
        private void dlgOpen(object sender, RoutedEventArgs e)
        {
            var dlg = new CommonOpenFileDialog("EPG データ(.txt)フォルダ選択");
            // フォルダ選択モード。
            dlg.IsFolderPicker = true;
            var ret = dlg.ShowDialog();
            if (ret == CommonFileDialogResult.Ok)
            {
                ReadProcess(dlg.FileName);
            }

        }

        private async void ReadProcess(string filePath)
        {

            SelBtn.Visibility = Visibility.Hidden;


            //総数取得
            int MaxCount = 0;
            try
            {
                MaxCount = Directory.GetFiles(filePath, "*.txt").Length;
            }
            catch (Exception ex)
            {
                StatusText.Text = "ファイルパスエラー(自動処理時は ini ファイルを確認してくだ
さい)";
                return;
            }
            var Count = 0;


            //csv データ貯める
```

```
StringBuilder CsvData = new StringBuilder();
CsvData.AppendLine("放送日,放送時刻(From),放送時刻(To),放送局,原タイトル,枠タイ,作品名,話数表記,サブタイトル,あらすじ,詳細情報,ジャンル(1)大,ジャンル(1)小,ジャンル(2)大,ジャンル(2)小,ジャンル(3)大,ジャンル(3)小,画面解像度,音声,サンプリングレート,OriginalNetworkID,TransportStreamID,ServiceID,EventID");
int WriteDataCount = 0;


//名前取得
IEnumerable<string> files = Directory.EnumerateFiles(filePath, "*.txt", System.IO.SearchOption.AllDirectories);
//ファイルを列挙する
foreach (string f in files)
{
    Count++;
    StatusText.Text = "処理中¥n" + Count + "/" + MaxCount;

    try
    {
        FileStream fs = new FileStream(f, FileMode.Open, FileAccess.Read);
        StreamReader sr = new StreamReader(fs, System.Text.Encoding.GetEncoding("shift-jis"));
        string onceline = null;
        int Nowline = 0;

        string LineDate = "";//放送日時 1
        string LineBroadcaster = "";//放送局 2
        string LineTitle = "";//番組名 3
        string LineSubTitle = "";//5 サブタイトル(ない時も)

        StringBuilder Story = new StringBuilder();//詳細内　あらすじ以後
        StringBuilder Info = new StringBuilder();//詳細内　その他
        StringBuilder LastData = new StringBuilder();//詳細内　ジャンル行以後

        DataType NowType = DataType.Info;//いまどのデータか
```

```
while ((onceline = await sr.ReadLineAsync()) != null)
{
    Nowline++;

    onceline = onceline.Replace(",", ",  ").Replace("¥r", "").Replace("¥n", "");//
カンマを全角に(csv の都合

    switch (Nowline)
    {
        case 1:
            LineDate = onceline;
            break;
        case 2:
            LineBroadcaster = onceline;
            break;
        case 3:
            LineTitle = onceline;
            break;
        case 4:
            //空白行
            break;
        case 5:
            LineSubTitle = onceline;
            break;
        case 6:
            //空白行
            break;
        case 7:
            //詳細情報とかかれた行
            break;
        default:

            //空白行とばす
            if (onceline == "")
            {
                break;
```

```
                }

                if (Regex.IsMatch(onceline, "^あらすじ"))
                {
                    NowType = DataType.Story;
                    break;
                }
                if (Regex.IsMatch(onceline, "^ジャンル"))
                {
                    NowType = DataType.Last;
                    break;
                }

                if (NowType == DataType.Story)
                {
                    Story.AppendLine(onceline);
                    NowType = DataType.Info;
                    //現状ではあらすじ一行のみ、すぐに詳細にもどす
                    break;
                }
                if (NowType == DataType.Info)
                {
                    Info.AppendLine(onceline);
                    break;
                }
                if (NowType == DataType.Last)
                {
                    LastData.AppendLine(onceline);
                    break;
                }
                break;
            }
        }

        //タイトルをできるだけ分ける
        String Waku = "";
```

```
String Wasuu = "";
String SubTitle = "";

String Title = LineTitle;//ここから話数などをけしていく LineTitle は一応そのままに。

//[新][字][終][再][デ][二][多]削除
Title = Regex.Replace(Title, Regex.Escape("[") + "[新字終再デ二多]" + Regex.Escape("]"), "");

//アニメ(全角スペース)ミニアニメ(全角スペースを削除
Title = Regex.Replace(Title, "(ミニアニメ|アニメ)　", "");


//枠(既知のものだけ)
MatchCollection WakuMacths = Regex.Matches(Title, "[【＜（](フジバラナイト　ＳＡＴ|アニメイズム|スーパーアニメイズム|アニおび|ノイタミナ|＋Ｕｌｔｒａ|天てれアニメ)[】＞）]");

foreach (Match WakuMacth in WakuMacths)
{
    if (WakuMacth.Success)
    {
        if (Waku != "")
            Waku += "　";
        Waku += WakuMacth.Groups[1].ToString();
        Title = Title.Replace(WakuMacth.Value, "");
    }
}
//ほかの枠
Match WakuMacthEx = Regex.Match(Title, "(Ａｎｉｃｈ U)");
if (WakuMacthEx.Success)
{
    if (Waku != "")
        Waku += "　";
    Waku += WakuMacthEx.Groups[1].ToString();
```

```
                Title = Title.Replace(WakuMacthEx.Value, "");
            }

            //話数(特殊表記には対応していません)
            String Suuji = "０-９0-9 一-十．零壱弐参拾";
            String[] WasuuKeys = { "総集編", "[＃#][" + Suuji + "]+", "第*[" + Suuji + "]+
話", "（[" + Suuji + "]+）" };
            foreach (String SearchKey in WasuuKeys)
            {
                Match RangeWasuuMacth = Regex.Match(Title, SearchKey + "[～・]" +
SearchKey);

                if (RangeWasuuMacth.Success)
                {
                    Wasuu = RangeWasuuMacth.Value;
                    Title = Title.Replace(RangeWasuuMacth.Value, "");
                    break;
                }
                else
                {
                    MatchCollection WasuuMacthes = Regex.Matches(Title,
SearchKey);

                    StringBuilder WauuTemp = new StringBuilder();
                    foreach (Match OnceMacth in WasuuMacthes)
                    {
                        WauuTemp.AppendLine(OnceMacth.Value);
                        Title = Title.Replace(OnceMacth.Value, "");
                    }
                    if (WasuuMacthes.Count > 0)
                    {
                        Wasuu = WauuTemp.ToString().Replace("¥r¥n", " ");
                        break;
                    }
                }
            }
```

```
//空のカッコ消す
Title = Regex.Replace(Title, "[「【〖＜][」】』＞]", "");


List<String> SubTitleList = new List<String>();
while (true)
{
    //文末のごみをけす
    Title = Regex.Replace(Title, "[　　★、・]+$", "");

    //一般名称テレビアニメ、アニメなどをいったんけして
    String TitleTemp = Regex.Replace(Title, "(テレビアニメ|アニメ)", "");

    //「」だけになるようだったら処理をやめる
    if (Regex.Match(TitleTemp, "^「[^「」]*」$").Success)
    {
        break;
    }

    //サブタイトル
    Match SubMacth = Regex.Match(Title, "「([^「」]*)」$");
    if (SubMacth.Success)
    {
        SubTitleList.Insert(0, SubMacth.Groups[1].ToString());//後から検索しているので、ひっかかったらどんどん前に追加していく
        Title = Title.Replace(SubMacth.Value, "");
    }
    else
        break;

}
SubTitle = string.Join(" ", SubTitleList.ToArray());
```

```
//日付を分ける
Match DateMacth = Regex.Match(LineDate, "(.*) (.*):(.*)～(.*):(.*)");

//時間を 30 時間表記に
int iFromHour = int.Parse(DateMacth.Groups[2].ToString());
string sFromMin = DateMacth.Groups[3].ToString();
int iEndHour = int.Parse(DateMacth.Groups[4].ToString());
string sEndMin = DateMacth.Groups[5].ToString();

bool ViewMod = false;//開始時刻に 24 たしたかどうか

if (iFromHour < 4)
{
    iFromHour += 24;
    ViewMod = true;
}
if (iFromHour < 4 || iFromHour > 6)//開始時刻が 4:00 より前ももしくは 6時より後
{
    if (iEndHour < 6)//00: 00～06:00 なら
        iEndHour += 24;
}


string FromTime = iFromHour.ToString("D2") + ":" + sFromMin;
string EndTime = iEndHour.ToString("D2") + ":" + sEndMin;

//・放送開始時刻が 00: 00～04:00 の場合は 24(時間)を加算、
//・放送開始時刻が 04: 00～06:00 の場合は触らず、
//・放送開始時刻が 4 時以前、かつ、放送終了時刻が 00: 00～06:00 の場合は24(時間)を加算、
//・放送開始時刻が 4 時以降、かつ、放送終了時刻が 00: 00～06:00 の場合は触らず、
//・開始時刻が 4: 00～6:00 で終了時刻が 00: 00～06:00 の場合は触らない
//・それ以外の終了時刻が 00: 00～06:00 の場合は 24 時間追加
```

```
                    //なかで LastdaysTemp がセットされるのではじくかどうかきめる
                    string DateText = DateModifiy(DateMacth.Groups[1].ToString(), ViewMod);

                    //AutoType.last で lastdays が空じゃなくて、LastdaysTemp が lastdays より小さい=前だったら飛ばす
                    if (AutoMode == AutoType.last && lastdays != "" && int.Parse(LastdaysTemp) <= int.Parse(lastdays))
                        continue;



                    //LastData を分ける(RegexOptions.Singleline で一行としてあつかう)
                    Match LastDataMacth = Regex.Match(LastData.ToString(), "(.*)映像 : (.*)音 声 : (.*) サ ン プ リ ン グ レ ー ト : (.*)OriginalNetworkID:(.*)TransportStreamID:(.*)ServiceID:(.*)EventID:(.*)", RegexOptions.Singleline);

                    //ジャンルを分ける
                    string[] SplitKey = { "¥r¥n" };//文字列で切る場合は第一引数を string 配列にして、オプション付ける必要あり
                    string[] GenreLine = LastDataMacth.Groups[1].ToString().Split(SplitKey, StringSplitOptions.RemoveEmptyEntries);

                    string[,] Genres = new string[3, 2];
                    int NowGenresCnt = 0;

                    foreach (string OnceLine in GenreLine)
                    {
                        //スペース削除
                        string[] OnceLineTemp = Regex.Replace(OnceLine, "[　]", "").Split('-');

                        Genres[NowGenresCnt, 0] = OnceLineTemp[0];
                        if (OnceLineTemp.Length >= 2)
                            Genres[NowGenresCnt, 1] = OnceLineTemp[1];
                        NowGenresCnt++;
```

```
                }


                //放送日,放送時刻(From),放送時刻(To),放送局,原タイトル,枠タイ,作品名,話数表記,サブタイトル,あらすじ,
                //詳細情報,ジャンル(1)大,ジャンル(1)小,ジャンル(2)大,ジャンル(2)小,ジャンル(3)大,ジャンル(3)小,
                //画面解像度,音声,サンプリングレート,OriginalNetworkID,TransportStreamID,ServiceID,EventID");


                StringBuilder CsvFormat = new StringBuilder();
                CsvFormat.AppendLine( DateText );//放送日
                CsvFormat.AppendLine(FromTime);//放送時刻 From
                CsvFormat.AppendLine(EndTime);//放送時刻 To
                CsvFormat.AppendLine(LineBroadcaster);//放送局
                CsvFormat.AppendLine(LineTitle);//原タイトル
                CsvFormat.AppendLine(Waku.Trim());//枠
                CsvFormat.AppendLine(Title.Trim());//番組タイトル
                CsvFormat.AppendLine(Wasuu.Trim());//話数
                CsvFormat.AppendLine(SubTitle.Trim());//サブタイトル

                //LineSubTitle 詳細に入れた

                CsvFormat.AppendLine(Story.ToString().Replace("¥r¥n",  "").Trim());//あらすじ      ※ない場合もある
                CsvFormat.AppendLine((LineSubTitle + " " + Info.ToString().Replace("¥r¥n", " ")).Trim());//詳細情報       ※ない場合もある


//CsvFormat.AppendLine(LastDataMacth.Groups[1].ToString().Replace("¥r¥n",  " "));//ジャンル最小１つ、最大３つまで設定あり
                CsvFormat.AppendLine(Genres[0, 0]);
                CsvFormat.AppendLine(Genres[0, 1]);
                CsvFormat.AppendLine(Genres[1, 0]);
                CsvFormat.AppendLine(Genres[1, 1]);
```

```
                    CsvFormat.AppendLine(Genres[2, 0]);
                    CsvFormat.AppendLine(Genres[2, 1]);


CsvFormat.AppendLine(LastDataMacth.Groups[2].ToString().Replace("¥r¥n", ""));//映像情報

CsvFormat.AppendLine(LastDataMacth.Groups[3].ToString().Replace("¥r¥n", ""));//音声情報

CsvFormat.AppendLine(LastDataMacth.Groups[4].ToString().Replace("¥r¥n", ""));//サンプリング
レート

CsvFormat.AppendLine(LastDataMacth.Groups[5].ToString().Replace("¥r¥n",
""));//OriginalNetworkID

CsvFormat.AppendLine(LastDataMacth.Groups[6].ToString().Replace("¥r¥n",
""));//TransportStreamID

CsvFormat.AppendLine(LastDataMacth.Groups[7].ToString().Replace("¥r¥n", ""));//ServiceID

CsvFormat.AppendLine(LastDataMacth.Groups[8].ToString().Replace("¥r¥n", ""));//EventID

                    CsvData.AppendLine(CsvFormat.ToString().Replace("¥r¥n",
","));//windows で AppendLine なので¥r¥n のみで平気
                    WriteDataCount++;//書いたデータの分だけカウント

                    sr.Close();
                    fs.Close();


                }
                catch (Exception ex)
                {
                }


            }
```

```
            if (WriteDataCount > 0)
            {
                if(await WriteTextAsync(".", CsvData))//exe と同じフォルダに書くため filePath
を"."に
                    StatusText.Text = WriteDataCount + "件の処理が完了しました";
                else
                    StatusText.Text = "書き込み時エラーが発生しました";
            }
            else
                StatusText.Text = "処理するデータがありませんでした";

            SelBtn.Visibility = Visibility.Visible;

            if (AutoMode == AutoType.last || AutoMode == AutoType.all)
                Close();//自動終了


        }

        string DateModifiy(string DateString, bool ViewMod)
        {

            Match DateMacthEx = Regex.Match(DateString,
"(.*)/(.*)/(.*)"+Regex.Escape("(")+"(.*)"+ Regex.Escape(")"));
            if (DateMacthEx.Success)
            {
                LastdaysTemp = DateMacthEx.Groups[1].Value +
DateMacthEx.Groups[2].Value + DateMacthEx.Groups[3].Value;

                if (NowLastdays == "" || int.Parse(LastdaysTemp) > int.Parse(NowLastdays))
                    NowLastdays = LastdaysTemp;

                if (!ViewMod)
                    return DateString;
```

```
                DateTime NowDate = new DateTime(int.Parse(DateMacthEx.Groups[1].Value),
int.Parse(DateMacthEx.Groups[2].Value), int.Parse(DateMacthEx.Groups[3].Value));
                DateTime BeforeDate = NowDate.AddDays(-1);
                return BeforeDate.ToString("yyyy/MM/dd(ddd)");
            }
            else
                return DateString;//万が一分解できなかったときはそのままかえす

        }


        private async Task<bool> WriteTextAsync(string filePath, StringBuilder srcData)
        {
            Encoding sjisEnc = Encoding.GetEncoding("Shift_JIS");
            byte[] encodedText = sjisEnc.GetBytes(srcData.ToString());

            filePath += @"¥epgdata"+ DateTime.Now.ToString("yyyyMMddHHmmss") + ".csv";

            try
            {
                using (FileStream sourceStream = new FileStream(filePath,
                    FileMode.Append, FileAccess.Write, FileShare.None,
                    bufferSize: 4096, useAsync: true))
                {
                    await sourceStream.WriteAsync(encodedText, 0, encodedText.Length);
                };
            }
            catch (Exception ex)
            {
                return false;
            }

            //設定ファイルもかく
            await iniWriteAsync();
            return true;
```

```
            }
        }
    }
```