

Copyright © 2000 – 2015, George Williams and the FontForge Project contributors.
Shared freely under the project license.
日本語版: 旧オンライン日本語版を元に作成 (2024)

関連文書・総目次 Documentation

※ 黒字部分が本分冊版の収録範囲です。

About FontForge	FontForge プロジェクトについて
Design With FontForge	FontForge 入門：FontForge をはじめるためのオンライン・ブック
Overview	FontForge の概要
Tutorial	チュートリアル
User Interface	FontForge のユーザー・インターフェース・ガイド
Scripting FontForge	FontForge 用スクリプト（定形処理プログラム）
Technical References	FontForge 開発者のための技術関連情報
Utilities	FontForge を補完するユーティリティ・プログラム（フォント検証ユーティリティ）
FAQ	FontForge よくある質問
Appendices	【付録】FontForge に関連する情報
Typographical glossary	フォントとタイポグラフィに関する用語集

この文書のライセンスは、
英語原文版および旧日本語翻訳版のライセンスに
準拠するものとします。

改版履歴

版数	発行日	内容
1.00	2024/03/15	FontForge Documentation 日本語アップデート版 • 旧日本語版を元に、英語版 https://fontforge.org/docs/ 内容に合わせて改訂、分冊化（Book 2）pdf 版「チュートリアル」
0.00		旧日本語版（底本） [Japanese documentation (outdated)] https://fontforge.org/docs/old/ja/overview.html#intro



FontForge チュートリアル

Tutorial

フォント作成のための手引

2024 版

※ 文中の画像は、必要に応じて 2023.01 版のものに差し替えまたは追加しています。

チュートリアル 目次

1. フォントとグリフの作成.....	7
1.1. フォントの作成.....	7
1.2. グリフの作成法（アウトラインのトレース法）	8
2. スピロ曲線でグリフを作成する.....	14
2.1. 手順 1	14
2.2. 手順 2	15
2.3. 手順 3	16
2.4. 手順 4	16
2.5. 手順 5	17
2.6. 手順 6	17
2.7. 手順 7	18
2.8. 手順 8	18
2.9. 手順 9	19
2.10. 手順 10	19
2.11. 手順 11	20
3. アウトライングリフの取り込み.....	22
4. グリフについてさらに詳しく	25
4.1. グリフの呼び出し方.....	25
4.2. 文字「o」の作成——矛盾の無い輪郭線の向き	25
5. 整合したステム幅、セリフと高さをもつ文字の作成法.....	28

6. アクセント付グリフ、合字、検索、機能.....	32
6.1. アクセント記号付きグリフの組み立て.....	32
6.2. 合字の作成.....	33
6.3. 検索と機能.....	36
7. メトリック、間隔、カーニング.....	39
7.1. メトリックの検査と制御.....	39
7.1.1. 縦書き用メトリック.....	41
7.1.2. フォントのベースライン間の間隔の設定方法.....	41
7.2. カーニング.....	42
7.2.1. カーニングクラス.....	44
7.2.2. 縦書きカーニング.....	44
8. 異体字とアンカーマーク.....	45
8.1. 異体字グリフ.....	45
8.1.1. 条件つき異体字.....	46
8.2. アンカーマーク.....	47
9. 条件つき機能.....	49
9.1. OpenType の例.....	49
9.2. Apple 高度組版機能.....	53
9.3. オープンタイプ、ギリシア文字の合字.....	55
手順.....	58
10. フォントの検証と生成法.....	68
10.1. フォントの検証.....	68
10.2. ビットマップ.....	68
10.3. フォントの生成.....	69
10.4. フォントファミリー.....	69
10.5. 最後のまとめ.....	71
11. ビットマップ・フォントを取り扱う.....	72
11.1. ビットマップ・ストライクの取り込み.....	72
11.2. ビットマップ・ストライクの作成.....	73
11.3. ビットマップの属性.....	74
11.4. 新しいビットマップ専用フォントの作成.....	74
11.5. フォントビューのビットマップ.....	74
11.6. ビットマップ・フォントの編集.....	75
11.7. 最小限のビットマップ・ストライク.....	75
12. FontForge スクリプト チュートリアル.....	77
12.1. 簡単な事例.....	77
12.1.1. とりあえずの解決法.....	77
12.1.2. 実世界で考慮すべき事柄.....	78
12.1.3. スクリプトの起動法と引数の指定方法.....	78
12.1.4. ループの使い方.....	78
12.1.5. より複雑な事例.....	79
12.2. その他の事例.....	82
12.2.1. Type1 フォントにアクセントつき文字を追加する.....	82
12.2.2. Type1 フォントと Type1 エキスパートフォントを併合し、適切な GSUB テーブルを	

作成する.....	82
12.2.3. より多くの事例.....	82
13. 特定の用字系において特に考慮すべき点.....	83
13.1. 特定の用字系において特に考慮すべき点.....	83
13.1.1. 共用文字.....	83
13.1.2. ラテン文字.....	84
13.1.3. ギリシア文字.....	84
13.1.4. キリル文字.....	85
13.1.5. アラビア文字.....	85
13.1.6. ヘブライ文字.....	85
13.1.7. インド系の諸文字.....	85
13.1.8. 韓国ハングル文字.....	86
13.1.9. 日本語と中国語（と韓国語の漢字）.....	86

1. フォントとグリフの作成

Font and glyph creation

このチュートリアルに進めば、必要な基本知識が得られます。

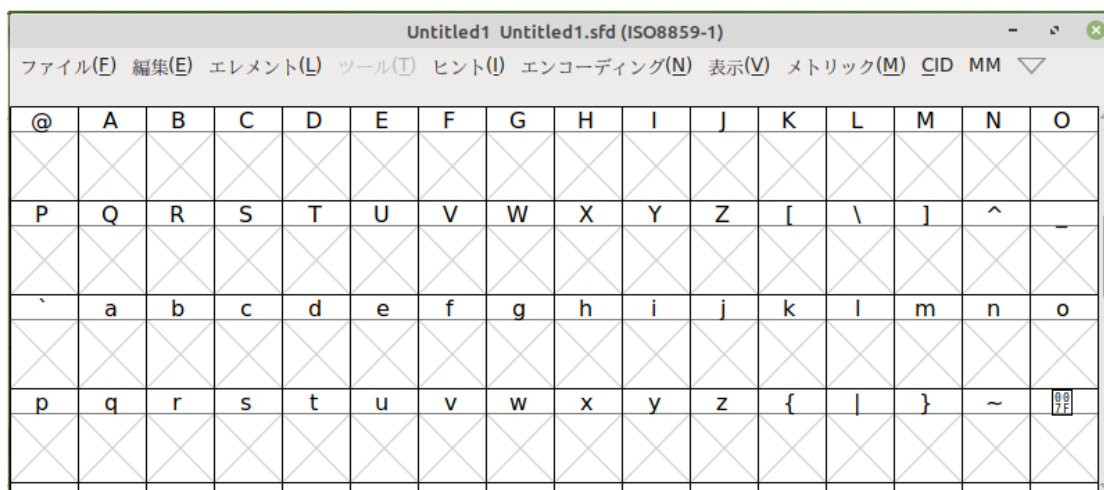
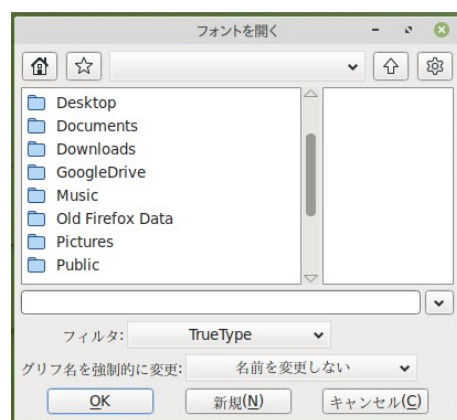
私にフォントの美学を教えるつもりはありません。ここでは機能のみに集中します。

貴族: これだよ、職人わざというのは。美しい書物以上に精緻をきわめたものはまたとあるまい。美しい縁どりに収められた漆黒の文字の見事な配列、巧みに挿入された色刷りの絵。ところがこの頃の人間は、書物を眺めようとせず、読んでばかりいる。これでは書物も、いま君が書きなぐっているベーコンや魚の注文伝票と、なんの変わりもありはしない。

— 『聖女ジョウン』 第四場
ジョージ・バーナード・ショー, 1924 年
(中川龍一・小田島雄志訳)

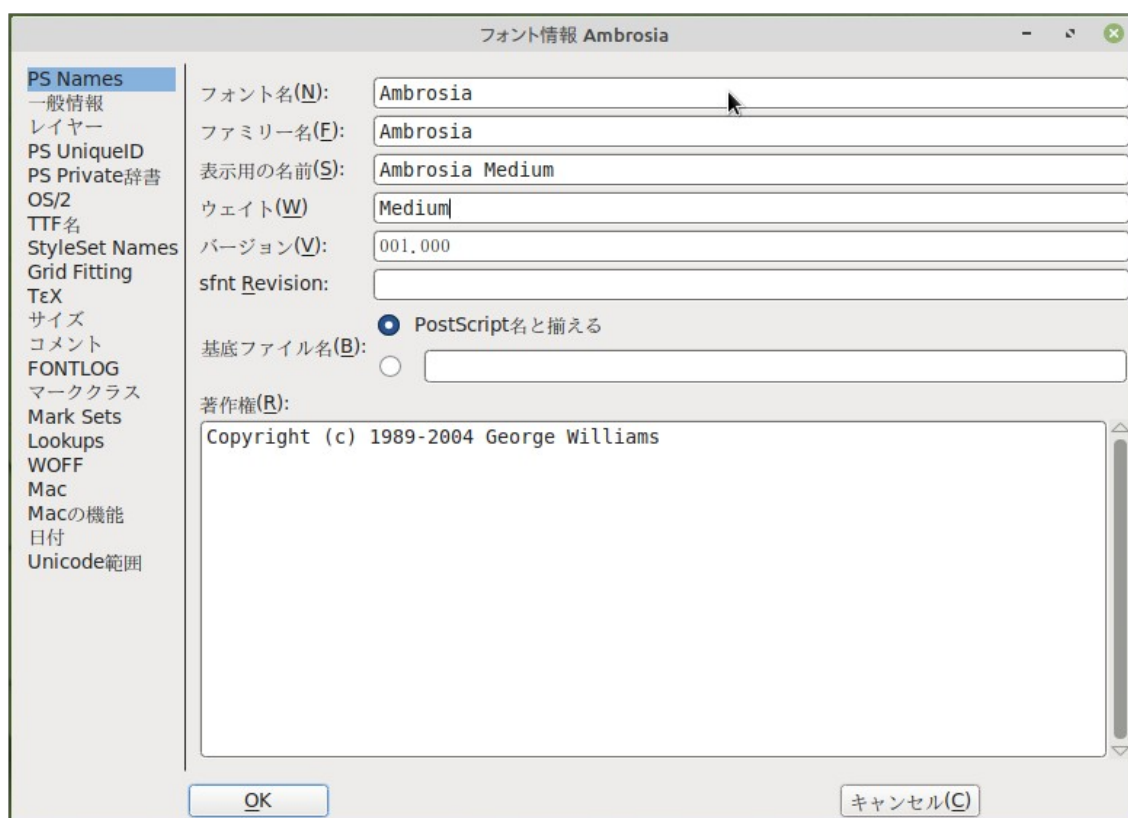
1.1. フォントの作成

新しいフォントを作成するには、まず **【ファイル(F)】** メニューの **【新規(N)】** コマンドで新規の「フォントビュー」 [=フォント一覧ウィンドウ、下図] を表示させます (または、起動ウィンドウ [右図] の **【新規(N)】** ボタンをクリックします)。



フォントには【**エレメント(L)**】メニューの【**フォント情報(F)...**】コマンドで名前をつけてください。このコマンドは他にも、著作権表示を設定したり「**高さ (ascent)**」「**深さ (descent)**」を設定するのに使えます。高さと深さ、このふたつの値の合計がフォントの「**全角正方形／EM の大きさ (em square)**」になります。一般には PostScript フォントでは「1000」、TrueType フォントでは「2 の冪乗」（多くは「2048」か「4096」）、Ikarus フォントでは「15000」です。

（もし TrueType フォントを作る予定ならば、【**レイヤー**】タブの「**すべて 2 次 (Q)**」「**二次スプライン曲線**」チェックボックスをオンにして、TrueType フォーマットそのもので編集した方がいいかもしれません。ただし、このモードでは編集が少し難しくなります。）

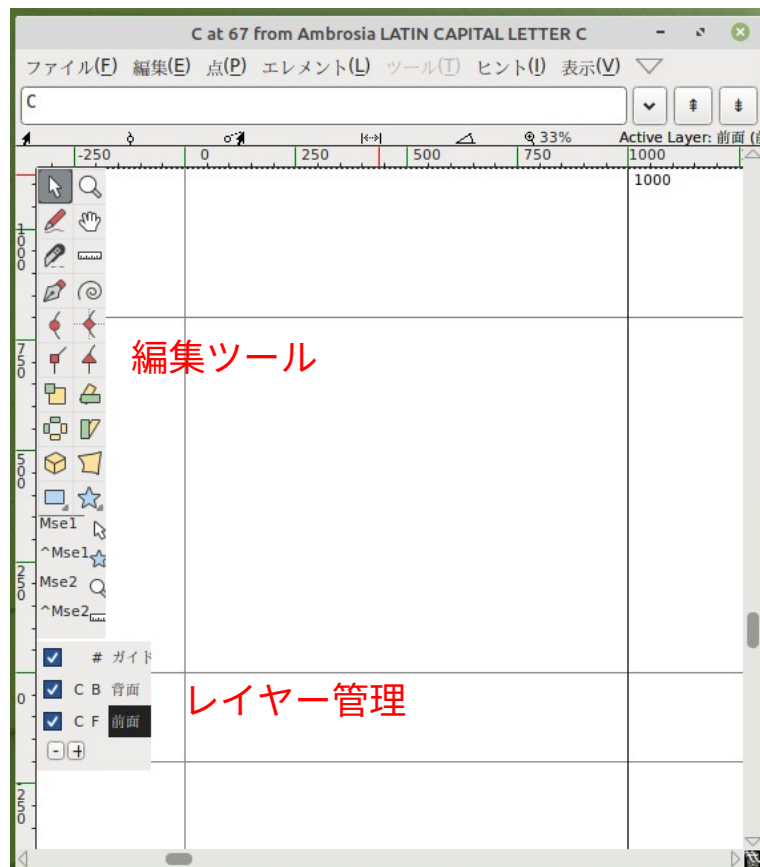


フォント内でどの文字が利用できるかを変更したい場合は、【**エンコーディング (N)**】⇒【**エンコーディング変換(R)**】を使用することができます。特に設定していなければ、FontForge は新しいフォントを **ISO-8859-1** で作成します。このエンコーディングには、西ヨーロッパで必要とされる文字——ラテン文字、いくつかのアクセント付き文字、数字および記号——（のほとんど）が含まれています。

1.2. グリフの作成法（アウトラインのトレース法）

準備が終わったら、文字の編集を始める用意ができています。上図の「フォントビュー」の「C」の項目をダブルクリック※して下さい。空の「アウトラインウィンドウ」〔グリフ作成画面〕が表示されます。

※ ダブルクリックで「アウトラインウィンドウ」が開かない場合は、右クリックで【アウトラインウィンドウを開く(U)】を選択します。



このアウトライングリフウィンドウの左端には、ふたつのパレットが埋め込まれています。上側のパレットには「編集ツール」の一覧を含み、下側のパレットは編集レイヤーの管理〔ウィンドウのどのレイヤーが見え、どれが編集可能か〕をしています。

「三次曲線レイヤー」（C 表示¹）では、PS フォント²のように、三次ベジェ・スプライン曲線を使用しています。「二次曲線レイヤー」（Q 表示³）は二次スプライン曲線を用いていますが、これは編集が難しい代わりに、数学的には制御が簡単で単純なので、TTF フォント⁴に使用されています。

「前面レイヤー」（F 表示⁵）はフォントの部品となるアウトライン〔輪郭線〕を含んでいます。「背面レイヤー」（B 表示⁶）はそのグリフを作成する場合の手助けとなる画像または線画を含んでいます。「ガイドレイヤー」（# 表示）はフォント全体の基準となる線（エックスハイトなど）を含んでいます。今のところ、すべてのレイヤーは空です。

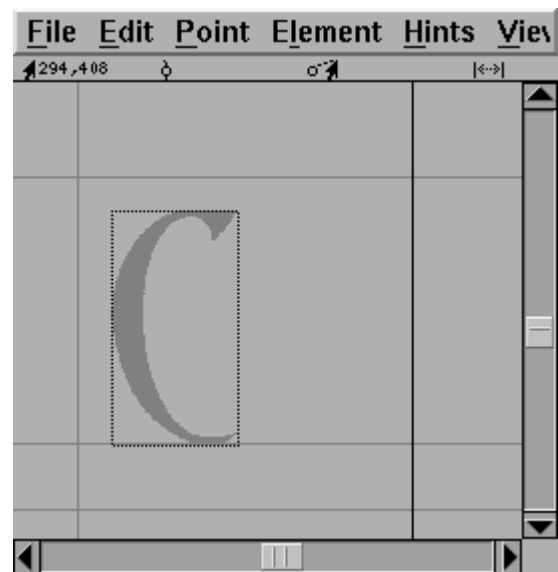
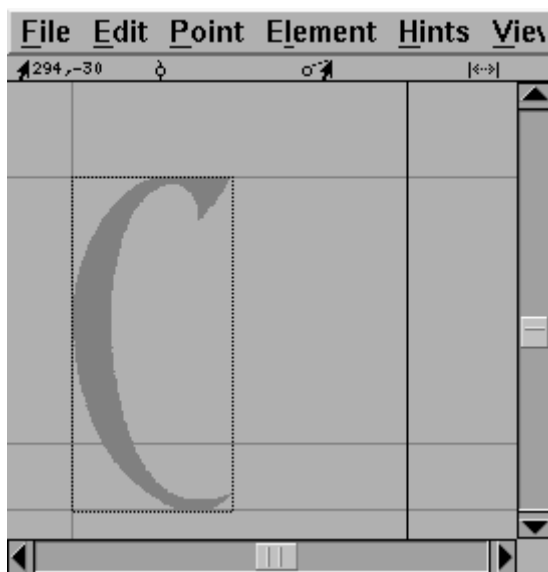
このウィンドウには、グリフの内部座標系が x 軸と y 軸とともに薄い灰色で示されています。グリフの「送り幅」（advance width）を示す線が黒で、ウィンドウの右端に描かれて

-
- 1 C 表示：C = Cubic 〔三次（曲線）〕
 - 2 PS フォント：PostScript フォント
 - 3 Q 表示：Q = Quadratic 〔二次（曲線）〕
 - 4 TTF フォント：TrueType Font
 - 5 F 表示：F = foreground 〔前面〕
 - 6 B 表示：B = background 〔背面〕

います。FontForgeは、新規に作るグリフの送り幅としては、「1em」の幅を割り当てます（これは、PostScript フォントでは通常「1000」ユニットです）。

すでに作成に取り掛かっているグリフの「ビットマップ画像」があると仮定しましょう。それを【**ファイル(F)**】メニューから【**取り込み(I)...**】コマンドを選び、作成したグリフの画像を取り込んでください。グリフの高さは「全角正方形」（em-square）に一致するように拡大／縮小されます。

レイヤーパレットで「背面レイヤー」を選択して、編集可能な状態にします。マウスポインターを画像の端のどこかに合わせ、シフトキーを押しながら隅をクリックし、画像が適切なサイズになるまでドラッグします。そののち、ポインターを画像の暗色の部分に載せてマウスボタンを押下し、画像が適切な位置に来るまでドラッグします。

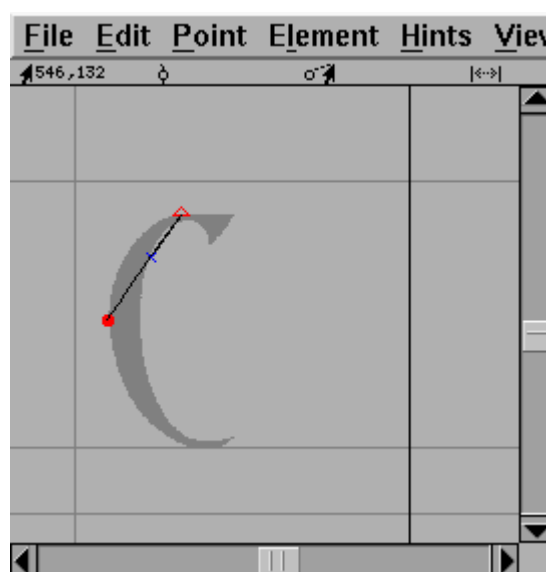
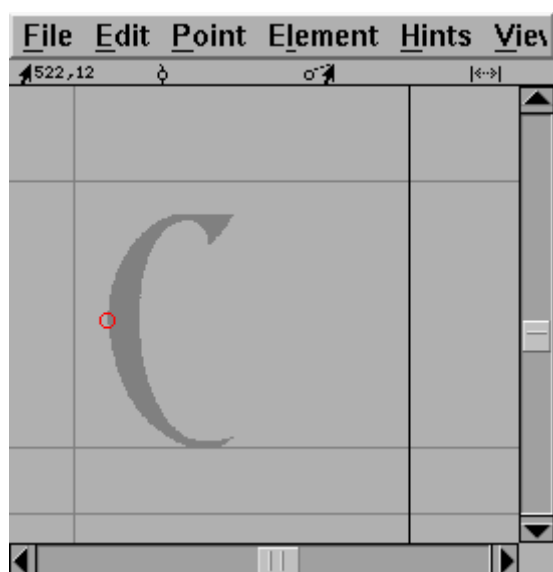


もし「**autotrace** プログラム」をダウンロード済みであれば、【**エレメント(L)**】メニュー⇒【**自動トレース(R)**】で画像からアウトラインを生成することができます。しかし、インストールしていない場合には自分で「**点**」〔輪郭作業点〕を追加しなければなりません。レイヤーパレットの「前面」を選択しアクティブレイヤに変更してから、ツールパレットで丸い「**曲線上の点**」を選択します。その後でポインターを画像の縁に移し、点を追加します。経験上では、点を追加する最良の場所は、曲線が水平または垂直になっているところや、線が折れ曲がった角、そして曲線の変曲点（Sの字曲線のように、曲線が右側に開いた部分から左側から開いた部分へ移るところ）です。この方針に従えば、「**ヒント処理**」がより適切に働くようになるでしょう。

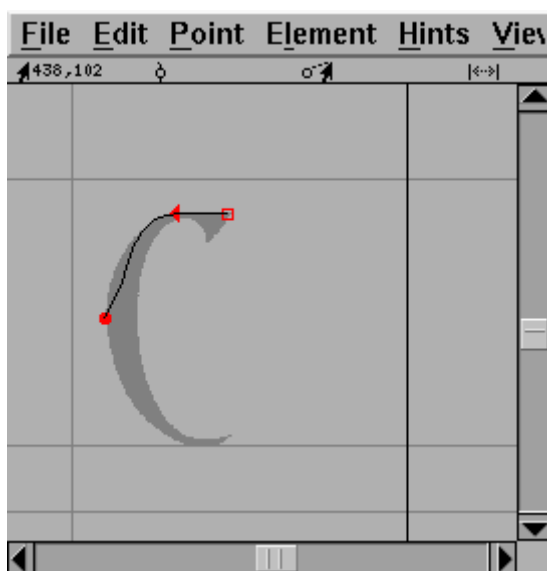
曲線は時計回りに入力していくのが最善です。ですので、次の点は、画像の上端の平らな部分に追加します。ただし、輪郭線がそこから直線に切り替わるので、「**曲線上の点**」ではなく、「**曲線と直線の接点**」（ツールパレット上では小さな三角形のように見えます）のほうが適切です〔そこで、ツールパレットで「点」を選択しなおしてから、画像上に新しい点を追加します。既に「**曲線上の点**」を追加してしまっている場合には、その点を右クリックし、表示されるメニューから適切な点⁷を選択して変更します〕。「**接点△**」は曲線から直線

⁷ 表示されるメニュー： 日本語化が完全ではないので、このメニューでは「○ 曲線上の点 = Curve」「◆ 水平垂直の点 = HVCurve」「■ 角の点」「△ 曲線と直線の接点 = Tangent」と表示されます。

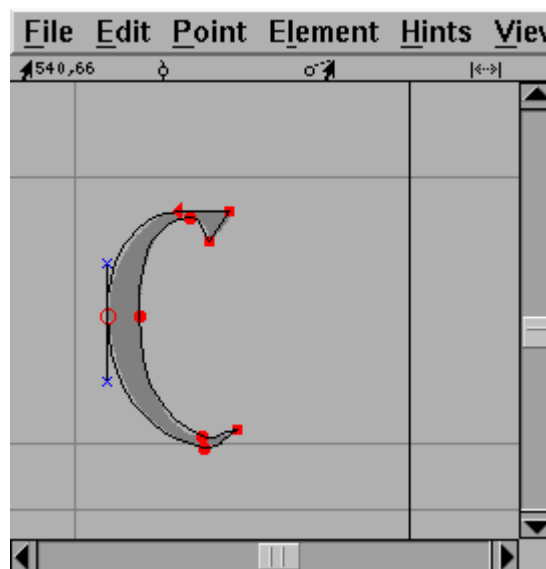
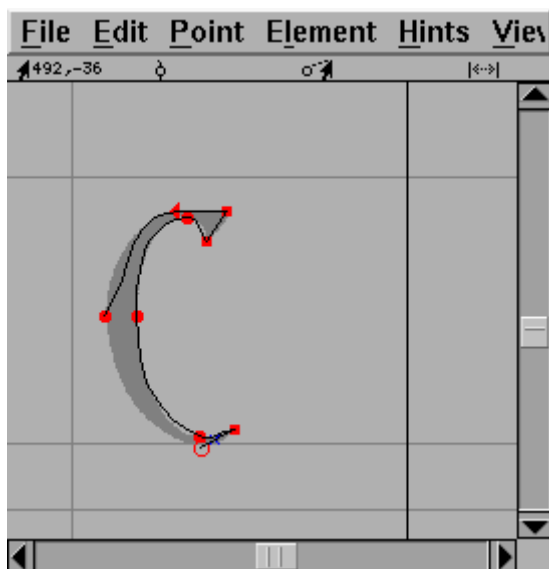
への美しい移り変わりを形づくりします。これは、その点における曲線の傾きが、そこから続く直線の入力された時の傾きと同じに保たれるからです。



今のところ、この「曲線」は画像と全く一致していません。しかし、後で修正しますし、作業を続けるに従っていずれにしろ変形するので、これには気にせず続けてください。ここでは、「接点△」に「**制御点**」（小さな青い×印）がつながって表示されていることに注意しておいてください。次の点は、画像で輪郭の方向が急に変化している箇所に置かなければなりません。〔そこは画像の先端で〕「曲線上の点」も「接点」も不適切ですから、「**角の点**」（ツールパレット上の小さな四角の一つ）を使用しなくてはなりません。



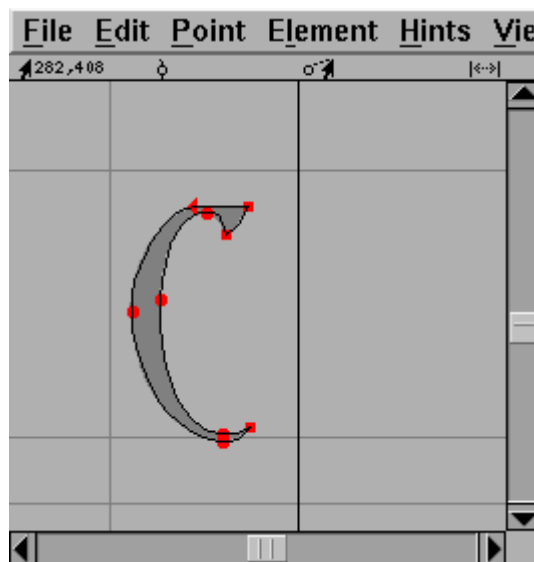
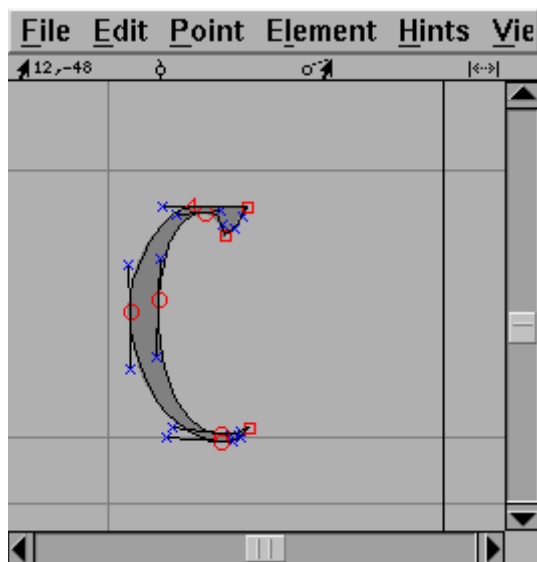
上の図に判るように、先ほどの曲線はわずかながら前より密接に画像に沿っています。パス〔輪郭線〕が閉じる直前のところまで、点の追加を続けましょう〔どの「点」を選択するのか、お間違えなく！〕。



開始点の上に重ねて新しく点を追加すれば、それでパスは閉じます。

ここで、曲線が画像をより正確にたどるようにする必要がありますが、そのためには「制御点」（青い×印）を調整する必要があります。すべての制御点が見えるようにするため、ポインターツール（矢印アイコン）を選んで曲線上でダブルクリックし、それから曲線が画像と同じような形になるように制御点を動かします。

最後に幅を設定します。ポインターツールを使って、マウスを画面の右端にある「**字幅線**」に合わせてクリックし、文字幅が適切な位置に来るまでドラッグして戻します。



これでこのグリフは完成です。

もしあなたが数学に興味のある方であれば、FontForge がウィンドウの左上に表示する座標値に興味を持たれたことでしょう。一般にはこの値を気にしなくても非常に快適にグリフを描画することができますが、興味をお持ちの方のために基本的な情報を提供しておきます：

- 各グリフはそれ自身の座標系をもっています。
- 垂直方向の原点は、フォントのベースライン（ほとんどのラテン文字が載っている線）の上にあります。
- 水平方向の原点が、グリフの描画が開始される基準位置です。上の例では、最初に描画されるのは空っぽの空間ですが、これは非常によくあることで、この空っぽの空間（原点からグリフの左端までの距離）を「**左サイドベアリング**」と呼びます。
- 座標系の単位はフォントの「**em サイズ**」により決まります。これはフォントの「高さ」と「深さ」の和です。上の例ではフォントの高さは「800」で、深さは「200」です。そして「高さの基準線」（「C」の最上部のすぐ上にある線）はベースラインから「800」ユニット離れており、「深さの基準線」は「200」ユニット下にあります。
- ですから、（上記の最後の図に示されている）「282,408」という位置は、カーソルが水平方向の原点から「282」ユニット右に、ベースラインから「408」ユニット上に（すなわち、上端と下端のほぼ中間に）あるということを意味しています。

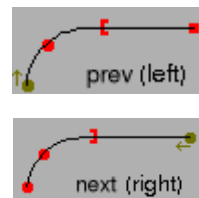


2. スピロ曲線でグリフを作成する

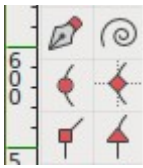

Creating a glyph using spiros

スピロ曲線〔螺旋曲線 Spiro〕は Raph Levien 氏の研究成果で、ベジェ・スプライン曲線に使用される従来のオン・カーブ・ポイント〔曲線上の点〕とオフ・カーブ・ポイント〔曲線外の点〕を組み合わせるのではなく、オン・カーブ・ポイントだけを使用してグリフを作成する代替の方法を提供します。

- 「**G4 点**」 (G4)、緩やかなカーブの曲線に用いられます。
- 「**G2 点**」 (G2)、よりきついカーブの曲線に用いられます。
- 「**角の点**」 (Corner)、角張った接合部分
- 「**前の拘束点**」 (Previous constraint)、輪郭線が曲線から直線になるところに用いられます。
- 「**後の拘束点**」 (Next constraint)、輪郭線が直線から曲線になるところに用いられます。



〔ベジェ曲線方式とスピロ（螺旋）曲線方式との比較〕

	ベジェ曲線	スピロ（螺旋）曲線
ツール パレット		
「点」の 種類	<ul style="list-style-type: none">• 「曲線上の点」• 「水平垂直の点 HVCurve」• 「角の点」• 「曲線と直線の接点 Tangent」	<ul style="list-style-type: none">• 「前の拘束点」※• 「G4 点」• 「G2 点」• 「角の点」• 「後の拘束点」

《※ 訳注》「前の拘束点」「後の拘束点」の原文は“Left Constraint”と”Right Constraint”で、直訳すれば「左拘束点」「右拘束点」です。意味的には「螺旋曲線の始まりを示す拘束点」「終わりを示す拘束点」のような意味と考えられるが、作業時のこの点の位置が必ずしも（視覚的に）「左・右」であるとは限らず、紛らわしく思われるため、「前・後」としてあります。開始点と終止点。

2.1. 手順 1

前章と同様に、Ambrosia フォントの「C」グリフを使って編集してみましょう。

再び、空白のグリフから始めます。ツールパレットには「スパイラル〔螺旋〕の形」をしたボタンがあることに注意してください。このボタンを押すと「螺旋／スピロ曲線 spiro」モードになり、使用できるツールが若干変わります。



(もう一度同じボタンを押すと、ベジェ曲線モードに戻ることができます。)

では、【ファイル(F)】メニューから【取り込み(I)...】コマンドを使って背景画像を読み込み、適切な大きさに調節しましょう（手順がよく判らない場合は、前章を参照してください）。

2.2. 手順 2

「G4 曲線点」を選択します（三行目の左側のツール）。

「G4 曲線点」には、スプライン曲線の傾きが両側で同じになり、スプライン曲線の曲率も同じになるという優れた特性があります。

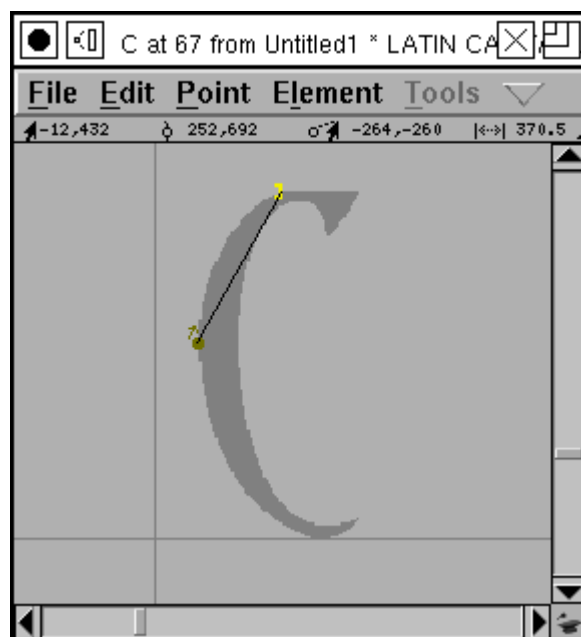
次に、ポインターを画像の上に移動し、クリックしてビットマップ画像の端の一点に「点」を配置します。



2.3. 手順 3

輪郭線を曲線から直線にうまく変化させる「次の拘束点」位置を選択します。

間違った拘束点を選択した場合は（筆者もしばしば間違えますが、その部分で輪郭線が歪んで見え、間違いが後で明らかになります）、拘束点を選択し、**【エレメント(L)】⇒【情報を得る(I)】**を使用して「点の種類」を変更するか、**【点(P)】**メニューを使用します〔該当する「拘束点」を選択します〕。

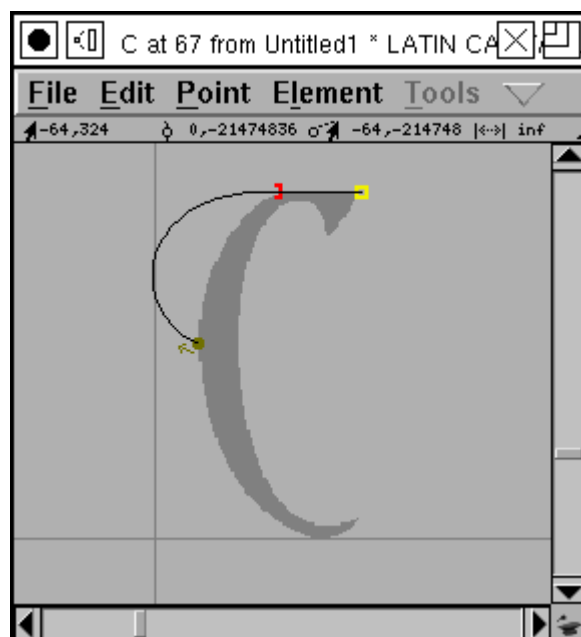


2.4. 手順 4

次に**【ツール(T)】**メニューから「角の点」（正方形のように見えるもの）を選択します。

この「角の点」を輪郭線の傾斜が急に变化する箇所、つまりコーナー〔角〕に置いてください。

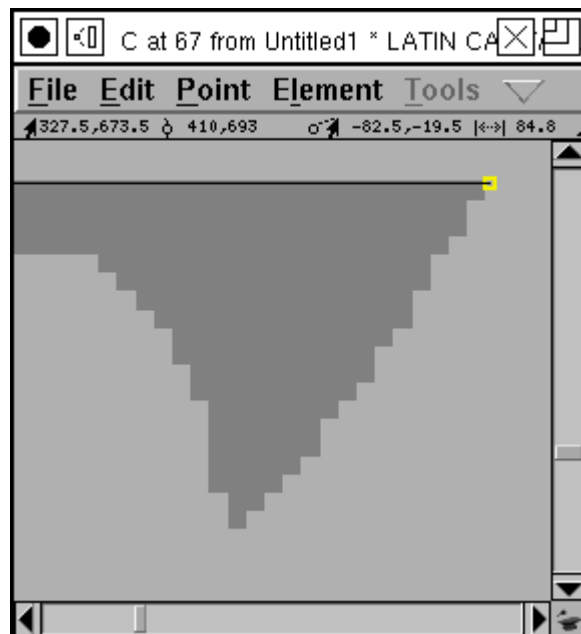
これで、「左接点」について説明する準備が整いました。では、「角の点」に立って、接点の方を向いていると仮定してください。この次の点（この場合は曲がった点）は左側ですか、それとも右側ですか？ 左側の場合は「左接線」を使用し、右側の場合は「右接線」を使用します。



2.5. 手順 5

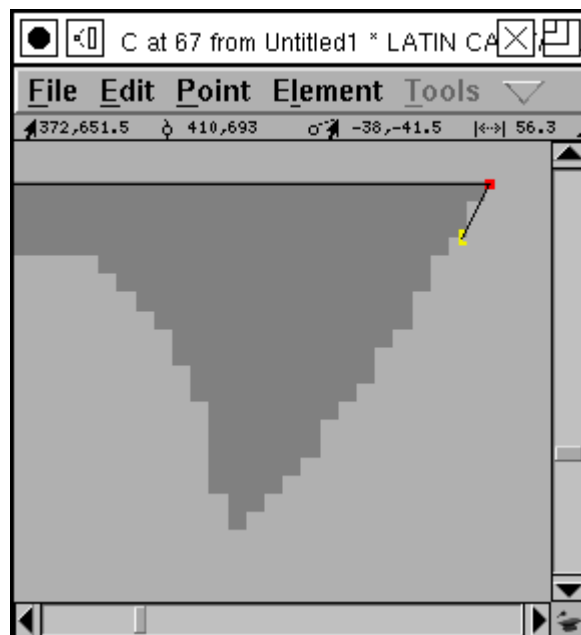
ここで、「C」の上部で面倒な作業を行ないたいと思います。この部分は「セリフ」で、ふたつの角——つまり二箇所の突然の方向転換——の間にわずかに湾曲しています。

より正確な作業を行なうには、画像を拡大して表示する必要があるので、ツール・パレットから「虫眼鏡ツール」を選択し、セリフの中央に移動してセリフが画面全体に表示されるまで数回クリックします。



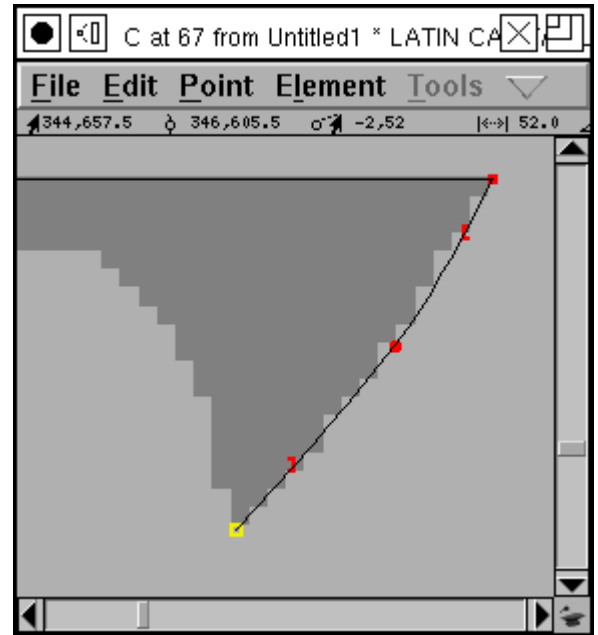
2.6. 手順 6

一般に、「角の点」の両側には「拘束点」（または別の「角の点」）が必要なので、別の「拘束点」を選択する必要があります。今回の場合は、この部分から輪郭線が直線から曲線に変わりますので、「前の拘束点」〔スピロ曲線の開始点〕を使用します。



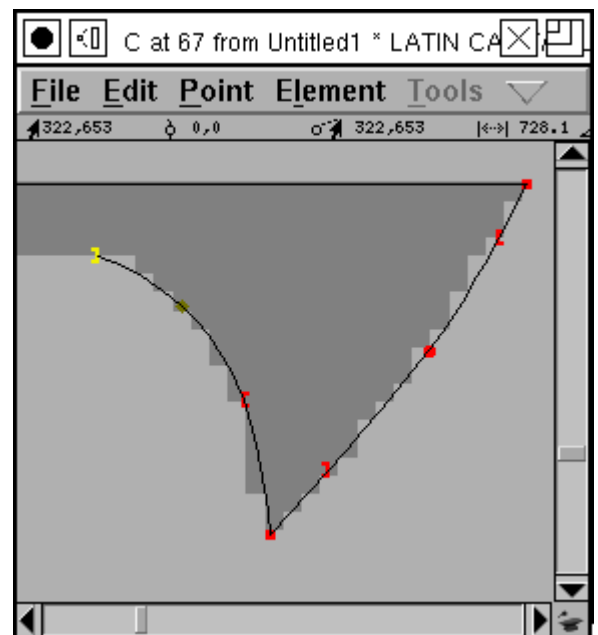
2.7. 手順 7

次に、セリフの滑らかな曲線を作成するために必要な他の点を埋めていきます。



2.8. 手順 8

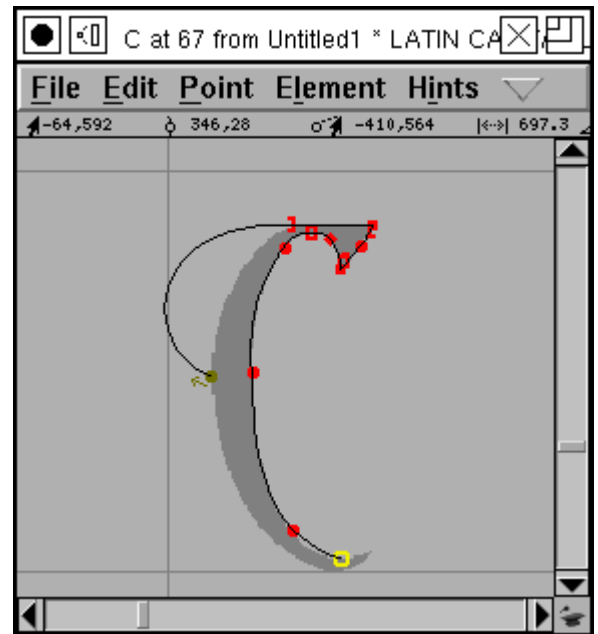
セリフの反対側の滑らかな曲線も同様に作成します。



2.9. 手順 9

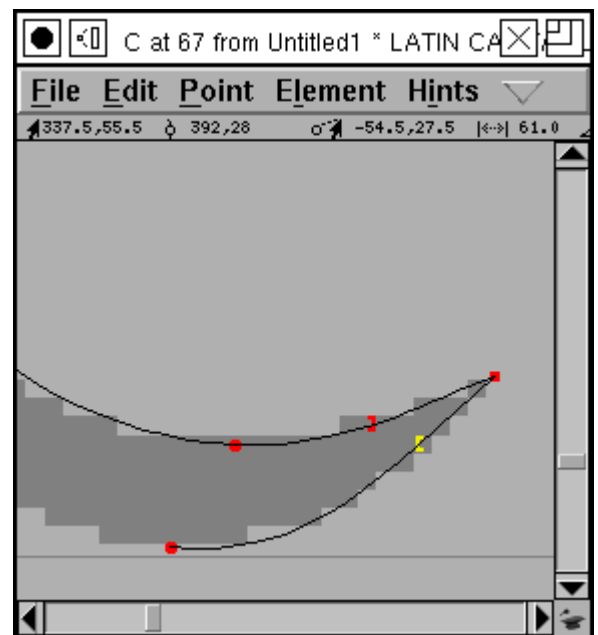
あとは、画像を拡大して見ることはもう必要ないので、虫眼鏡ツールをもう一度掴み、「Alt キー」（メタ・キー、オプション・キー）を押したままにします。すると、カーソルの形が変わり、クリックするとズームアウト〔表示縮小〕します。

続けて、右側の残りの点を設定します。



2.10. 手順 10

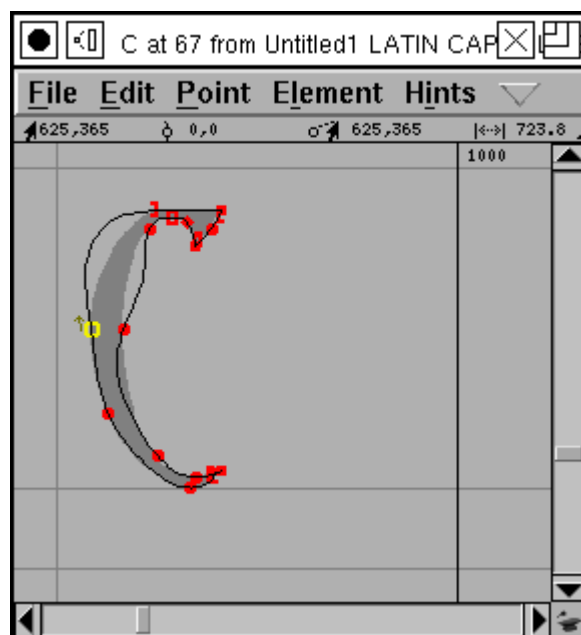
グリフ「C」の下端部に近づくと、再びズームイン〔表示拡大〕する必要があります。



2.11. 手順 1 1

最終的に、グリフの大まかなアウトラインが完成しました。始点をクリックして、曲線を閉じます。

残念ながら、作業結果は期待していたものとは異なり、かなり不規則な膨らみがあります。



これを、以下の方法で修正できます。

1. 点の位置を調節します。

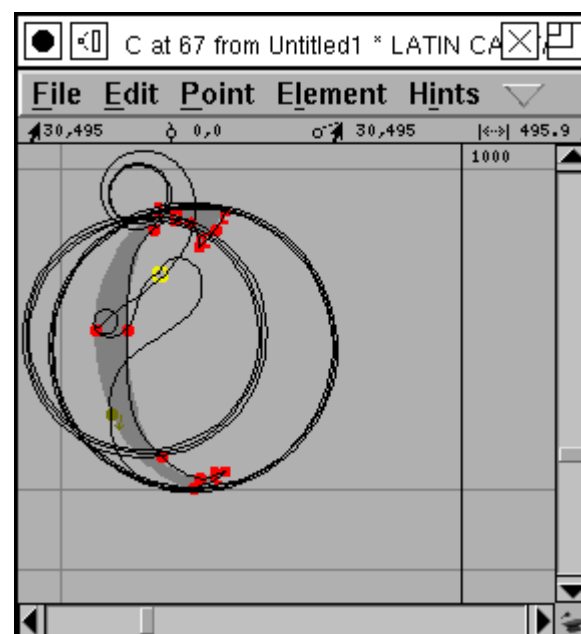
ポインターツールを使用して点をクリックし（または Shift キーを押しながら複数の点を選択し）、ドラッグして移動させます。

2. 新たな点をアウトライン〔輪郭線〕に追加します。

適切な「スピロ／螺旋」ツールを選択して、輪郭線上のどこかでマウスをクリックすると、そこに新しい点が追加されます。追加した点をドラッグして移動します。

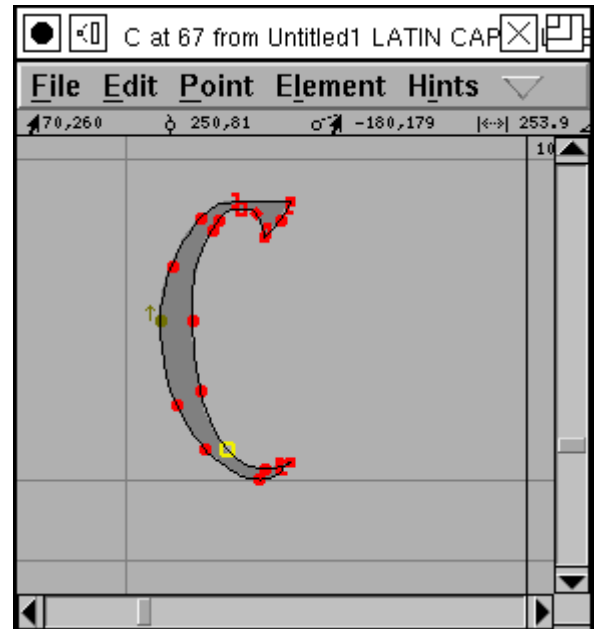
問題を修正している最中に、スピロ曲線コンバーターが対処できない位置まで「点」を移動してしまうことがあります。すると突然、（ある程度）きれいな形状であった輪郭が不規則な螺旋になります。

心配しないでください。点を元の位置に戻せば、通常の状態に戻ります。点をさらに移動すると状況はさらに悪化し、輪郭が完全に消えてしまいます。それも心配しないでください。ただその点を元に戻せば大丈夫です。あるいは、**【編集(E)】 ⇒ 【元に戻す(U)】** を使用します。



思わず作ってしまった螺旋の不思議な美しさを楽しんでください。

(Raph Levien 氏がスピロ曲線〔螺旋曲線〕の研究しています。螺旋はある時点で完全に消えてしまいますが、螺旋にはある種の魅力があり、螺旋が消えていくところを見るのは残念なことです。)



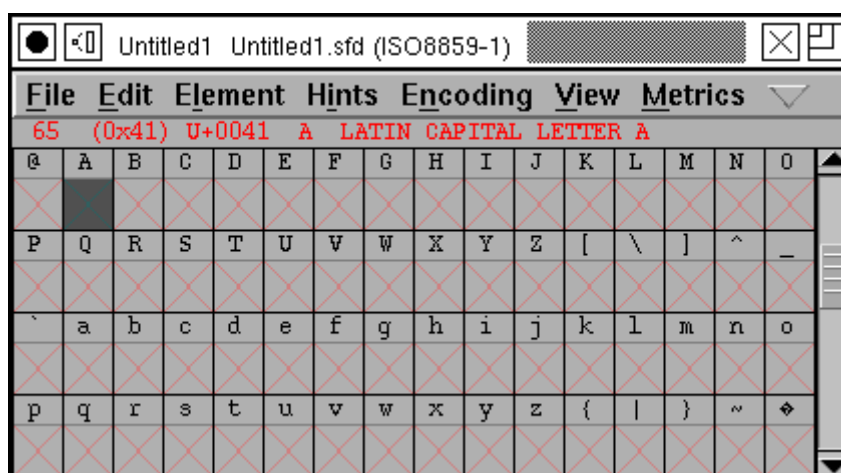
3. アウトライングリフの取り込み

Importing Outline Glyphs

他のベクター・デザイン・プログラム（「イラストレーター⁸」、「フリーハンド⁹」、「インクスケープ¹⁰」、「what-have-you¹¹」など）でグリフのアウトラインをデザインしてから、それを FontForge に取り込む方が快適かもしれません。

前節のふたつでは、ビットマップ画像を取り込み、その画像の周囲をトレースする方法について説明しました。この章の方法は少し異なります。

まず、お気に入りのプログラムでグリフをデザインし、そのプログラムから「eps 形式¹²」または「svg 形式¹³」でデータを書き出します。「イラストレーター」または「フリーハンド」を使用している場合は、おそらく「eps 形式」を選択するでしょう。「インクスケープ」を使用している場合は、おそらく「svg 形式」を使用することでしょう。



次に、FontForge の「フォントビュー」で、デザインしたばかりのグリフを追加するグリフのスロットを開き（ダブルクリック [または右クリック] して「アウトラインウィンドウ」を開きます）、そのウィンドウ内で **【ファイル(F)】 ⇒ 【取り込み(I)】** を選択します。

これにより「**取り込み**」ウィンドウ [下図] が表示され、最初はすべての画像データが表示されます。フォーマット欄のデータ形式を（使用したものに依じて）「EPS」または「SVG」に変更します。次に、作成したばかりのグリフを見つけて読み込みます。

8 Adobe Illustrator： アドビ社が販売するベクター・イメージ編集ソフトウェア（ドローソフト）。

9 Macromedia FreeHand： マクロメディア社（Adobe 社が吸収合併）の 2D ベクター・イメージ編集ソフトウェア。2003 年開発終了。

10 Inkscape： オープンソースのベクター形式画像を作成できる無料で多機能なドローソフト。

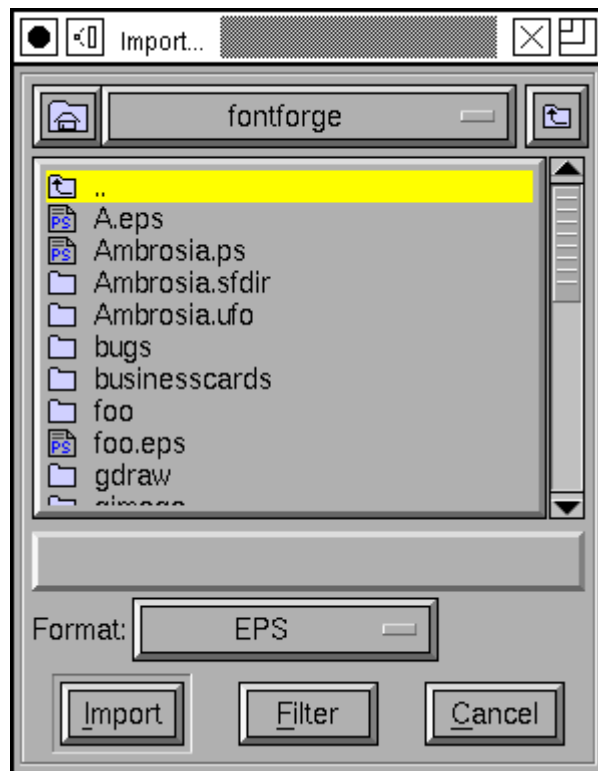
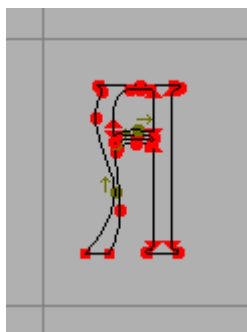
11 What-have-you： 詳細不明。

12 eps 形式： Encapsulated PostScript の略。画像ファイル・フォーマットのひとつ。

13 svg 形式： Scalable Vector Graphics の略。画像ファイル・フォーマットのひとつ。

これで、FontForge のグリフ編集ウィンドウ
〔=アウトラインウィンドウ〕にグリフのアウト
ラインが表示されるはずです。

しかし、表示されたグリフのサイズが間違っ
ている可能性がありますし、間違った位置にあ
るかもしれません。



この例ではふたつの位置が間違っています
(「A」の下部はベースライン——画像の下部
近くの灰色の線——と一致する必要がありますし、左側のベアリングは画像に対して大き
すぎます)。**〔編集(E)〕⇒〔選択(S)〕⇒〔すべて選択(A)〕**を使用して画像内のすべての点
を選択し、その中の点のひとつをマウスで押して、画像全体をベースラインまで下にドラッ
グし、場合によっては少し左にもドラッグします。

例のグリフはサイズも小さ過ぎます。すべてのポイントがまだ選択されていることを確認
してから、**〔エレメント(L)〕⇒〔変形(T)〕⇒〔変形(T)〕**を呼び出します。「**Transform**
〔変形〕」ウィンドウのプルダウン・リストのひとつから**〔一様に拡大・縮小...〕**を選択し、
拡大率を「140%」程度にしてみます。

これには当たり外れがあるのは明らかです。グリフ・デザインに使用したプログラムでグ
リフを統一した方法で作成すると、すべてのグリフに統一した倍率を使用できます。

「**インクスケープ**」では、次のアプローチによりプロセスが簡素化できます。

1. 「インクスケープ」を開く。
2. **〔ファイル(F)〕**メニューから「**ドキュメントのプロパティ(D)...**」を選択。
3. 「表示の単位(U)」に「px」〔ピクセル〕を、「ページサイズ」を「1000 × 1000」ピクセルに設定し、「OK」をクリックします。

もしフォントに 1000 以外の異なる「em 単位」を使用している場合には、
その値を使用してください。ただし、FontForge の既定値は「1000」で
す。

4. 水平ガイドラインを「200 px」の位置に設定し、これを「ベースライン」として
ください。
5. グリフを描きます——最大の難所です！ :-)

6. 完成したグリフを「SVG」ファイルとして保存します。
7. FontForge を起動します。
8. **【ファイル(F)】**メニューから、**【取り込み(I)】**を選択し、フォーマットを**【SVG】**に切り替えて、グリフのデータを選び、「OK」をクリックします。
9. **【エレメント(L)】**メニューから**【変形(T)】**を選択、「Y 値」に「-200」を指定し、「OK」をクリックします。

《※ 訳注》 20230101 版 FontForge では、「変形 (Transform)」ウィンドウでは、「Y 値に -200 を指定する」(set the Y value to -200)を部分が見当たらず、指定方法が不明。



4. グリフについてさらに詳しく

More on glyphs

4.1 グリフの呼び出し方

「**フォントビュー**」（表示画面「フォント一覧ウィンドウ」）は、フォント内のグリフを呼び出すためのひとつの方法を提供します。単に、必要とするグリフが見付かるまで画面をスクロールし、そのグリフをダブルクリックして、グリフを見るためのウィンドウを開いてください。

文字をタイプすれば、その文字に割り当てられたグリフに移動します。

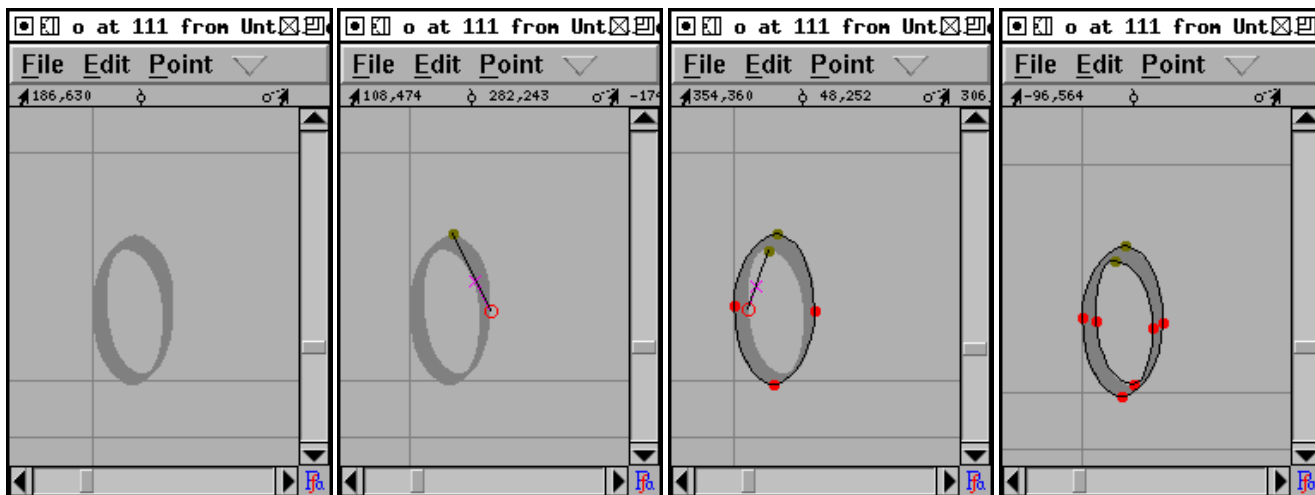
しかしながら、一部のフォントは大規模で（中国語や日本語・韓国語のフォントは数千から数万字にも及ぶグリフを含みます）、フォントビューをスクロールして回るのはグリフを探す方法としては非効率です。**【表示(V)】⇒【移動(G)】**を選択するとダイアログが表示され、そこに「グリフの名前」（または「文字符号」）を入力すれば、目的の文字に直接移動することができます。フォントが Unicode フォントであるなら、グリフを探すときにこのダイアログに「ブロック名」を入力することができます（たとえば、「Alef」でなく「Hebrew」のように）。

いちばん単純な呼び出し方法は、前後のグリフに移動することです。これらの操作は、**【表示(V)】⇒【次のグリフ(N)】**と**【表示(V)】⇒【前のグリフ(P)】**でできます。

4.2 文字「o」の作成——矛盾の無い輪郭線の向き

さきほどの例では、ビットマップで作成した文字は画像のサイズいっぱいに描かれていたので、FontForge に画像を取り込んだ時、最初にプログラム内で拡大／縮小を行なう必要がありました。しかし一般的には、文字の画像を作成する時には、その周りにどれだけの空白を置くかをあらかじめ想定しているのがふつうです。画像の高さがちょうど全角（em）に等しい場合、FontForge は自動的に正しいサイズに拡大／縮小します。これ以降の例では、すべての画像は、全角にぴったり一致するだけの空白を文字の周りに含んでいるものとします。

次の事例を説明するために、「フォントビュー」文字「o」の区画をダブルクリック〔または右クリックで「アウトラインウィンドウ」を表示〕して、「o_Ambrosia.png」の画像を取り込みます。



文字“o”の編集の各段階

最初のアウトラインは時計回りに、二番目を反時計回りに描かれていることに注意してください。PostScript・TrueType のどちらにおいても、グリフの輪郭は特定の向きに描かれなくてはならないことに決まっております（ちょっと面倒なことに、PostScript と TrueType では描く方向が互いに逆方向になっていますが）、**FontForge では外側の輪郭線はすべて時計回りに、内側の輪郭線はすべて反時計回りでなくてはなりません。**

外側の輪郭と内側の輪郭を逆向きにするのを忘れた場合、この図の左側のような結果〔o 中央の白抜き部分も塗り潰されてしまう字形〕が得られるでしょう。外側の輪郭を時計回りにするというのを忘れた場合のエラーはもっと軽微なものでしょうが、それでも一般的には、グリフを「**ラスターライズ**¹⁴」された時にあまり愉快ではない結果となるでしょう。



【メモ】 **技術的で分かりにくい注記：** ラスタライザ〔ドット変換プログラム〕の正確なふるまいは様々です。初期の PostScript ラスタライザは「**非ゼロ回転数塗り潰しルール**」を使っていましたが、最近のものは「**偶奇塗り潰しルール**」を使います。TrueType は「**非ゼロ回転数塗り潰しルール**」です。上で行なった説明は「**非ゼロ回転数塗り潰しルール**」です。「**偶奇塗り潰しルール**」の場合では、パス〔輪郭線〕の向きがどちらであっても問題なく「o」は正確に表示されます（ただし、ヒント処理のときには微妙な問題をひき起こすでしょう）。

「**偶奇塗り潰しルール**」を用いたグリフの塗り潰し方法は、あるピクセルから（任意の方向に）直線を引き、輪郭線と交差する回数を数えるというものです。この回数が偶数なら、そのピクセルは塗り潰しません。回数が奇数ならそのピクセルを塗り潰します。「**非ゼロ回転数塗り潰しルール**」でも同じ線を引きますが、この方式では、時計回りの輪郭線と交差するたびに交差回数に「1」を加え、反時計回りの線に交差するたびに「1」を引きます。結果が「0」ならば、そのピクセルは塗り潰さず、それ以外の値ならば塗り潰します。

14 rasterize： デジタル画像データ（ベクトル画像／ベクター形式画像）を画面表示や印刷用にドットの集合に変換する仕組み。

【エレメント(L)】 ⇒ 【アウトラインの向きを修正(C)】 コマンドは、選択された各輪郭を調べ、それが外側の輪郭になるのか内側の輪郭になるのかを判断し、輪郭線が間違って逆向きに描かれている場合にはその向きを修正します。



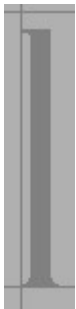

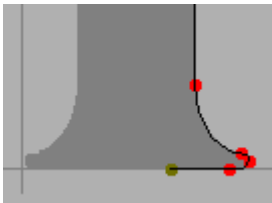

5. 整合したステム幅、セリフと高さをもつ文字の作成法

Creating letters with consistent stem widths, serifs and heights

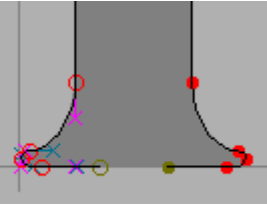
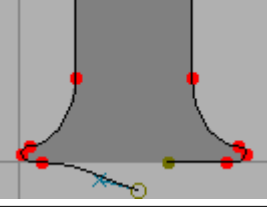
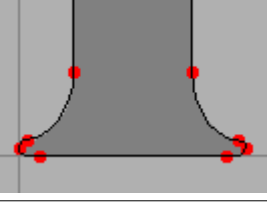
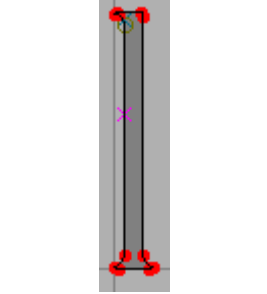
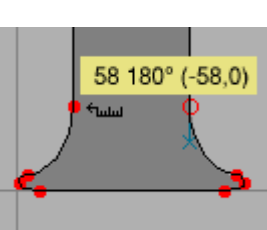
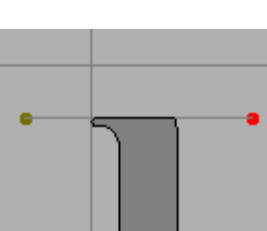
多くのラテン文字（ギリシア文字、キリル文字）のフォントには「**セリフ**」があります。「**ステム**」〔縦軸〕の終わりにある特徴のある終端形の飾りのことです。ほとんどすべての「LGC フォント¹⁵」においては少数の異なるステム幅のみを用いるようになっています（たとえば、小文字エル「l」と小文字アイ「i」の垂直ステムはおそらく同じ幅になっているでしょう）。

FontForge には、デザインの一貫性を維持するためのよい方法が備わっているわけではありませんが、一貫性をチェックしたり矛盾を発見したりするためのコマンドは多数含まれています。

まず小文字「l」から始めましょう。お馴染みの作業を進めて、ビットマップ画像を取り込み、そのアウトラインを決定していきます。

	取り込んだ画像
	拡大ツールを使って下部のセリフを確かめます。ここが左右対称な形になっていることに注意してください。
	セリフの右半分のアウトラインを作ります。
	アウトラインを選択し、 【編集(E)] ⇒ 【コピー(C)] 、 【編集(E)] ⇒ 【貼り付け(P)] を行ってから、最後に 【エレメント(L)] ⇒ 【変形(T)] ⇒ 【変形(T)...】 を順に選択して「Transform ウィンドウ」を表示し、（プルダウンリストから） 【反転...】 を選択して 【水平】 にチェックを入れます。


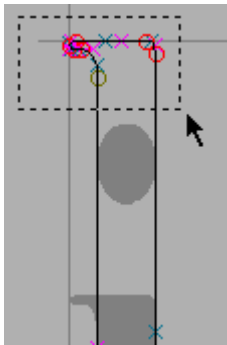
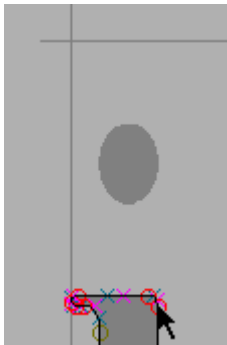
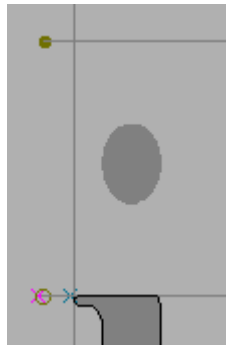
¹⁵ LGC フォント： L（ラテン文字）、G（ギリシア文字）、C（キリル文字）のフォントの略。

	<p>反転したセリフを左側にドラッグし、グリフの左端が合うまで動かします。</p>
	<p>パスを選択解除し、終端点を選択してドラッグし、反対側の終端点に重なるまで伸ばします。</p>
	
	<p>グリフを完成させます。</p> <p>まだやるべき事がふたつ残っています。最初に「ステムの幅」を測定し、次に「l」の「高さ」に印をつけましょう。</p>
	<p>「物差しツール」を「ツールパレット」から選択し、それをステムの片方の端からもう片方にドラッグしましょう。小さなウィンドウが現れて、幅が「58 ユニット」、ドラッグ方向が「180°」、ドラッグしたのは「水平に -58 ユニット」、垂直に「0 ユニット」であると表示されます。</p>
	<p>「レイヤーパレット」に移動し、【ガイド(G) ラジオボックス¹⁶を選択し（これでガイドレイヤが編集可能になります）</p> <p>「l」の高さに合わせて直線を引きます。この線はすべてのグリフから見ることができ、このフォントの「アセント」の高さを表示しています。</p>

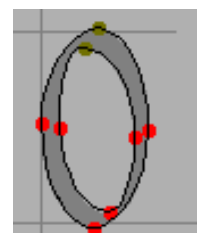
今度は「i」を作りましょう。このグリフは短い「l」の上に点を打ったものに非常に似ています。なので「l」のグリフを「i」の区画にコピーしましょう。これにより自動的に正しい「ステム幅」と正しい「送り幅」（advance width）が設定されます。グリフのコピーは、フォントビューから「l」を含む区画を選び、**【編集(E)】** ⇒ **【コピー(C)】** を押すことでも、アウトラインビューから **【編集(E)】** ⇒ **【選択(S)】** ⇒ **【すべて選択(A)】** した後に、**【編集**

16 radio box： 「ラジオボタン」は選択肢の中から必ず一つだけ選択（排他的）できるフォームで、通常「○ 選択肢 1」のように「○」ボタンで表示されます。「レイヤーパレット」には「□」ボタンが並ぶ、任意の数だけ複数の項目を選択可能な「チェックボックス」が用いられているので、「チェックボックス」の誤記、または最近の FontForge バージョンで仕様変更されている可能性があります。


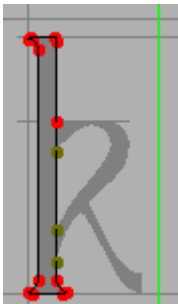
【E】⇒【コピー(C)】を実行することでも、行なうことができます。同様に、ペーストもフォントビューから「i」の区画を選んで【編集(E)】⇒【貼り付け(P)】を押すことでも、アウトラインビューから「i」のグリフを開いて【編集(E)】⇒【貼り付け(P)】ででも、可能です。

			
「i」の画像を取り込み、次に「l」のグリフをコピーします。	「l」の上端のセリフを選択します。	ドラッグして、正しい高さまで下ろします。	ガイドレイヤに移動し、「エクスハイト」に合わせて線を追加します。

先ほど作成した「o」に少しだけ戻りましょう。「o」は、エクスハイト〔小文字の高さ〕の印として置いたガイドラインから上に（ベースラインの下にも）少しはみ出しているのが分かるでしょう。これは「**オーバーシュート**」と呼ばれるもので、目の錯覚を補正するためのものです。曲線がエクスハイトと一致しているように見えるためには、（直径の）およそ3%膨らんでいる必要があります。



「k」を見てみましょう。ここでも「l」をコピーし、適切な画像をインポートします。

	「k」の画像を取り込み、「l」のグリフをコピーします。エクスハイトの線が（期待どおりに）「k」と一致していることに注意してください。また、「l」の幅が「k」には不釣り合いであるため、輪郭を選択して、画像の線幅と重なり合うまでドラッグしなくてはならないことにも気をつけてください。
	ツール・パレットから「ナイフツール」を選び、「l」字の縦の輪郭線を、「k」の右側の部分と接続する辺りの適切な位置で分割します。

	<p>分割した点と点の間の線分を削除してください。これを行なうひとつの簡単な方法は、線分自体を掴み（これにより両端の点を選択されます）、【編集(E)] ⇒ 【クリア(L)] を使うことです。</p>
	<p>端点をセレクトし、【点(P)] ⇒ 【角の点(C)] で「角の点」に変換します。</p>
	<p>それから外側の輪郭を描きます。</p>
	<p>そして内側の輪郭も作ります。最後に、【編集(E)] ⇒ 【選択(S)] ⇒ 【すべて選択(A)] し、【エレメント(L)] ⇒ 【パスの方向を修正(C)] を行ないます。</p>



6. アクセント付グリフ、合字、検索、機能

Accented glyphs, ligatures, lookups and features

6.1. アクセント記号付きグリフの組み立て

ラテン文字、ギリシア文字およびキリル文字はすべて、大量の補助的なアクセント記号付きグリフを含んでいます。FontForge には、基底グリフ (base glyphs) からアクセント記号付きグリフを組み立てる方法がいくつかあります。

いちばん分かりやすいやり方は単純なコピー&ペーストです。グリフ「A」をコピーして「Ä」の区画に貼り付け、次にアクセント記号の「チルダ」をコピーして「Ä」の区画に追加貼り付けします（「追加貼り付け」が「貼り付け」とは微妙に異なることに注意してください。[貼り付け(P)] コマンドは貼り付けの前に文字を消去しますが、[追加貼り付け] コマンドは、その文字の元の内容にクリップボードの中身を付け加えます）。それから「Ä」を開いてアクセント記号の位置を調整し、「A」の真上に中心が来るようにします。

このやり方は、特に効率的というわけではありません。それは、もし文字「A」の字形を変更した場合には、その文字をコピーして組み立てられたすべてのアクセント記号付きグリフを作成し直さなければならないからです。FontForge にはグリフへの「参照」という概念があります。そこで、「A への参照」をコピーして貼り付け、「チルダへの参照」をコピーして追加貼り付けし、それから「A」の上のアクセント記号の位置を再調整します。

このやり方では、後で「A」の字形を変更しても、「Ä」に含まれる「A」の字形や「Ä」の文字幅を、魔法のごとく自動的に更新してくれます。

というのも、FontForge は「Ä」が文字「A」とアクセント記号「チルダ」から造られていることを知っているので、FontForge 自体がこのふたつのグリフへの参照内容を「Ä」に自動配置し、アクセント記号を「A」の真上に揃えることにより、アクセント記号付きグリフを簡単に作成できるのです（Unicode 規格では、Unicode に含まれる各アクセント記号付きグリフの構成要素〔グリフ部品〕をリストにしたデータベースを提供しており、FontForge はこの情報を利用しています）。

試しに、ファイル「tutorial/Ambrosia.sfd¹⁷」を開き、符号位置が「0xc0～0xff」の範囲にあるすべてのグリフを選択して [エレメント(L)] ⇒ [組み立て(U)] ⇒ [アクセントつきグリフを構築(B)] を押せば、すべてのアクセントつき文字がお任せで現れます（この範囲にはアクセント記号付き文字でない物が数文字ありますが、それらは空白のままになります）。

FontForge はアクセント記号の位置決めに経験則（ほとんどのアクセント記号はグリフの

¹⁷ tutrial/Ambrosia.sfd： このファイルの所在は不明。チュートリアルで用いられている「Ambrosia」フォントは FontForge の生みの親である George Williams 氏が作成したフォントで、以下のサイトからダウンロード可能 (<https://www.fontspace.com/ambrosia-font-f4276>)。「*.sfd」ファイルは、FontForge 独自の作業用ファイル（フォントを作成・修正し、フォントとして出力する前のデータ）です。

最も高い位置の上に、中央配置します）を用いていますが、時にこれが悪い結果を招きます（たとえば、「u」の字のふたつの縦線の一方が他方よりもわずかに高ければ、アクセント記号はグリフの中央の上に来ないで高い方の縦線の上に来てしまいます）ので、アクセント記号付きグリフを作成した後は、自分の目で確かめる心構えが必要です。一、二箇所の調整が必要となる場合があります（し、基底グリフのほうを少し再設計する必要さえあるかもしれません）。

リガチャー

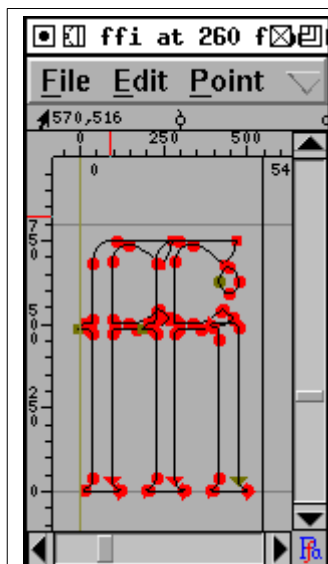
6.2. 合字の作成

Unicode には多数の「**合字**」グリフが含まれています（ラテン文字では「Æ」、「OE」、「fi」など、アラビア文字では数百にも及びます）。これに関しても、Unicode は標準的な各合字の構成部品をリストにしたデータベースを提供しています。

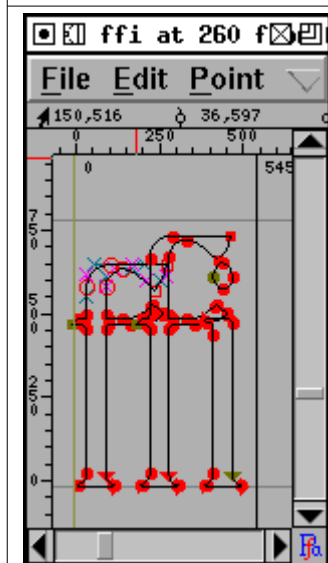
FontForge 自体が美しい合字を作ることは不可能ですが、**【エレメント(L)】⇒【組み立て(U)】⇒【複合グリフを構築(C)】**を用いれば、その合字のすべての構成要素〔グリフ部品〕をひとつのグリフ上に配置することが可能です。これで、合字のデザインが（少なくともラテン文字では）少しは楽になります。これが機能するには、元のグリフと結合するグリフ（円とグリフで識別されます）がオーバービューに存在する必要があります。どれかが存在しない場合、メニュー項目が無効（グレー表示）になります。グリフがどの構成要素で構成されているかを確認するには、**【エレメント(L)】⇒【グリフ情報(G)...】**から「構成要素」タブの内容を確認してください。

合字組み立てのための手順

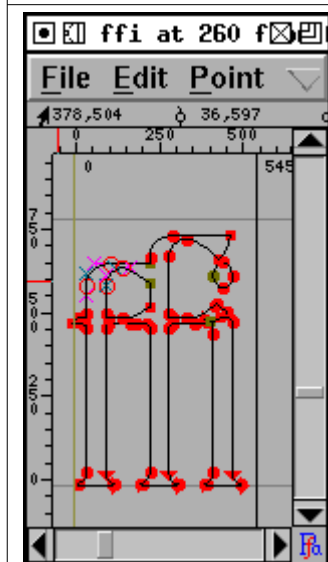
【エレメント(L)】⇒【グリフ情報(G)...】 ダイアログを使い、「グリフの名前」を決め、「合字」であることを指定します。次に **【エレメント(L)】⇒【組み立て(U)】⇒【複合グリフを構築(C)】** で合字の「構成要素への参照」を挿入します。



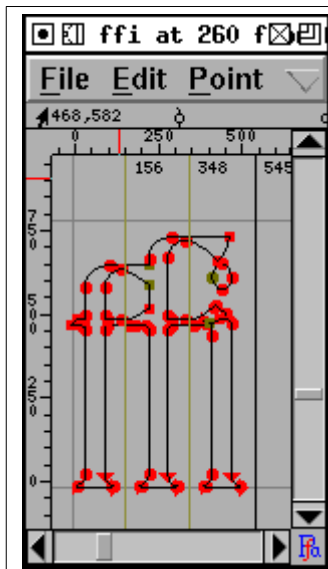
【編集(E)】 ⇒ 【参照を解除(N)】 コマンドを使って、参照を輪郭線に変換します。



各要素がより揃って美しく見えるように調整します。ここでは最初の「f」の縦軸を下に動かしています。



【エレメント(L)】 ⇒ 【重複処理(V)】 ⇒ 【重なり合う図形を結合(R)】 コマンドを使って、グリフを整理します。



最後に合字キャレットの線〔合字でテキストカーソルが置かれるべき位置〕を、原点からもっと適切な構成要素間の場所に設定します。script

一部のワードプロセッサには、合字の内部（合字の各構成要素の間の位置）にキャレットを置いて編集できる機能があります。これは、ワードプロセッサの利用者が、合字を取り扱っていることを意識する必要が無く、そこに構成要素が置かれているときとほとんど同じふるまいを目にするということです。しかし、ワードプロセッサがこの機能を可能にするためには、フォントデザイナーが適切なキャレット位置として設定した地点について、いくつかの情報を得る必要があります。FontForge は、グリフが合字であると判断すると、合字要素の間の数と同じ数のキャレット位置境界線をその文字に挿入します。キャレット線は原点に置かれます。もしそのまま原点に置いたままにしている場合には、FontForge はそれらを無視します。しかし、合字を作成し終えたら、ポインタツールを使って原点にあるキャレット線を移動し、ボタンを押してキャレット線の正しい位置に設定するとよいでしょう（この機能に対応しているのは TrueType と OpenType のみです）。

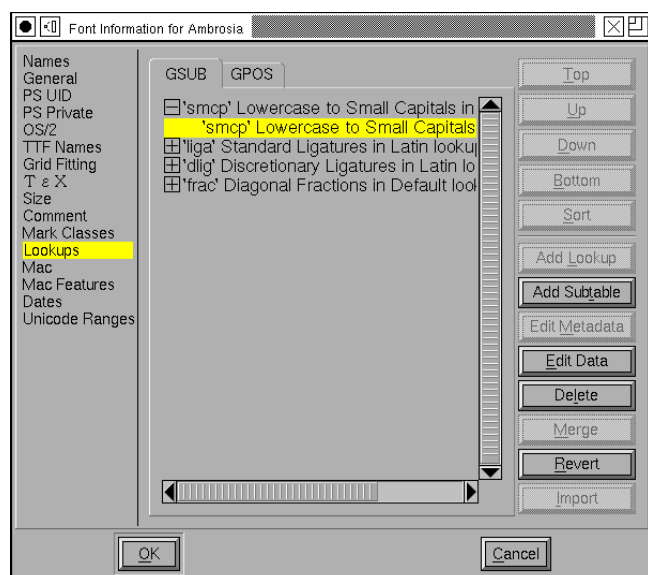
インド諸語の用字系には非常に多くの合字が必要ですが、Unicode ではそのような合字に対してエンコーディング〔符号化〕を割り当てていません。Unicode に含まれない合字が必要な場合、以下のようにすれば構築することができます。まず[エンコードされていないグリフをフォントに追加](#)し（フォントが Unicode フォントの場合は、私的使用領域の区画を使用できます）、そのグリフに名前を付けます。名前は重要で、正しく命名すれば、FontForge はそれが合字であり、どんな要素から構成されているかを知ることができるからです。たとえば、グリフ「longs」、[「longs」](#)、および「l」からなる合字を作成したい場合には「longs_longs_l」という名前を付けます。Unicode の 0D15、0D4D、および 0D15 から合字を作成する場合は「uni0D15_uni0D4D_uni0D15」と命名します。

合字に名前を付け終え、（[【エレメント\(L\)】](#) ⇒ [【組み立て\(u\)】](#) ⇒ [【複合グリフを作成\(C\)】](#)）を使用して）その構成要素を挿入したならば、おそらくグリフを開きたくなるでしょう。[参照を解除](#)して、それらが美しい形になるように編集してください（上記の手順で）。

6.3. 検索と機能

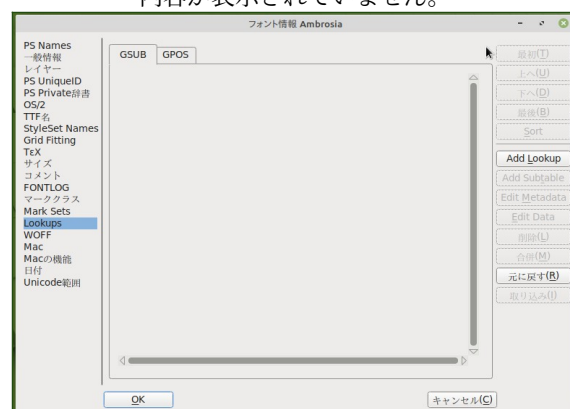
Lookups and Features

残念ながら、単に合字グリフを作成するだけでは十分ではありません。フォントには、グリフが合字であること、およびグリフがどの構成要素から構築されているかを示す情報も含める必要があります。



旧英語版画面

日本語版は「検索」未設定のため、
内容が表示されていません。



2023.01 日本語版画面

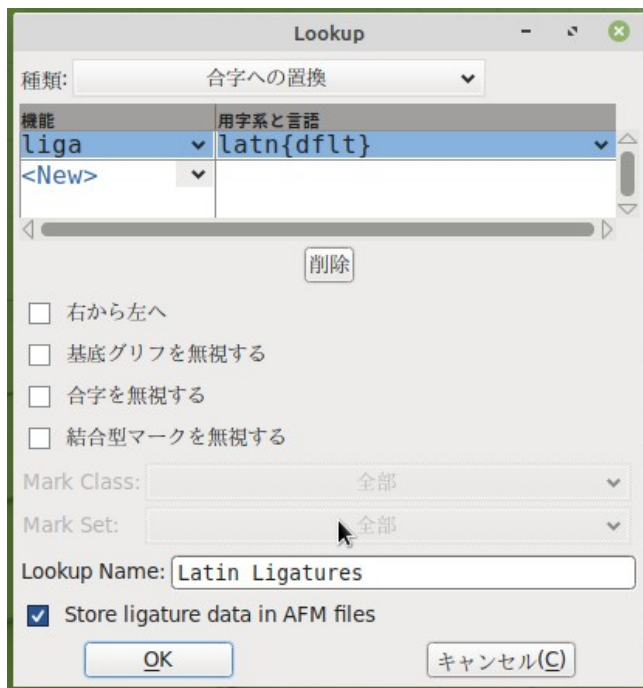
OpenType では、これは「**検索 Lookups**」と「**機能 Features**」によって処理されます。「検索」は、変換情報を含むフォント内のテーブルの集合体です。「機能」は「検索」の集合体であり、その一連の「検索」が何をしてくれるのかについての「意味情報¹⁸」〔機能の内容〕をフォント外の世界に提示するものです。したがって、上記の例では、「検索」には「f」＋「f」＋「i」を「ffi」に変換する必要があるという情報が含まれますが、「機能」はこれがラテン文字の標準合字であることを示します。

したがって、初めて合字グリフを作成するときは、そのグリフの情報が内在する「検索」（および「検索サブテーブル」）を作成する必要があるでしょう。後続の合字はおそらく同じ「検索」と「サブテーブル」を共用できます。

（これはラテン語の合字にはやり過ぎのように思えるかもしれませんが、おそらくやり過ぎなのでしょうが、より複雑な書記体系ではその複雑さが必要なのです）。

[エレメント] ⇒ [フォント情報] コマンドの「検索 (Lookup)」タブを開き〔上図参照〕、右側にある **[Add Lookup (検索を追加)]** ボタンを押します。すると、新らしくダイアログが開いて、新たに設定する「検索」内容の「属性 attribute」を入力できます。

18 semantic information (意味情報) : 機械が解釈可能な形式や構造情報と、人間が解釈可能な意味内容を兼ね備えた情報。



最初に「種類」〔検索タイプ〕を選択する必要があります。合字の場合、ここは「合字への置換」である必要があります。その後、この「検索」を「機能」、「文字」、「言語セット」と関連付ける〔バインドする〕ことができます。合字の「ffi」はラテン語文字の標準合字であるため、「liga」タグと「latn」文字に関連付ける必要があります。（〔ウィンドウ上部の「機能／用字系と言語」と書かれた表にある〕「liga」の右側にある小さな下矢印のボックスをクリックすると、「機能」のいわゆる「フレンドリー名」のプルダウン・リストが表示されます。「liga」は「標準の合字」と表示されます）。

「言語」の設定選択は少し難しいです。この合字「ffi」はおそらく、ラテン文字を使用

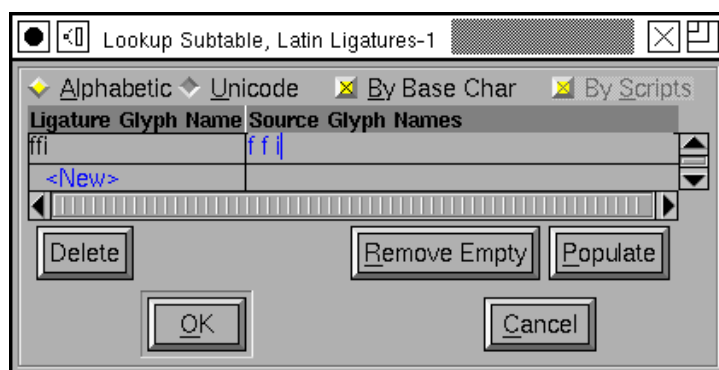
するトルコ語を除く、すべての言語で有効になるはずです（トルコ語では「点のない i」（ドットレス・アイ）が使用され、「fi」および「ffi」合字の「i」にドットがあるかどうかは不明です）。このため、対象言語にはトルコ語を除くすべての言語を列記したいのですが、それはかなりの言語数です。代わりに、慣例では、フォント内のどこにも言語が明示的に記載されていない場合、その言語が「デフォルト」言語として扱われるようになっています。したがって、この合字の「機能」をトルコ語に対して有効にしないようにするには、言語リストにトルコ語を明記した別の「機能」を作成する必要があります。

「機能」リストの下には一連のフラグがあります。ラテン語の合字では、これらのフラグを設定する必要はありません。アラビア語では「右から左」と「結合マークを無視」の両方を設定したい場合があります。

次に、「検索」に名前を付ける必要があります。この名前はユーザーが使用するためのもので、実際のフォントでは表示されません。ただし、名前は他の「検索」の名前とは異なっている必要があります。

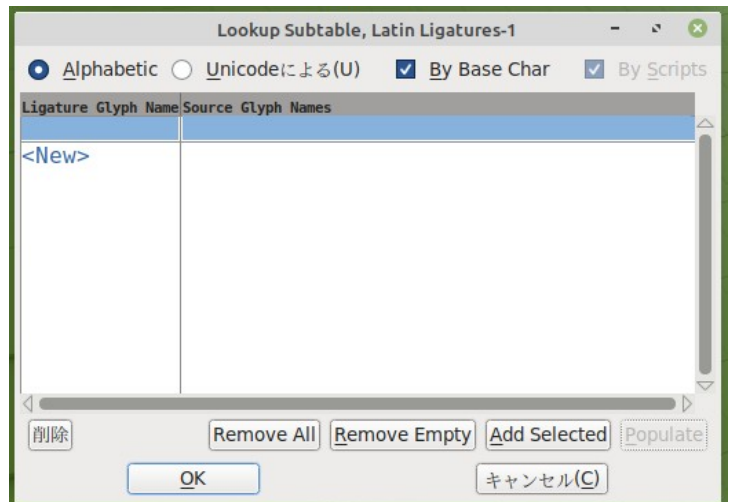
最後に、この「検索」の合字を「afm¹⁹」ファイルに保存するかどうかを決定します。

「検索」を作成したら、その「検索」内にサブテーブルを作成する必要があります。「検索」行（フォント情報の「Lookups」タブ内）を選択し、**[Add Subtable]** ボタン〔サブテーブルの追加〕を押します。これは非常に単純なダイアログ画面です。サブテーブルの名前を指定するだけで、別のダイアログ画面が表示され、（最終的



19 afm = Adobe Font Metrics File : アドビ社が提唱した PostScript フォントの文字情報を扱う仕組み。

に) 合字情報を保存できるようになります。



202301 日本語版での「検索 Subtable」設定画面レイアウト

《警告》 OpenType エンジン、その用字系において適正であると判断した時に限り、「機能」を適用します（ラテン文字では、Uniscribe²⁰ は「liga」機能を適用します）。さらに悪いことに、いくつかのアプリケーションはおそらく一切の機能を適用しないことを選んでいます（Word はラテン文字の合字処理を行ないません。しかし、2007 年版から変更されているかもしれません²¹）。Microsoft は、Uniscribe においてどの用字系にどの機能を適用するかを文書化しようとしていますが、Word および Office はデフォルトと全く異なる振舞いをするため、これはほとんど参考になりません。



20 Uniscribe： Microsoft Windows において、Unicode によって符号化されたテキストを描画するためのレンダリングサービス [出典 Wikipedia]

21 Word 2007 release： この「日本語版チュートリアル」（2024）は、英語ウェブ版（おそらく 2012 年頃完成？）に基づいていますので、内容的に 10 年程のタイムラグがあります。

7. メトリック、間隔、カーニング

Metrics, spacing and kerning

《※ 訳注》 この『チュートリアル』で用いられている操作画面の呼称は、FontForge 上では「フォントビュー」= 起動時の「フォント一覧ウィンドウ」、「アウトライン（グリフ）ビュー」= 「アウトラインウィンドウ」、「メトリックビュー」= 「メトリックウィンドウ」と読み替えて利用してください。

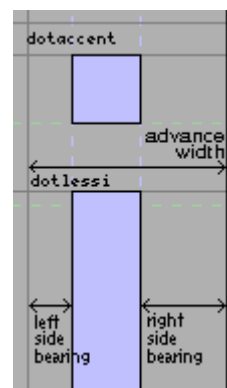
7.1. メトリックの検査と制御

グリフの形を作成し終わったら、次にする必要があるのはグリフの間の空間を正しく設定することです。ふたつのグリフの間隔はすべて、最初のグリフの後の空白と次のグリフの前の空白のふたつの構成要素からなります。左から右に文字を書く世界では、これらはそれぞれ「**右サイドベアリング**」と「**左サイドベアリング**」と呼ばれます。

「左サイドベアリング」は、簡単な操作で変更することができます。

【編集(E)] ⇒ 【選択(S)] ⇒ 【すべて選択(A)] を（アウトラインビューで）行ない、選択されたオブジェクトを適切な位置に移動します。「右サイドベアリング」は「送り幅」を表す線を選択し、適切な位置に調整することによって変更できます。

しかしながら、一般的には個別のグリフのメトリックを単独で設定するのではなく、他のグリフの間に囲まれた文脈の中で眺め、その観点から設定したほうがいいのです。【ウィンドウ(W)] ⇒ 【メトリックウィンドウを開く(M)] コマンドを使用してください。

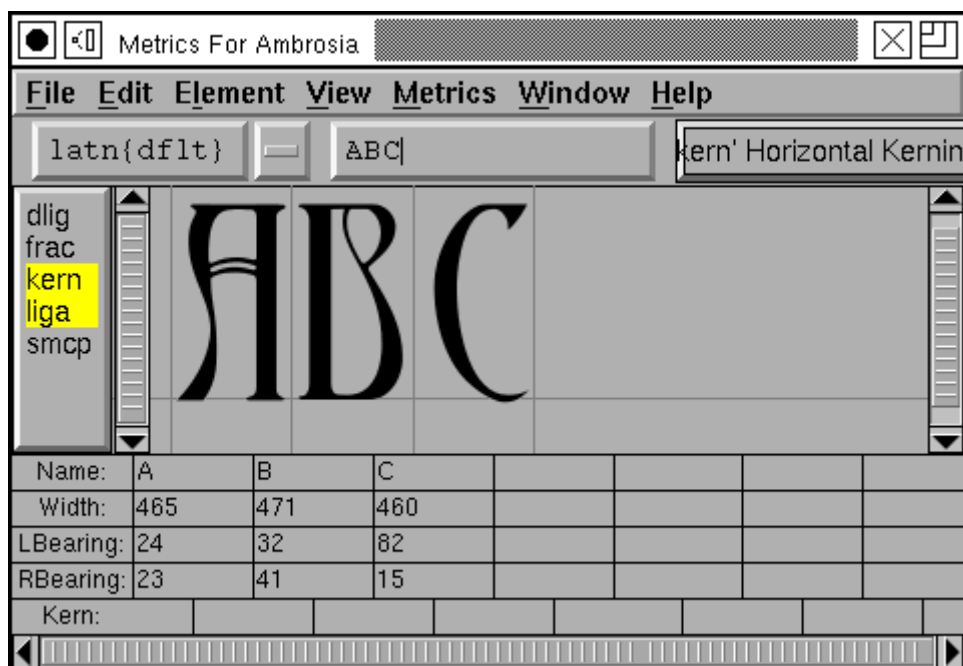


《2023 日本語版のメトリックウィンドウ》

以下の説明では、日本語版事例画像がないため、英語版の画像を表示しています。

「フォントビュー」〔フォント一覧〕内で選択されたグリフは（メトリックビューを呼び

出したとき)「メトリックビュー」の中に表示されます。表示されるグリフの並びは、新しくウィンドウの最上部のテキストフィールドに文字列をタイプするか、またはフォントビューからグリフをドラッグすることによって変更することができます。



この画面からグリフの「メトリック²²」を調整することができます。その下のテキストフィールドに設定値をタイプするか、グリフを(その画像をクリックして)選択し(左サイドベアリングを調整するために)位置を動かしたり(右サイドベアリングを調整するために)グリフの幅を表す線をドラッグしたりして行ないます。

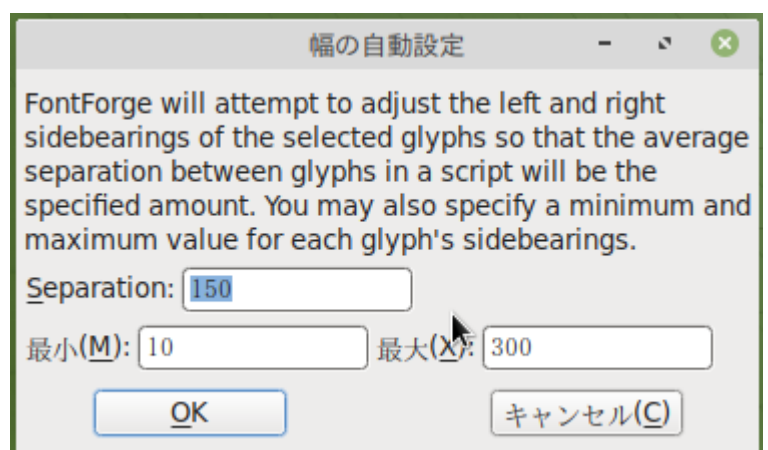
「タイプライタ」体のフォント(すべてのグリフの幅が同じ「等幅体」を生成したい場合、**【編集(E)】⇒【選択(S)】⇒【すべて選択(A)】**をフォントビューから実行し、次に**【メトリック(M)】⇒【幅を設定(W)...】**を実行します。これにより、すべてのグリフの幅が同じ値に設定されます。これを行なった後、**【メトリック(M)】⇒【幅の中央に(C)】**を実行して各グリフの左と右の空きを均等にしたほうがいいでしょう。

この手動動作が複雑すぎると思う人は、**【メトリック(M)】⇒【幅の自動設定(A)...】**を試してみてください。これは全自動で魔法のように各グリフの幅を設定します。この幅はプロの水準には達していませんが、一般的に見てまあまあ良いところを行っています。

〔《英文メッセージの内容》:

FontForge は選択された

グリフの左右のサイドベアリングを調整し、用字系でのグリフ間の平均的な間隔が指定された値になるようにします。両サイドベアリング値に対して、最小値と



22 メトリック (メトリクス) metrics: フォントデータ内にあらかじめ用意されている文字詰め情報を参照して、自動で文字詰めを行なう設定値。

最大値をしていすることもできます。

Separation : グリフ間距離

7.1.1. 縦書き用メトリック

FontForge は CJK フォントに必要な縦書きメトリックにある程度対応しています。まず、FontForge に、このフォントが縦書きメトリックを含むように指示する必要があります。これを行なうには **【エレメント(L)】 ⇒ 【フォント情報(F)...】 ⇒ 【一般情報】 ⇒ 【☐ 縦書きメトリックが存在(V)】** を使用します。それから「レイヤーパレット」で各アウトラインググリフの縦書きメトリックを有効にします。

これで、「縦書きの送り幅」を示す線がグリフの下の方に見えるはずです。この線を、「横書きの送り幅」の線と同様にドラッグすることができます。

7.1.2. フォントのベースライン間隔の設定方法

この、外観上重要な意味をもつ値を設定する簡単な方法があるとお考えになるかもしれませんが、残念ながら無いのです。

PostScript Type1 フォント（または裸の CFF フォント）における方法

この値を設定する方法はありません。

本当に、まったく無いのです。

伝統的な活版印刷では行と行の間隔は「1 em」です（これは、FontForge ではフォントの「高さ」＋「深さ」に相当します）。いくつかのアプリケーションはこの値を使用するでしょう。他のアプリケーションはフォントの「**バウンディングボックス**²³」（アセンダの最大高さ＋ディセンダの最小深さを足した数値）を使用するでしょう——非常に問題のある、しかしながら非常によく使われているアプローチです。

TrueType または OpenType フォントにおける方法

残念ながらこれはプラットフォームに依存します。

Mac

Mac では、ベースライン間隔は、やはり「hhea²⁴」テーブルで指定されているフォントの「バウンディングボックス」の値によって決まり、この値は「行間」（FontForge では **【エレメント(L)】 ⇒ 【フォント情報(F)...】 ⇒ 【OS/2】** タブにある「メトリック」内の「（hhea テーブルの）行間アキ」）で補正でき

23 bounding box : 像編集ソフトや文書作成ソフトなどで、操作・処理の対象となる図形・画像・文字列などの周囲を取り囲む矩形の境界線のこと。

24 hhea = horizontal header table : 水平レイアウト（横書き）でのヘッダーが情報が格納されているテーブル。

ます。

Windows では

OpenType の仕様書によると、ベースライン間の距離は「OS/2 テーブル」の「組版上の高さのオフセット Typographic Ascent」と「組版上の深さのオフセット Typographic Descent」の値によって設定されます。これらは **【エレメント(L)】 ⇒ 【フォント情報(F)...】 ⇒ 【OS/2】** によって設定できますが、一般には FontForge のデフォルト値のままにしておくことが広く行なわれています——この値の合計は一般に「1 em」になり、伝統的な行間アキ無しのときのデフォルトと等価です。

この値も行間 (linegap) フィールドで補正できます。

残念ながら Windows のプログラムはほとんど標準に従わず（今さら誰も驚かないとは思いますが）、一般にフォントの「バウンディングボックス」を「OS/2 テーブル」の「Win Ascent のオフセット」「Win Descent のオフセット」フィールドで指定されたものとして使用するでしょう。

Linux/Unix では

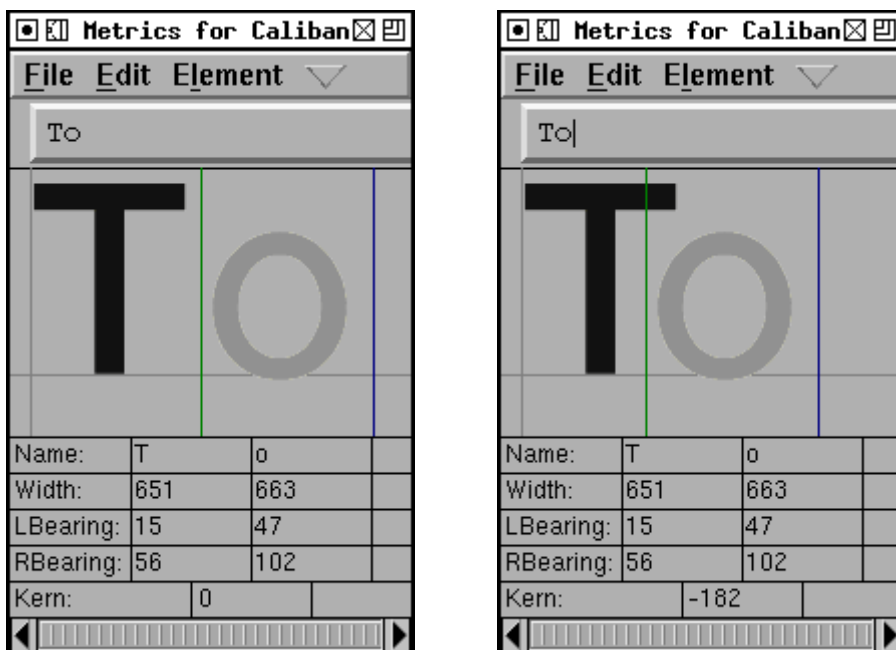
標準的なやり方が存在しているのか疑問に思います。Unix のアプリケーションはおそらく上記のどれかひとつを選ぶでしょう。

組版機能が豊富なアプリケーションは、ユーザーがベースライン間の間隔を調整できるようにしていますので、デフォルト値がどんな値でも無関係です。

7.2. カーニング

左右のサイドベアリングを注意深く設定していれば、ほとんどの場合、字間が美しく見えるようにフォントをデザインすることができるでしょう。しかし、常にいくつかは単純に解決できない場合が存在します。

ここでは「To」や「Av」などの標準的設定が不適切だった場合を考えましょう。文字「o」は、もっと左寄りにずらして「T」の横線〔トップバー〕の下に潜り込むようにしたほうが美しく見えるでしょう。これは「**カーニング kerning**」と呼ばれ、グリフ間の間隔を隣り合う文字同士に対して調節しています。



カーニング有無の比較

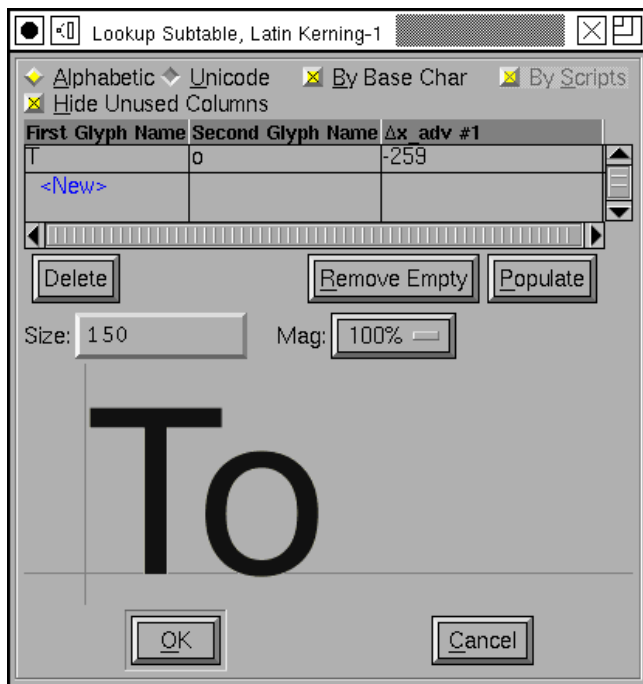
上の例では、左の画像は「カーニング無し」のテキストを、右は「カーニング有り」のテキストを示しています。カーニングのペアを作成するのは、簡単です。「メトリックウィンドウ」でペアとなる右のグリフをクリックすると、ふたつのグリフの間の線（通常は「横書きの送り幅の線」）が緑色になり（「カーニング時の送り幅線」となり）ます。この線を、文字間の隙間が美しく見えるようになるまでドラッグしてください。

悲しいことに、この説明は単純化しすぎました……。カーニング・ペアを作成する前に、「カーニング検索」を登録する必要があります（「6.3. 検索と機能」の章を参照）。もういちど、**[エレメント(L)] ⇒ [フォント情報(F)] ⇒ [Lookups]**（検索）ウィンドウを開き、**[GPOS]** タブを選択し、**[Add Lookup]** ボタンを押します。開いた「Lookup」ダイアログ・ウィンドウで、今回は検索の「種類」で「Pair Position (kerning)」を選び、すぐ下の「機能」欄では「kern 横書きカーニング」を指定します（もしかして、縦書きカーニング設定を行なう場合でしたら、「vkm 縦書きカーニング」を指定してください）。



カーニング用の「検索」を作成したら、それを再度選択し、**[Add Subtable]**（サブテーブルの追加）ボタンを押します（サブテーブルに名前を付けるよう求められます）。すると「Kerning Format」というサブ・ウィンドウが開き、FontForge は、「カーニング・ペア」または「カーニング・クラス」の、どちらのサブテーブルが必要かを尋ねます。

次に、FontForge はカーニング・サブテーブルを直接設定できるダイアログをポップアップ



プします。ここからカーニング・ペアを設定できますが、作者自身としては、より多くのグリフが見渡せ、フォントの「色合い」をよりよく把握できるので、メトリクスビューの方が好みます。

（ある種のグリフの組み合わせは、カーニングを設定するよりも合字を作成したほうが適切に扱えます。）

7.2.1. カーニングクラス

同じようなカーニング機能をもつグリフがたくさんある場合には、それらをひとまとめにした「カーニング・クラス」を作成したほうが楽でしょう（これにより「A」と

「À、Á、Â、Ã、Ä、Å」などがカーニングの上ですべて同じ物として扱われます）。

7.2.2. 縦書きカーニング

FontForge は縦書きカーニングについても横書き同様に対応しています。TrueType、OpenType および SVG フォントから縦書きカーニングを読み込んだり書き出したりすることができます。縦書きカーニングのクラスを作成することも可能です。メトリックウィンドウには縦書きカーニング・ペアを設定できる縦書きモードがあります。そして最後に、縦向きに回転したグリフに横書きのカーニング情報をコピーするコマンドがあります（つまり、「A」と「V」の組合せが横書きで「-200」だけ詰められている場合、縦書きグリフの「A.vert」と「V.vert」では「-200」だけ縦に詰められるということです）。

（縦書きカーニングは、フォントに縦書きメトリックが存在するときにかぎり利用可能です。）



8. 異体字とアンカーマーク

Glyph variants and anchoring marks

8.1. 異体字グリフ

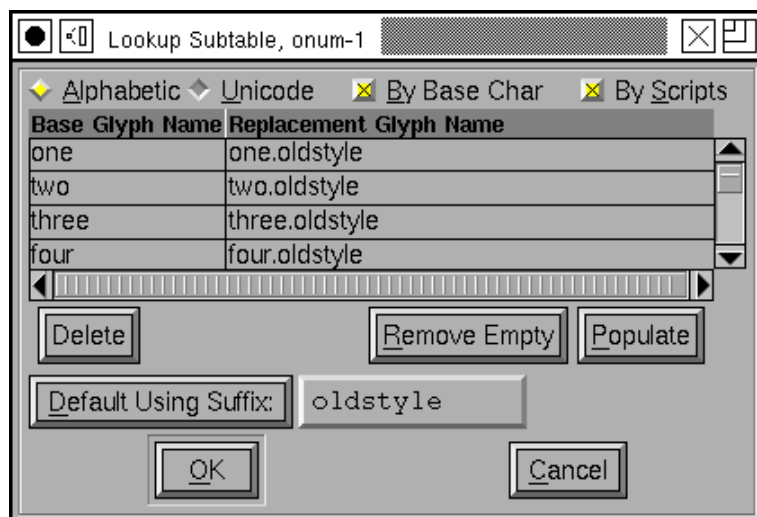
多くの用字系では、グリフがいくつかの異体字グリフ「字形が異なる同じ文字」を持つものがあります。ラテン文字では、最も明らかな例は、どの文字も大文字と小文字という異体字を含むということです。より秘教的な例としては、ルネッサンス時代の「長い s」という「s」の異体字が単語の頭と中間で用いられ、「短い s」は単語の終わりでしか用いられなかったというものがあります。

ほとんどのアラビア文字は四種の異なる形（語頭、語中、語尾および独立形）をもっています。

数字にはしばしば幾つかの異なる字形が設けられることがあります：「表用数字」（すべての数字が同じ送り幅をもっていて、表の数値が不揃いに見えない）、「プロポーションアル数字」（各数字が、その字形にふさわしい幅をもつ）と「オールドスタイル数字」あるいは「小文字数字」（123456789）と呼ばれる、ある数字にはディセNDERがあり、またある数字にはアセNDERがある字形のものです。

このような異体字のうちのいくつかは符号化された位置をもっています（大文字と小文字の区別はこれにあたります）が、その他の場合では、フォントに追加情報を与え、ワードプロセッサが異体字の存在を知ることができるようにする必要があります（アラビア文字はこのふたつの中間にあたり、多くの異体字形が符号化されてフォントに含まれていますが、追加情報も同時に与える必要があります）。

上記で言及した数字の場合を考えましょう。「one」という名前のグリフには表用の等幅数字の「1」が、「one.prop」というグリフにはプロポーションアルな異体字が、「one.oldstyle」にはオールドスタイル異体字が格納されていると仮定します。ここで、フォントビューから「one」を選択し、**[エレメント(L)] ⇒ [グリフ情報(G)...]** ^{※1}のダイアログで**[置換]**タブを開いて**[新規(N)...]** ^{※2}ボタンを押します。新しいダイアログが出てきますので、「構成要素」フィールドに「one.prop」と入力し、「プロポーションアル数字」を**[タグ(I)]**プルダウンリストから選択します（これは“pnum”となります）。それから興味のある用字系を選択します（用字系「latn」〔ラテン文字〕と言



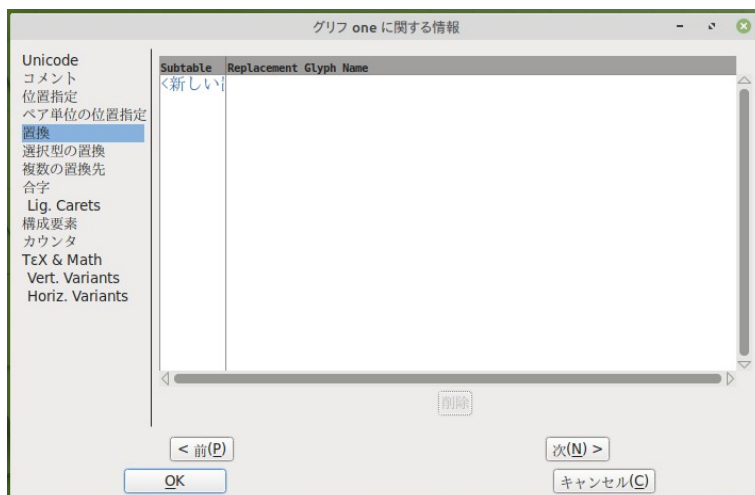
語「dflt」〔デフォルト〕は多くの場合に当てはまります)。最後に **[OK]** を押します。

FontForge が「オールドスタイル数字」の検索サブテーブルに入力するダイアログを表示すると、**[Default using Suffix:]** ボタン〔デフォルトに接尾辞を使用する〕の後に接尾辞部分を入力する「テキスト・ボックス」があることに注意してください。テキスト・ボックスを「oldstyle」に設定し、**[Default using Suffix:]** ボタンを押します。この「機能」設定で指定されたすべての文字のすべてのグリフを検索し、接尾辞に「oldstyle」持つ異体字を見つけてこのテーブルに追加します。

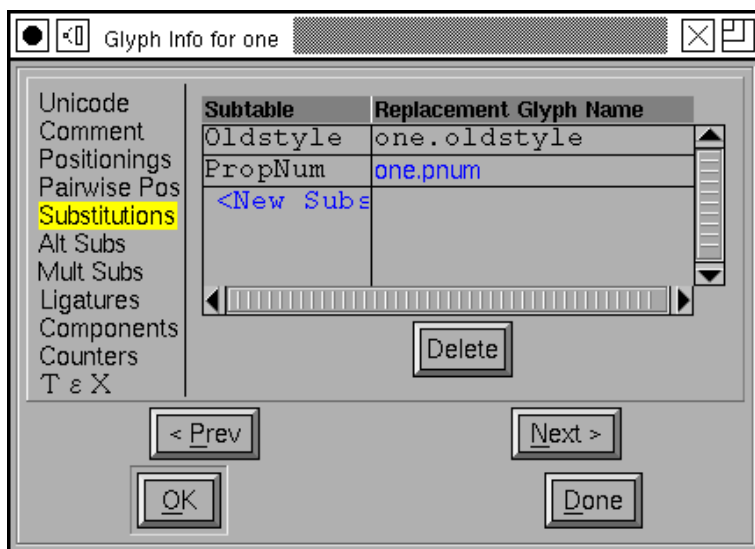
《※1 訳注》 右クリックで表示される「メニュー」では **[グリフ情報(I)...]** を選択します。

《※2 訳注》 202301 日本語版では、上記のようなダイアログが表示されないの、詳しい操作手順は不明です。

[置換] タブを選択すると右のウィンドウが表示され、「<新しい置換>」の部分に、置換に関する「検索」処理を記述するものと思われます（「**6.3. 検索と機能**」の章の手順を参考にしてください）。



時には、（特定の検索条件でのすべての置換よりも）特定のグリフで利用できるすべての置換を考える方が合理的な場合があります。したがって、サブテーブル・ダイアログで「プロポーショナルな数字」の置換設定を行なうのであれば、検索条件の入力を行なう代わりに、フォントビューで「1」を選択し、**[エレメント (L)]** ⇒ **[グリフ情報 (G)]** ⇒ **[置換]** タブを選択して表内の「<新しい置換>」部分を押します。



（【注意】 Type0、Type1 および Type3 PostScript フォントにはこれを取り扱うデータ表現がありません。このような異体字を出力したければ、OpenType か TrueType フォントを作成する必要があります。）

8.1.1. 条件つき異体字

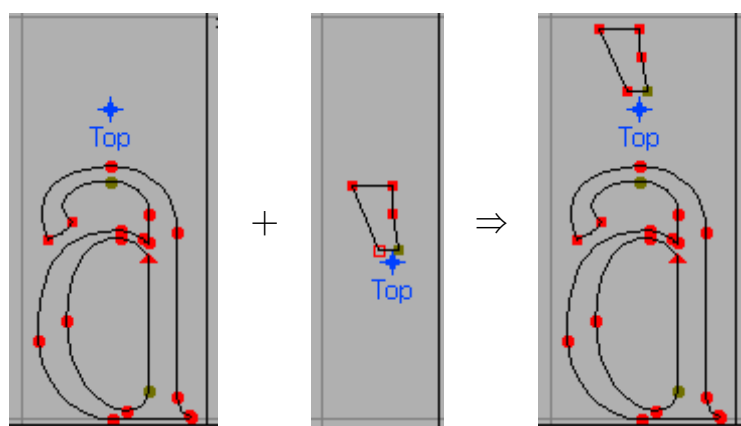
FontForge は OpenType の条件つき置換および連鎖的条件つき置換サブテーブルをサ

ポートしていますし、Apple の条件つきグリフ置換サブテーブルへのサポートも、一部機能のみの限定ですが、対応しています。これは、条件付きの異体字をフォントに含めることができるということを意味します。これについては 次章「9. 条件つき機能」にて詳しく解説します。

8.2. アンカーマーク

いくつかの用字系（アラビア文字、ヘブライ文字など）では、主要なテキスト・グリフの回りに「**母音記号 vowel marks**」を置く必要があります。別の用字系（いくつかのラテン文字やギリシア文字の異体字）では非常に多く種類の組み合わせ可能なアクセント記号があるので、すべての組み合わせをあらかじめ集めてグリフとして持っているのは非効率です。

OpenType フォントでは（ここにはマイクロソフト社の TrueType フォントも含まれます）、すべての基底文字上に記号が接続すべき位置を、すべての記号上に基底文字が接続すべき位置を、指定することができます。ですから、大文字「A」の上部中央に、すべてのアクセント記号（アキュート、グレーブ、ウムラウト、サーカムフレックス、リング、キャロン…）が接続すべき場所を指定するアンカーマークを、すべてアクセント記号の下にそれぞれ別のアンカーマークを置くことができます。これにより、ふたつのグリフがテキスト内で隣り合って現れたときに、ワードプロセッサは「A」の上に重なるアクセントがどこに位置するのか知ることができます。



すべてのアクセント記号が文字の上部中央に重なるわけではありません（「ドット」と「オゴネク」は文字の下に重ねられます）ので、種類の異なる「付属記号」に対してはアンカーマークをもう一つ定義する必要があるでしょう。

それに加え、幾つかの文字——例えば Unicode「U+1EA4」はサーカムフレックスとアキュートというアクセント記号がひとつずつ付いた「A」ですが——は、複数の「付属記号」をもつことができます。通常では、サーカムフレックスとアキュートは同じ位置に接続されてしまい、醜悪で見づらいことになるでしょう。その代わりに、異なる種類のアンカーマーク（「記号から記号へのアンカー」）をサーカムフレックス上に作成し、アキュートがサーカムフレックスに接続できるようにします。

Ã

U+1EA4

まだグリフにアンカーマークを作成できるようにする前に、「検索」と「サブテーブル」

を（言うまでもなく）作成しなければなりません。これは別の「グリフ位置指定検索」（GPOS ウィンドウに入力します）です。「サブテーブル」を作成すると、「アンカー・クラス」のためのダイアログが開きますので、「付属記号」の種類ごとにそれぞれ「**アンカー・クラス**」を作成する必要があります。したがって、二種類の「付属記号」（ひとつは上に、ひとつは下に）を持つ上記の「A」の場合、ふたつのアンカー・クラスを作成することになります。²⁵

それから、〔アウトラインウィンドウで〕付属記号の配置点を作成するそれぞれにおいて、まずアンカー・マークを配置する位置をクリックし、次に **〔点(P)] ⇒ [アンカーを追加(A)...]** ダイアログを呼び出す必要があります。

〔表示(V)] ⇒ [アンカーの制御(A)...] ²⁶ コマンドを使えば、そのグリフが他のグリフと組み合わせられるときにどのように組み合わせられるのかを確認することができます。

《警告》 付属記号の配置に関する警告： すべてのソフトウェアがこのような付属記号の配置方法に対応しているわけではありません。さらに紛らわしいことに、ソフトウェア側で一部の用字系には対応し、別の用字系には対応していないこともあります。



25 《旧日本語版には以下の追記あり》 「この作業は **〔エレメント(L)] ⇒ [フォント情報(F)...] ⇒ [アンカー・クラス]** ダイアログによって行なえます。

26 旧日本語版では「 **〔メトリック(M)] ⇒ [アンカーの制御(A)...]** 」、英語版（旧日本語版よりも内容があたらしい）では「 **〔表示(V)] ⇒ [アンカーの制御(A)...]** 」となっています、FontForge 20230101 日本語版のメニューではこのメニュー・コマンドは確認できていません。

9. 条件つき機能

Conditional Features

OpenType と Apple のフォントのどちらでも、条件付きの機能を使用することができます。「**条件付き機能**」とは、所定の文脈下でのみ実行される機能（contextual features）であり、インド系およびアラビア系の用字系の組版においては必須のものです。OpenType では、文脈は、文書の連続するグリフに対して照合を行なう文字パターンの集まりとして指定されます。あるパターンが合致すると、そのパターンに定義された「置換」を適用します。Apple のフォントでは、文脈は「**状態機械 a state machine**」——グリフの並びを解析して変換する小さなプログラム——として指定されます。

「条件つき機能」には「置換」、「合字」、あるいは「カーニング」（さらには「不明瞭なふるまい」）を含めることができます。ここでは、最初に「**文脈依存の置換 contextual substitution**」の一例を示し、次に「文脈依存の合字」の例を示します。

インド系文字やアラビア文字の例を取り上げる代わりに、筆者がより慣れ親しんでいるものを例に取り上げることにしましょう。ラテン文字の筆記体フォントを組版する問題です。このフォントは、



「b」「o」「v」と「w」が次の文字と「エックスハイト」のあたりで接続し、その他の文字はすべてベースラインのそばで接続します。この場合、各グリフにつき、ベースラインのあたり（の左側）に接続線があるデフォルトの字形と、エックスハイトのところに接続線がある別字形との二種類の異字形が必要となります。このふたつ目の接続線が高い位置にある文字セットを「高い」文字と呼び、その文字を「a.high」、「b.high」…といった具合に名付けることにしましょう。

9.1. OpenType の例

【著者による警告文】 以下に挙げる例は動作しないかもしれません！ この手順によって生成されるフォントテーブルは完全に正しいのですが、〔OpenType の設計者たちは、ラテン文字では複雑な条件つき機能を必要ではないと決定したので、²⁷⁾ OpenType の初期の実装ではラテン筆記体文字に対する「文脈依存の文字代替 Contextual Alternative」機能（「calt」²⁸⁾）に対応していないのです。

27 〔 〕内は旧日本語版の内容。英語版の記述では「... OpenType did not support decided the “Contextual Alternative” ...」と不自然な decided が残っているため、何らかの説明文が削除されたのか脱落しているのか判断できないため、旧日本語版内容を補記しておきます。

28 calt = **C**ontextual **A**lternative の略。単なる合字置換ではなく、文脈に依存した前後関係に依存する置換であることから、「文脈依存の代替字形」と呼ばれるようです。

では、なぜ、私は動作しない例を提示しているのでしょうか？ それが私にできる最善の事だからです。もし私がインド系文字やアラビア文字の組版に通暁していたなら、それらの用字系の例を挙げていたことでしょう。しかし私はその素養を欠いています。手順は同じです。他の何らかの用字系でそれに従って作業すれば、動く物ができるでしょう。

時間が経つにつれて、ラテン文字用のより多くのアプリケーションで「calt（文脈依存の文字代替）」機能に対応するようになり、この警告文は、もはや嘗てほどには当て嵌まりません。

グリフを三つのクラスに分けます：「bovw の 4 文字」、「それ以外のすべての英字」、そして「英字以外のすべてのグリフ」です。ふたつのパターンを作成する必要があります。第 1 のパターンは「bovw」クラスに含まれるグリフの後に「bovw」クラスのグリフが続く場合で、第 2 のパターンは「bovw」クラスの後にそれ以外の英字が続く場合です。このふたつのパターンのどちらかに一致するときには、後に続く文字のグリフを「高い」異字形に変換する必要があります。

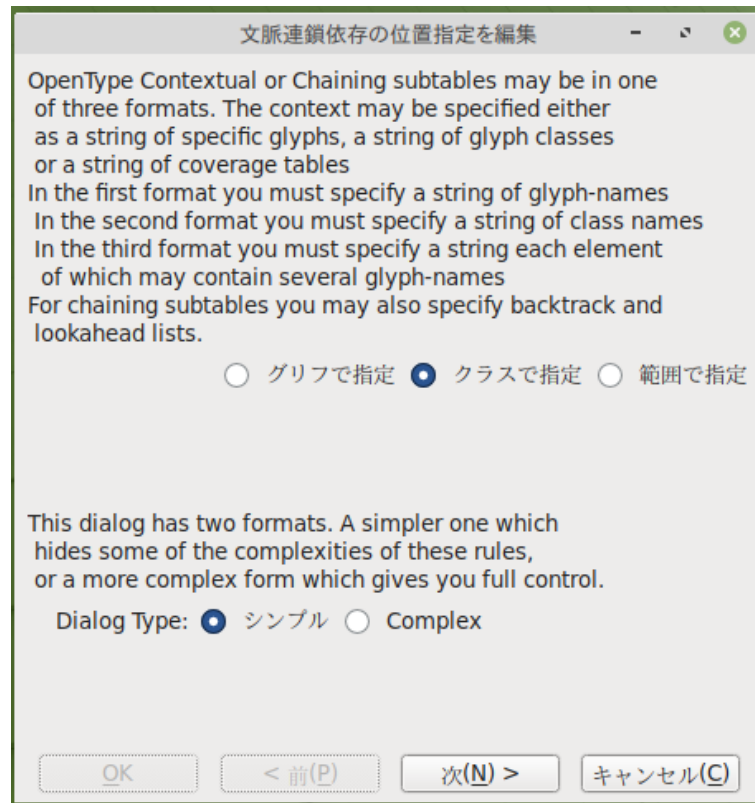
- [bovw] [bovw] の並び ⇒ 二番目の文字に「置換」を適用
- [bovw] <bovw 以外の文字> の並び ⇒ 二番目の文字に「置換」を適用

（なぜ「すべての文字」のクラスだけを用意し、ふたつのルールではなくひとつのルールを使用しないのかと疑問に思われるかもしれません。そのようにした場合、すべてのクラスは互いに素になるはずなので、ひとつのグリフがふたつのクラスに表示されない可能性があります。）

最初に行なう必要があるのは、各々の「低い」文字に「高い」異字形を対応づけるだけの単純な置換を作成する必要があります。これが「**単純置換**」検索ですが、どの「機能」とも結び付けられておらず、代わりに、文脈に基づいた検索により呼び出されます。この「検索」を「high」と呼ぶことにしましょう。（もちろん）「検索」時に用いられる「サブテーブル」も作成しなければなりません。〔8.1.項と同じように〕**[Default using Suffix:]** ボタンを使用して、「高い」異体字を入力できます。

難しいのは文脈の定義です。これは、別の「検索」、つまり「calt」機能に関連している「**文脈連鎖検索**」を定義することによって行なわれます（もちろん、検索にもちられる「サブテーブル」も必要です）。この作業では「文脈依存の置換」を編集するダイアログがいくつか表示されます。

次のダイアログ「文脈連鎖依存の位置指定を編集」では、「置換」の全体的なフォーマットを指定することができます。使いたいのは「クラス」ベースのシステムです——「グリフ」クラスがどのようなものかは、既に述べたとおりです。



《説明文の内容》

OpenType 「文脈依存または連鎖サブテーブル」は三つのフォーマットのいずれかひとつです。文脈は、「グリフ」の連続、「グリフ・クラス」の連続、または「対象テーブル」の連続として指定できます。

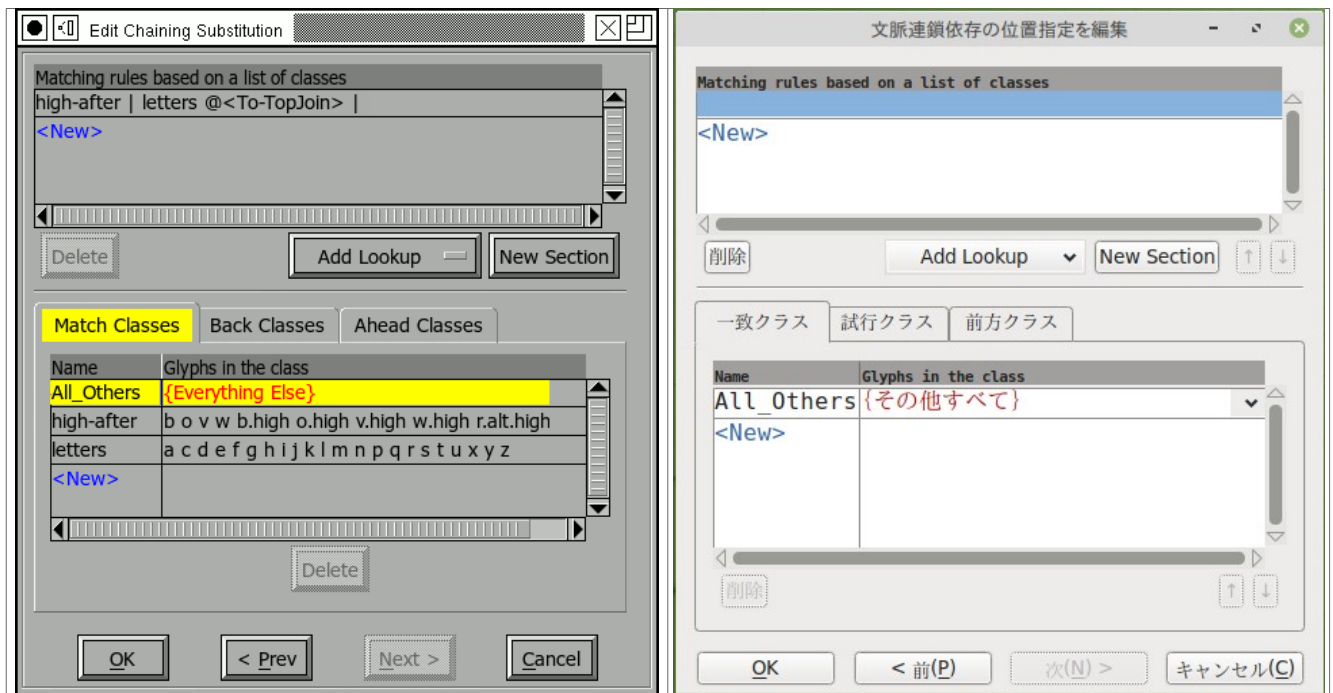
「第一のフォーマット」では「グリフ名」を、「第二のフォーマット」では「クラス名」を、「第三のフォーマット」ではグリフ名をいくつか含む「各要素の文字列」を指定する必要があります。

また、どの「連鎖サブテーブル」にも、「後戻りリスト」「先読みリスト」を指定します。
【フォーマット指定用選択肢あり】

このダイアログには二つのフォーマットがあります。「シンプル」フォーマットはルール複雑な部分をいくらか非表示にし、「Complex」フォーマットでは全ルールを完全に制御可能になります。 【フォーマット指定用選択肢あり】

その次のダイアログ〔下図参照〕でやっと、興味をひく情報が表示されます。いちばん上の枠には照合するパターンと、文字列が一致したときに行なわれる「置換」が並べて表示されています。

下側の枠にあるのは、その置換が利用する「グリフ・クラス」です。「文脈連鎖依存」のダイアログは、グリフの並びを三つのカテゴリーに分割しています： 1) 現グリフの前にあるグリフ群（これらは「**後戻りグリフ**」と呼ばれます）、2) 現グリフそのもの（これは二文字以上でもよく、そのグリフに対して適用される「単純置換」を指定可能です）、3) 現グリフの後のグリフ（これらは「**先読みグリフ**」と呼ばれます）、の三つです。



20230101 日本語版 画面（ただし設定前）

各グリフ・カテゴリはグリフをそれぞれ異なるクラス・セットに分類することができますが、この例ではすべてのカテゴリで同じクラスを用いることにします（これにより、置換処理を Apple のフォーマットに変換するのがやりやすくなります）。最初の行「ウィンドウ上側の」（「Matching rules ...」と書かれている行）は、以下のように読みます：「もし、クラス「high-after」に含まれる「後戻りグリフ」（現グリフの直前のグリフ）のあとに続く現グリフがクラス「letter」に含まれる文字であれば、一致した文字列（これは現グリフのことです）中の位置には、単純置換の「To-Top.Join」〔「高い位置へ接続」＝「high」〕を適用しなければならない」。

グリフ・クラス定義「ウィンドウ下側の枠にある「Glyphs in the class」の欄」を見ると、クラス「high-afters」には「高い」異字形の後にくるグリフがあることが分かるので、この定義には納得がいきます。

では、「high-after | high-after @<To-TopJoin> |」と書かれた二行目ですが、これは前述の二番目のケースを処理するためのものです。残念ながら、これはうまくいかないようです。OpenType の仕様では許可されていますが、OpenType の実装ではサポートされていないようです。代わりに、この行は同じ「検索」内の「別のサブテーブル」に存在する必要があります。

グリフ・クラスの編集は、その行をダブルクリックし入力するだけです。新しく作成するには、（クラスリストの下にある）青字で表示された < New > の部分を押してください。右側にある小さなボックスみたいなものをクリックすることもできます。それを押すと、「フォント一覧ビュー」が表示されますので、グリフの名前を覚えるよりも簡単であれば、グリフを選択して入力できます。

パターンの編集「ウィンドウ上側の枠」は明白です。グリフ名を入力すると、自動的に入力されます。パターン・リストの下部にはふたつのボタンがありますが、そのひとつ **[New Section]** は新しいセクションを開始するため（「後戻り back-tracking」からアクティブ

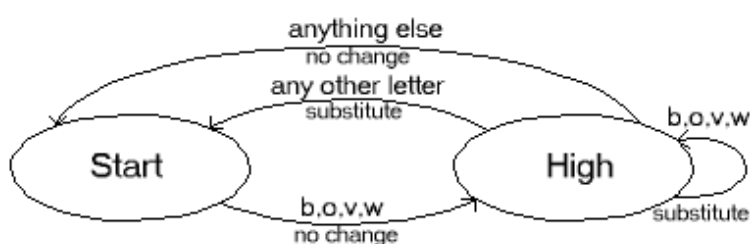
へ、またはアクティブから「先読み lookahead」へ）、もうひとつ **[Add Lookup]** は新しい検索参照を追加します。

【警告】 文脈依存のふるまいに関する警告： すべてのソフトウェアがこれに対応しているわけではありません。さらに紛らわしいことに、ソフトウェア側で一部の用字系には対応し、別の用字系には対応していないこともあります。

9.2. Apple 高度組版機能

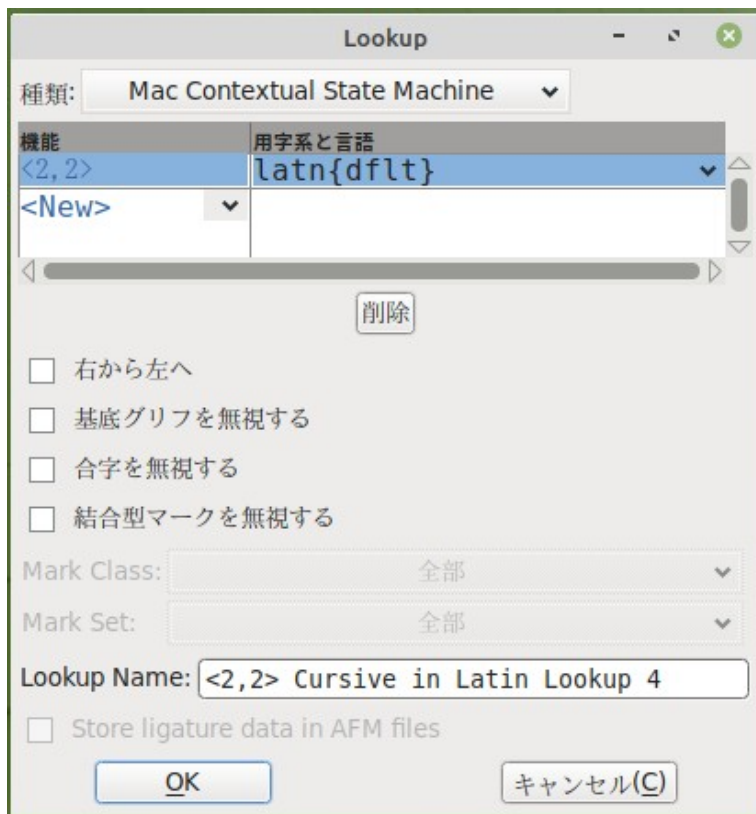
Apple は「有限状態機械²⁹」で文脈を指定しています。これは、本質的にはグリフの並びを調べて、どの置換を適用するかを決定する小さなプログラムです。各「状態機械」には、（OpenType の例で示したのと同じような）グリフ・クラスの定義の集合と「状態」の集合があります。処理は、グリフの並びの先頭から、「状態 0」で始まります。コンピュータは、現在のグリフがどのクラスに含まれるかを決定し、次に現在の状態を参照して、そのクラスからの入力を与えられたときにどのように振舞うかを調べます。この振舞い〔処理内容〕の中には、「別の状態への遷移可能性」、「次のグリフへ処理を進めること」、「置換処理を、現在のグリフに適用するか、それとも前のグリフ（“記号つき”グリフ）に適用するか」などが含まれます。

前と同じ、ラテン文字の筆記体フォントの例を使うことにしましょう……。今回も各文字をその「高い」異体字に置換する「単純置換」が必要となります。この処理は OpenType で行なったのと同じで、実際に同じ置換処理を使用すること



ができます。今回もグリフを三つのクラスに分類します（Apple は、使用するかしないかにかかわらずいくつかの追加クラスを定義していますが、概念上 OpenType の例と同じ三つのクラスを使用することにします）。この例では、状態機械には二つの「状態」が必要で（こちら、Apple により追加の「状態」が無償で定義済みですが、それも無視しましょう）、ひとつは「開始状態」（基本状態——変更は行なわれない状態です）、もう一つは「bovw」クラスからグリフを読み込んだ「直後の状態」です。

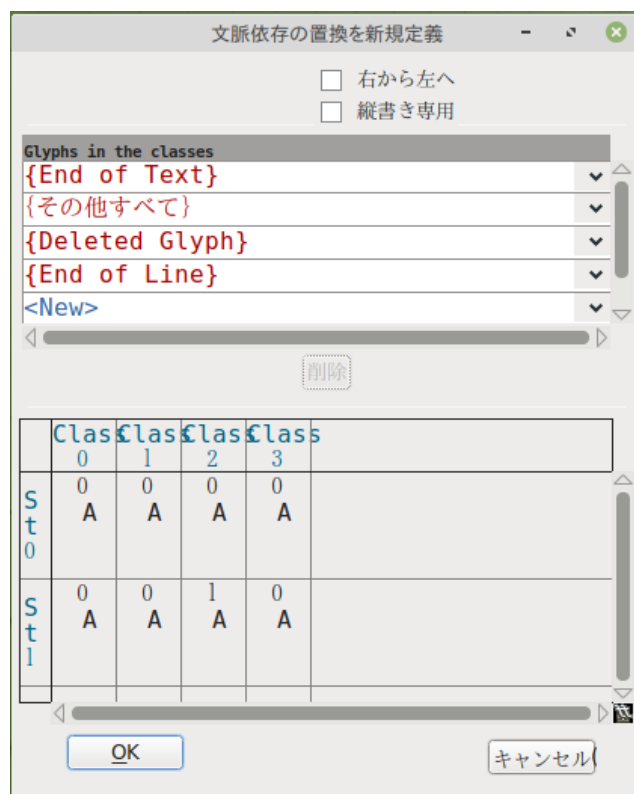
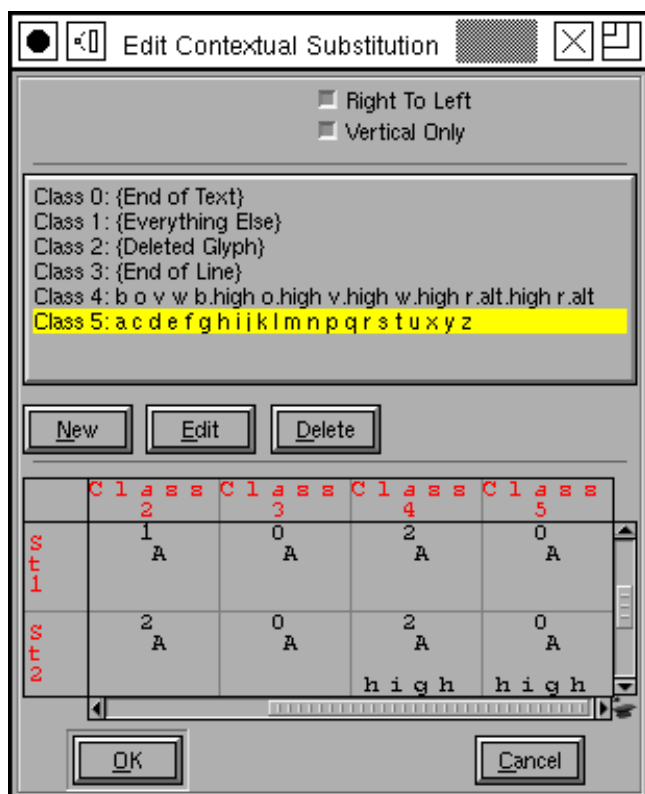
²⁹ 状態機械： state machine の訳。「状態機械」とは、操作対象の「状態」を定義して、それによって処理内容（振舞い）を切り替える〔状態が遷移する〕仕組みの論理回路（プログラム）です。「オートマトン」とも呼ばれています。「有限状態機械」では有限個の状態を定義し、その個々の状態に応じた処理結果が出力されます。



「Apple 高度組版機能」は、OpenType の「検索」と「機能」の概念に完全には適合していませんが、かなり似ているので、無理やり適合させます。そこで、もう一度「GSUB 検索」を作成します。今回の「検索種類」は「Mac Contextual State Machine」で、「機能」は実際には Mac の機能／設定、つまり「ふたつの数字」です。この新しいサブテーブルが作成されると、次のような「状況機械」のダイアログ・ウィンドウが開きます。

ダイアログのいちばん上には、状態機械の機能/設定を指定するフィールドが表示されています。これは Apple の方式では OpenType の四文字の機能タグと等価です。その下に

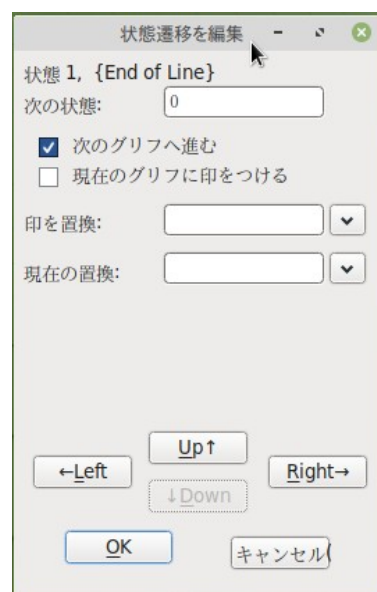
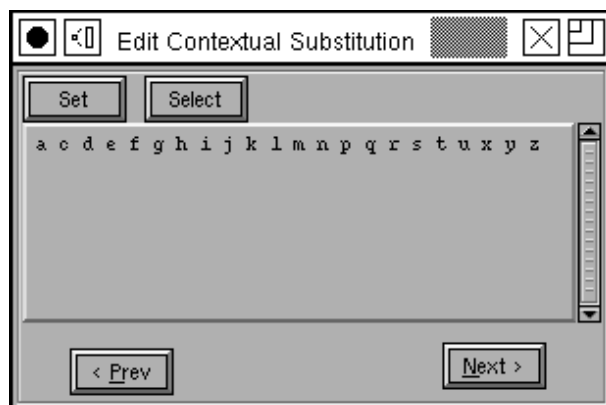
あるのはクラス定義のセットで、いちばん下にあるのは状態機械そのものです。



《参考》 20230101 日本語版画像

クラスの上でダブルクリックすると、OpenTypeで使用しているのと似たダイアログが立ち上がります。

状態機械の遷移〔変化推移〕の直交表（「状態 St0、St1、...」と「クラス 0、1、2、...」の組合せごとにひと組みの遷移があります）の上でクリックすると、状態遷移ダイアログが立ち上がります。指定された状態において指定されたクラスに含まれるグリフを受け付けたときにどのように振舞うかを、ここで制御します。



《参考》 20230101 日本語版画像

この例〔左図〕では、「状態 2（St2）」のときに（これは、「bovw」グリフを読み込み済みであることを意味します）、「クラス 4」に含まれるグリフを受け取ったところです（これは「bovw」のグリフが続けてもうひとつ来たことを意味します）。この場合、次の状態も再び「状態 2」なので（新しい「bovw」グリフを読み込んだ直後の状態）、もう一字を読み込んで「high」置換をそのグリフに施すことになります。

ダイアログのいちばん下にあるのは、状態機械の遷移を操作できるボタンです。

[OK] を何回か押していくつかのウィンドウを閉じると、このダイアログの連鎖から抜け、フォントに新しい状態機械を追加するウィンドウに戻ります。

9.3. オープンタイプ、ギリシア文字の合字

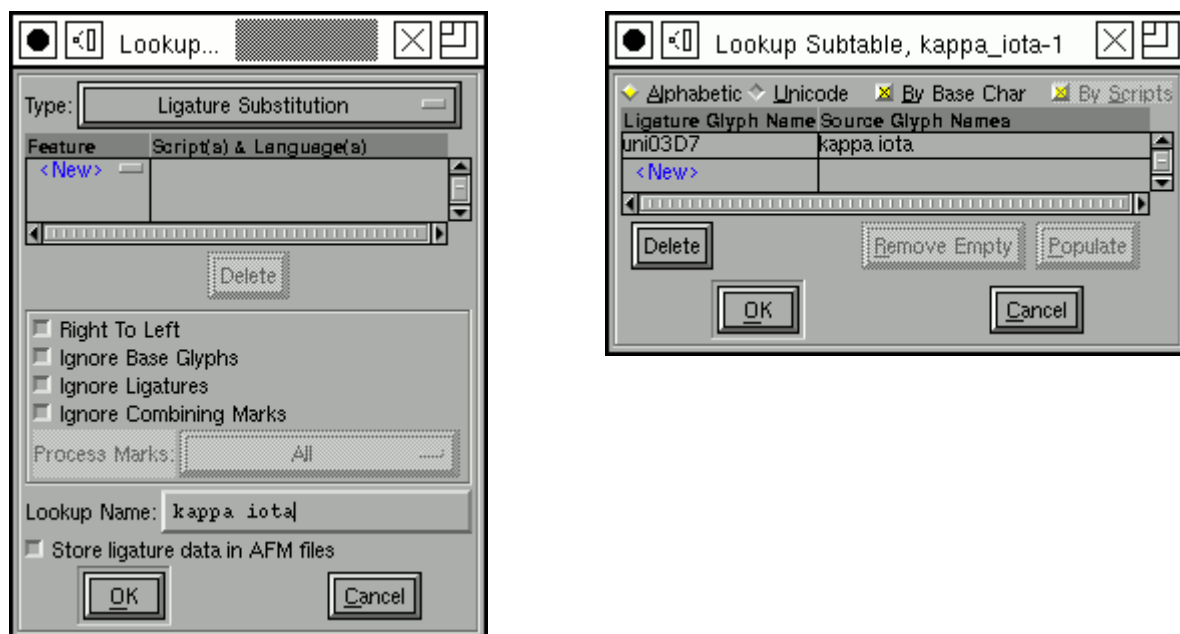
ギリシア語には、ラテン文字のアンパサンド「&」に相当する文字（U+03D7）があります。アンパサンドが（本来は）「E」と「t」の合字であるのと同じように、「U+03D7」は「kappa（カッパ）」と「iota（イオタ）」の合字です。ただし、この合字は、「カッパ」と「イオタ」が単独で単語を構成する

ⲥ

U+03D7

場合にのみ使用する必要があり、長い単語内でこのふたつが出現する通常の場合には使用しないでください。

したがって、最初に行なうことは、合字自体を作成することです。「U+03D7」のグリフを追加し、合字検索とサブテーブルを作成して（**[エレメント(L)]** ⇒ **[フォント情報(F)]** ⇒ **[Lookups (検索)]** タブを用いて）、「U+03D7」を「kappa」と「iota」の合字であるということに結び付けます。この「検索」は直接使用されることはなく、別の条件付き機能の制御下でのみ使用されるため、機能タグは付けません。



次に「判定条件」の部分です。

すべての文字がギリシア文字で構成されるクラスを表すために「< letters >」という表記を用いることにします。

- < letters > kappa iota の並び ⇒ 置換なし
- kappa iota < letters > の並び ⇒ 置換なし
- kappa iota の組み合わせ ⇒ 合字「語」を適用します

（さて、著者が標準を読んだところ、このすべてのルールはひとつのサブテーブルに入れることができるようで、私が持っているフォント検証ツールも「ひとつのサブテーブル方式」に同意していますが、どのレイアウト・エンジンも対応しません。レイアウト・エンジンは、各ルールが独自のサブテーブルに存在することを主張しているようです。これはクラスは各サブテーブルで定義する必要があり不便ですが、機能するようです。³⁰⁾

これらのルールは順番に実行され、入力テキストに一致した最初のルールが（唯一の）ルールとして適用されます。次の三つの文字列（ $\alpha\kappa\iota$ 、 $\kappa\iota\theta$ 、 $\alpha\kappa\iota\theta$ ）にはすべて「カッパ」と「イオタ」が含まれていますが、それぞれの周囲にさらに他の文字が含まれているため、合字で置き換えるべきものではありません。

- 最初の文字列 $\alpha\kappa\iota$ は、上記の最初のルール（「カッパ」「イオタ」二文字の前に文字

30 《※ 訳注》このコメントは 2010 年前後の状況と思われますが、2024 年現在の状況は不明です。

がある) に一致するので、合字への置換は行なわれません。これは三番目のルールとも一致しますが、そこまでは到達しません。

- 二番目の文字列 $\kappa\iota$ は、上記の二番目のルール(「カッパ」「イオタ」二文字の後ろに文字が含まれる)と一致し、ここでも置換は行なわれません。これも三番目のルールと一致しますが、最初の一致で停止します。
- 三番目の文字列 $\alpha\kappa\iota\theta$ はすべてのルールに一致しますが、検索は最初の一致で停止するため、最初のルールのみが適用され、置換は行なわれません。
- 文字列 $_\kappa\iota_$ は最初の 2 つのルールのどちらとも一致しませんが、最後のルールには一致するため、ここで合字が形成されます。

なぜルールをひとつだけにしないのかと不思議に思うかもしれません。たとえば、

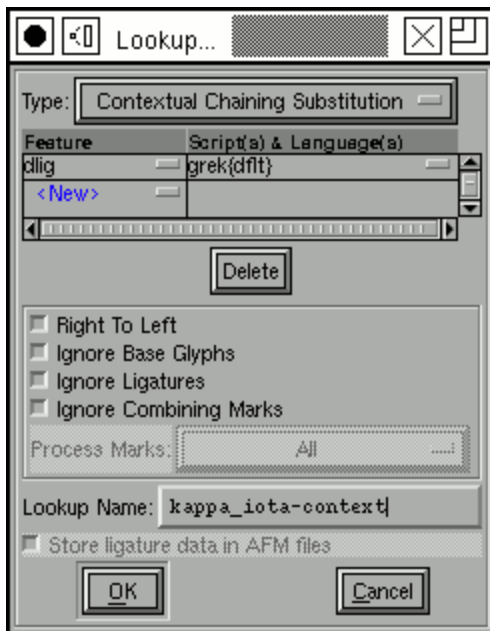
1. < letters 登録以外の任意の文字 > kappa iota < letters 登録以外の任意の文字 >
⇒ 合字を適用

これなら、ずっと簡単に思えます。

しかし、主な理由がひとつあります。

1. このルールは、文字「カッパ」が入力文字列の先頭にある場合(つまり、その前にはグリフはありませんが、合字置換が必要な場合があります)、または文字「イオタ」が文字列の末尾にある場合には機能しません。

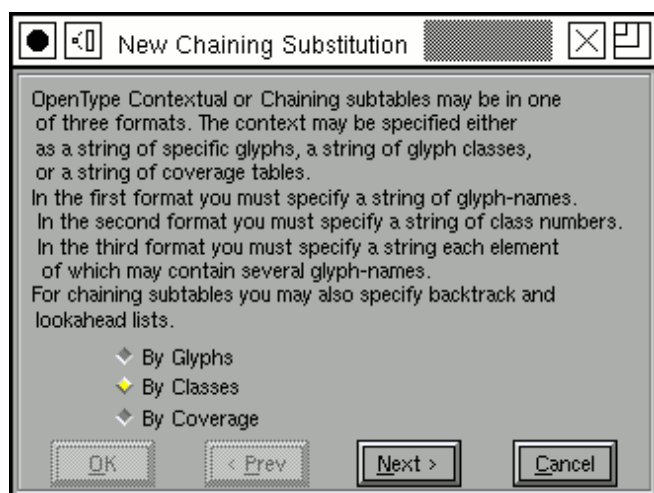
では、このルールを文脈検索に変換するにはどうすればよいのでしょうか？



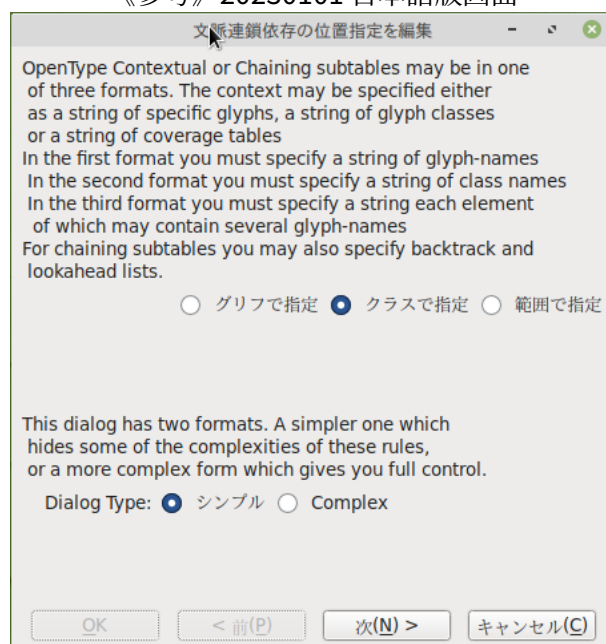
[エレメント (L)] ⇒ [フォント情報 (F)] ⇒ [Lookups (検索)] ⇒ [Add Lookup (検索を追加)] を使用して、新しい「文脈連鎖検索」を作成します。これは最上位の検索であり、ギリシア文字の機能タグに結びつける必要があります。

〔上記のように〕ルールは三つあり、各ルールは独自のサブテーブル内に存在するため、それぞれにひとつずつ、三つのサブテーブルを作成します。「検索」ウィンドウ内に記述されるこれらのサブテーブルの順序は、サブテーブルに含まれるルールが実行される順序であるため重要です。実際に合字を呼び出す最終ルールが最後に実行されるルール(おおよびリストの最後)であることを保証する必要があります。

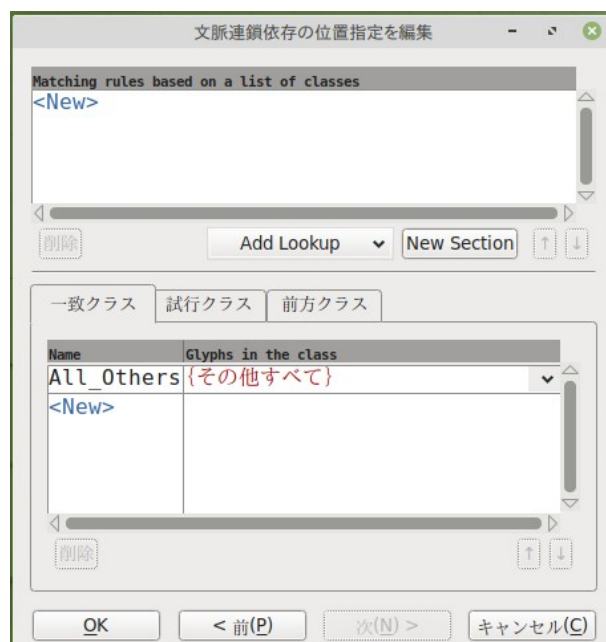
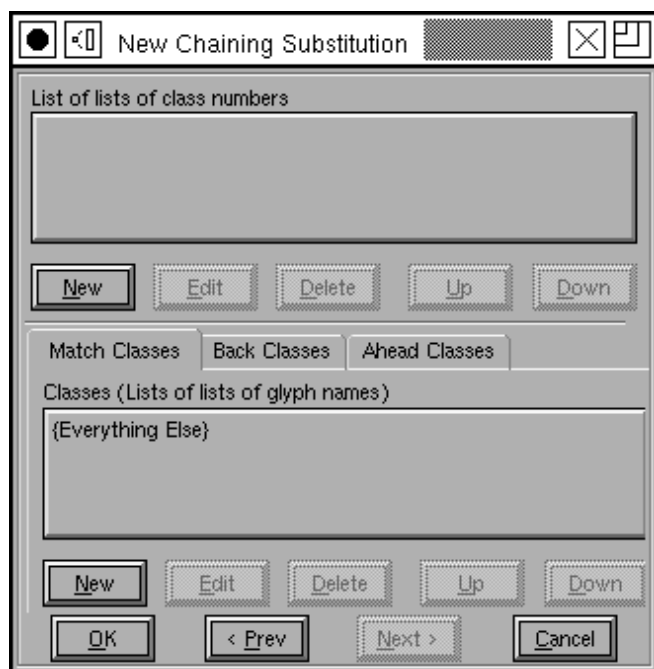
手順



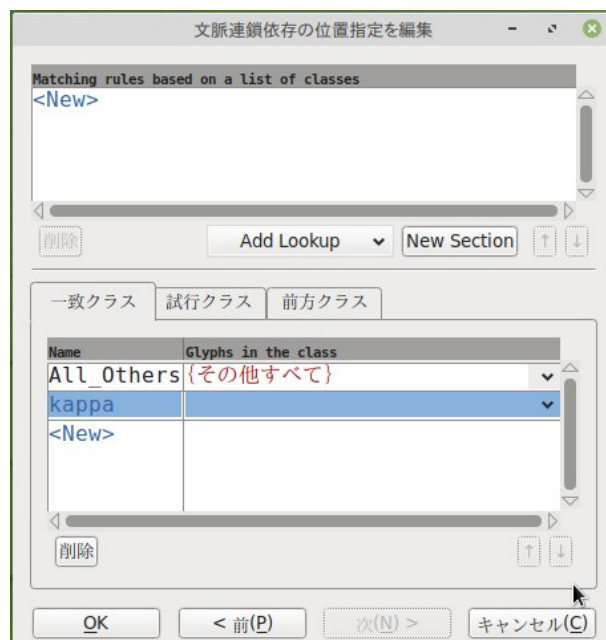
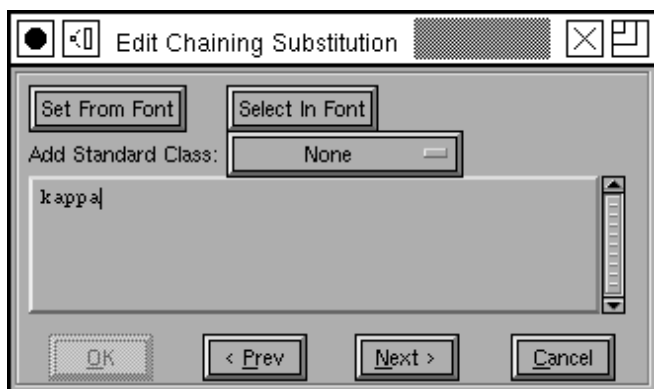
《参考》20230101 日本語版画面



すべてのギリシア文字のクラスを使用する予定なので、この「機能」にはクラス形式を使用する必要があります。[**Next / 次(N) >**] ボタンを押します。



主な一致は三つのルールすべてで、文字「カッパ」と「イオタ」に対して行なわれるため、それぞれにひとつのクラスが必要です。そこで、「一致クラス / Match Classes」タブ内で「< New >」ボタン〔2023 版では <New> の青文字部分〕を押します…。

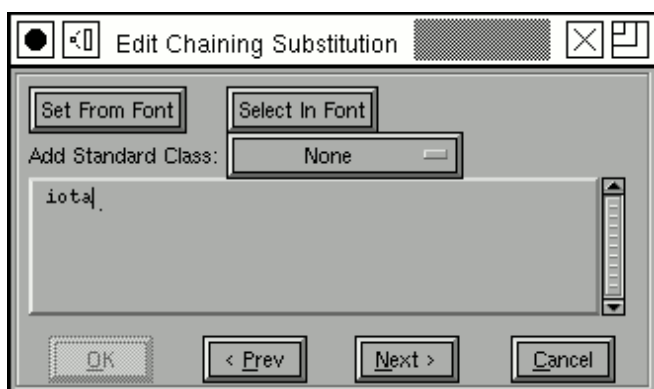


「kappa」（カッパ）と入力し、**[Next >]** ボタンを押します。〔※ 20230101 版画面では、追加された行に入力し、**[OK]** ボタンを押します。〕

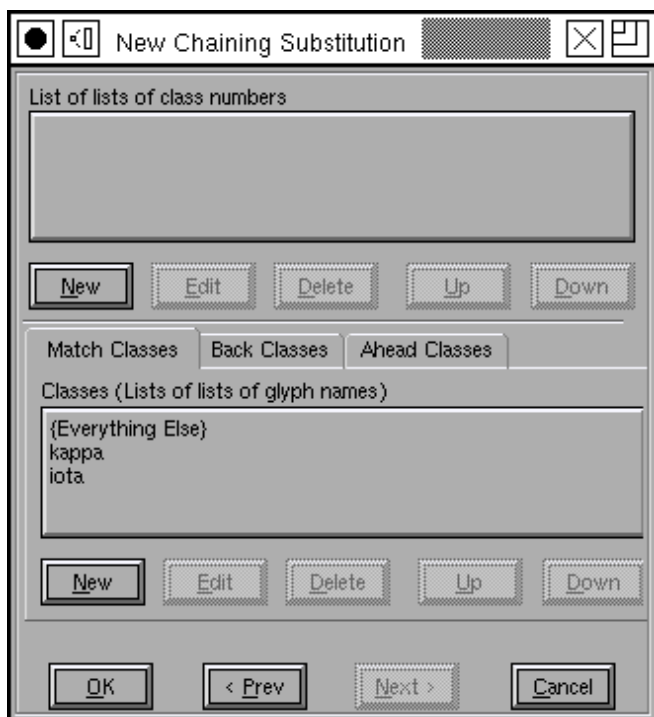


これで、単一のグリフ「kappa」を含むクラスができました。「iota」についても同じ操作を行なうので、もう一度 **[New]** を押します。

※ 以下、上記と同様の画像のため、もしくは同等内容の画像例が日本語版で作成できない（フォントサンプルがない）ため、英語版画像のみでの説明になります。



再度「iota」と入力し、**[Next >]** を押します。



これで、ここで必要な「クラス」がすべて揃いました。しかし「先読み」（look-ahead）と「後戻り」（backtrack）のためのクラスを作成する必要があります。このグループにはクラスがひとつだけ必要で、そのクラスはすべてのギリシア文字で構成されます。



「Back Class」タブを開きます（日本語版ウィンドウでは「試行クラス³¹」となっています）。

チェックボックス「☐「一致クラス」と同じ」にチェックが入っていますが、ここでは独自のクラスが必要で、この設定は望ましくないなので、チェックを外してください。

31 「試行クラス」： Back Class の訳語であるが、「後戻り検索 backtrack」用のクラス定義なので、「遡行（＝後戻り）」の誤記と思われます。

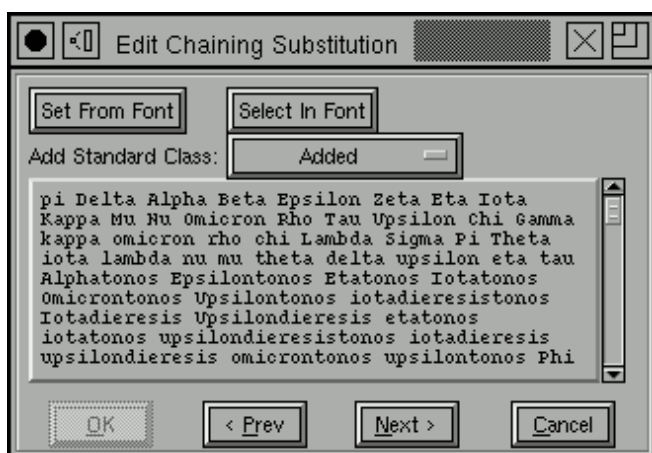


すると「入力エリア」がアクティブになり、**[New]** を押せば新しいクラスを作成できるようになります。

〔※ 20230101 版画面では **< New >** と書かれた部分をクリックすると、新しい入力行が現れ、入力行の右端のボタンをクリックすると「フォントビュー」が表示されます。〕

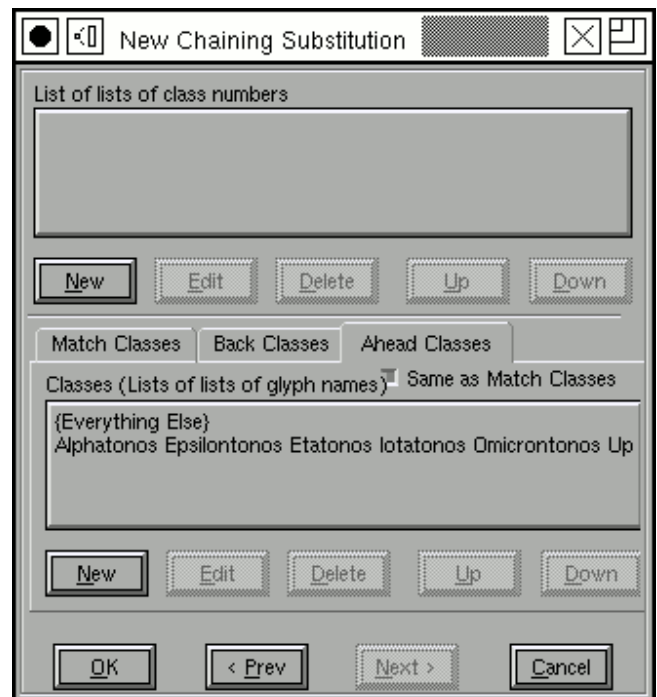
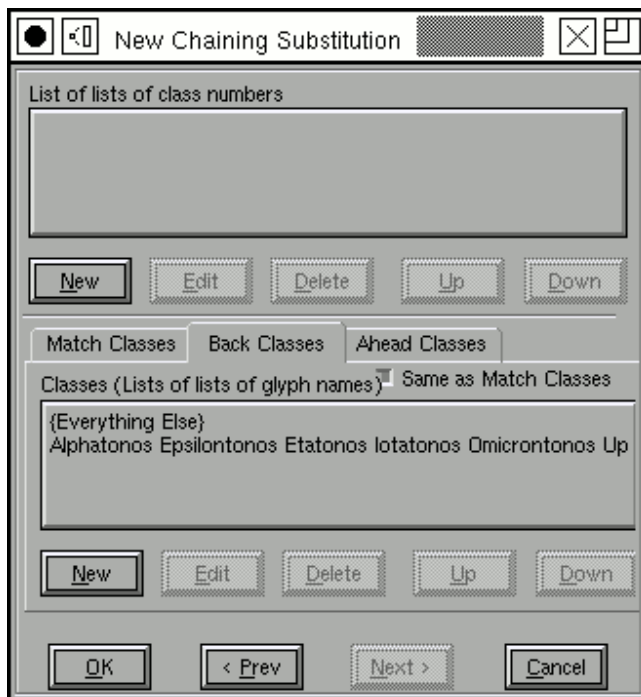


そこで、フォント・ビューに戻ってギリシア文字をすべて選択し、クラス・ダイアログ〔次の図〕左上の**【フォントから設定】**（Set From Font）ボタンを押します。

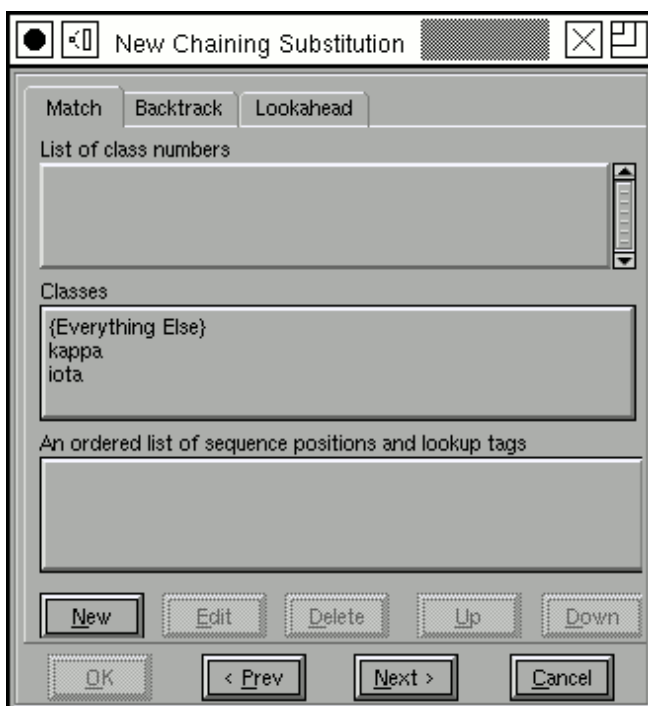


ただし、この場合は、対象のクラス（すべてのギリシャ文字）が構築されており、**【標準クラス Standard Class】** プルダウンリスト（スクリプト内の文字）から選択できます。その後、**【Next / 次へ>】** を押します。

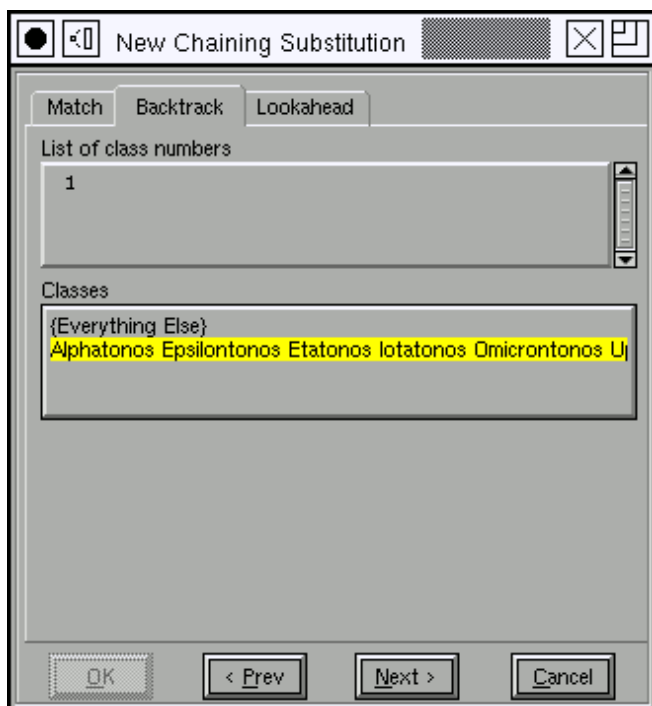
※ 《訳注》この画面以降は、20230101 日本語版で同様の画面・作業手順が検証できておらず、参考情報としてお読みください。



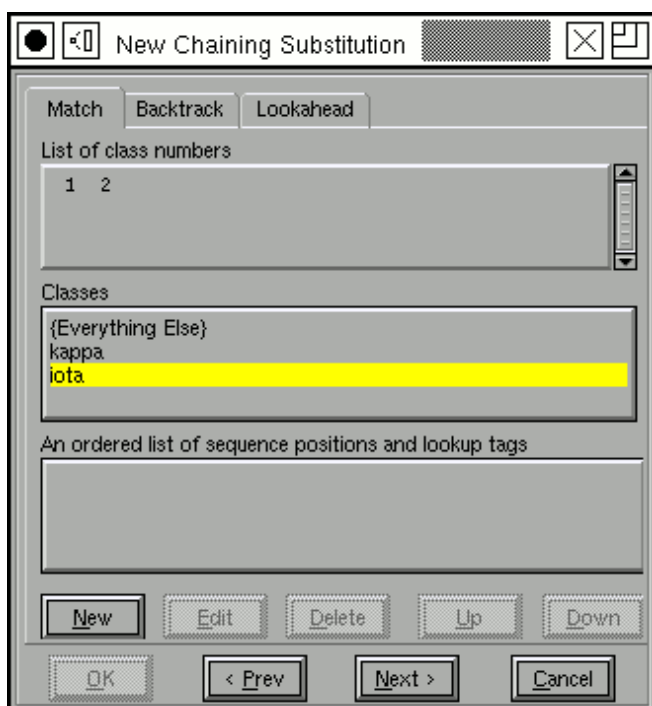
次に、「先読み」クラスに対しても同じプロセスを実行します（すべてのギリシャ文字で構成されるクラスをひとつ追加します）。



これですべてのクラスが定義され、いよいよルールのパターンを作成する準備が整いました。「クラス番号のリストのリスト」の下にある[新規]ボタンを押します。



最初のルールは「後戻り」領域のすべてのギリシア文字のどれかで始まりますので、「Backtrack」タブ（「後戻り」）をクリックし、「すべてのギリシャ文字」で構成されるクラスを押します。これにより、クラス番号がパターン領域（クラス番号のリスト List of class numbers）に配置されます。



「一致 Match」領域では、文字 kappa、次に iota と一致させたいので、「一致」タブをクリックし、「kappa」と「iota」のエントリをクリックします。

このルールには置換がないため、下部の領域を空白のままにして [Next / 次へ>] を押します。



これで最初のルール設定は終わりです。
こう書かれています。

- ひとつ前の文字が「後戻り Back」クラスの「クラス 1」と一致する必要があります（このクラスにはすべてのギリシア文字が含まれており、それが求めているものです）。
- 現位置の文字は、「一致」クラスの「クラス 1」と一致する必要があります（このクラスには「kappa」が含まれています）。
- 次の文字は、「一致」クラスの「クラス 2」（「iota」）に一致する必要があります。
- 一致した場合は、何もしません。

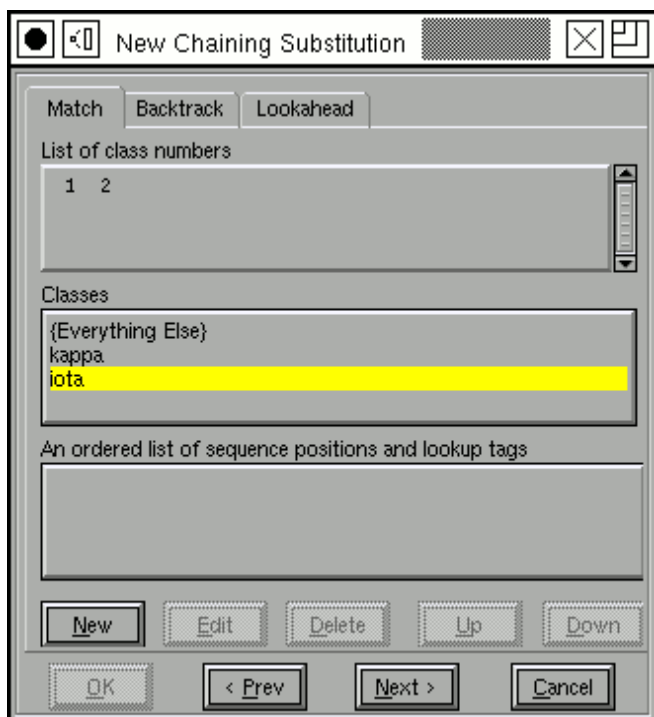
まだルールは二つあるので、**[OK]** を押してから **[Add Subtable]** （「サブテーブルの追加」）を押します。次に、必要なクラスを追加する作業を実行し、ルールの一致条件（文字列）を追加します。



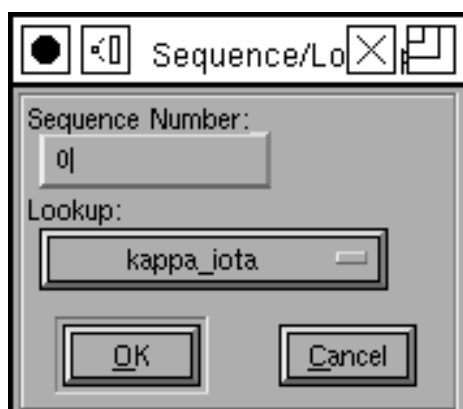
二番目のルール設定が終わりました。こう書かれています。

- 現位置の文字は、「一致」クラスの「クラス 1」と一致する必要があります（このクラスには「kappa」が含まれています）。
- 次の文字は、「一致」クラスの「クラス 2」（「iota」が含まれています）に一致する必要があります。
- その後の文字は「先読み」クラスの「クラス 1」と一致する必要があります（このクラスにはすべてのギリシア文字が含まれています）。
- 一致した場合は、何もしません。

[OK] を押し、最後のルール用に **[Add Subtable]** （「サブテーブルの追加」）を押します。



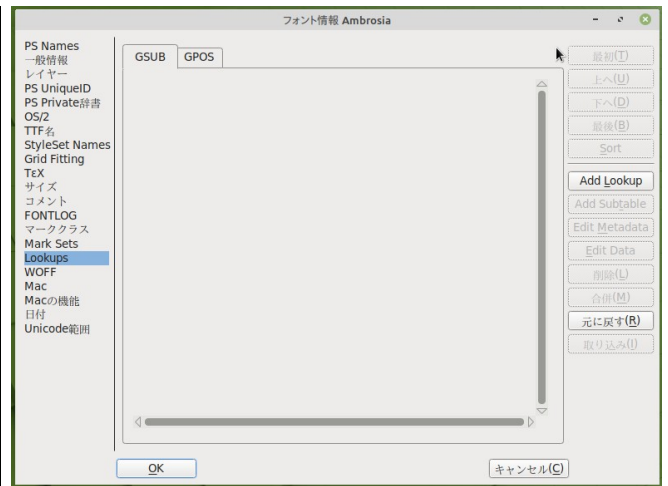
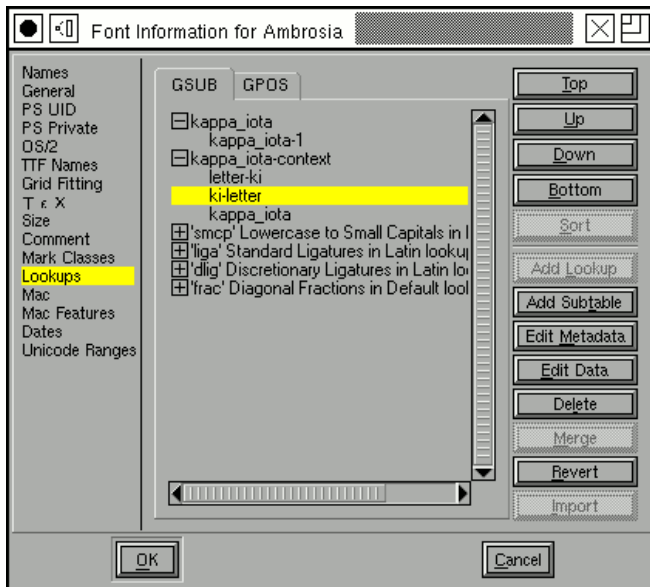
このルールには置換があります。二つの文字を取得して合字に変換するものです。したがって、「シーケンス位置リスト」の枠の下にある **[New / 新規]** ボタンを押すと、最初の文字（シーケンス位置 0 の文字）から開始して、「指定した語」の合字を適用します。



つまり、最初のふたつのルールに一致しないで、文字「kappa」に文字「iota」が続いている場合、それは独立した二文字のギリシア語であるはずですが、それに合字が適用されます。

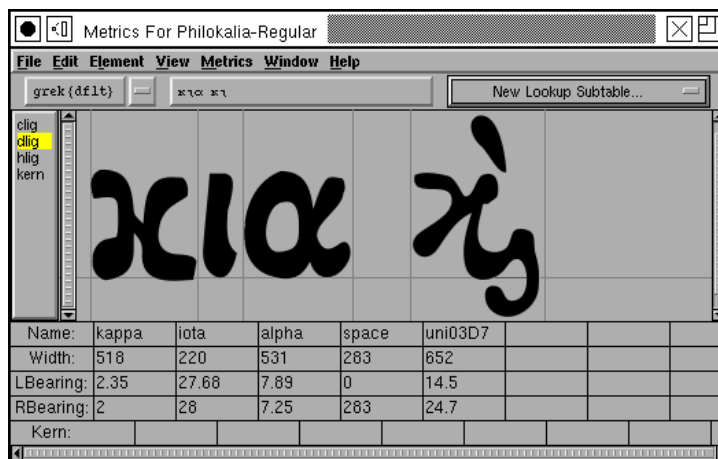


これで完了です。すべてのダイアログ・ウィンドウが閉じられるまで、**[OK]** ボタンを繰り返し押します。



20230101 日本語版画面

「検索」を作成したら、「メトリクス・ビュー」でその結果を検証できます。



(この例は Apostolos Syropoulos によって提供されました)



10. フォントの検証と生成法

Checking and generating a font

10.1. フォントの検証

フォントに含まれるすべてのグリフの作成を終えた後に、矛盾点の検証をするべきです。FontForge には、多くのよくある問題を発見するために設計された **【エレメント(L)】 ⇒ 【Validation】 ⇒ 【問題点を発見(O)...】** コマンドがあります。

単にフォント内のすべてのグリフを選択し、「**問題点を発見(O)...**」ダイアログを呼び出すだけです。とはいえ、ご注意ください。問題点として報告される事象すべてが、実際の問題というわけではなく、そのいくつかは FontForge が予期しないフォントのデザインの要素が報告されただけなのかもしれません。

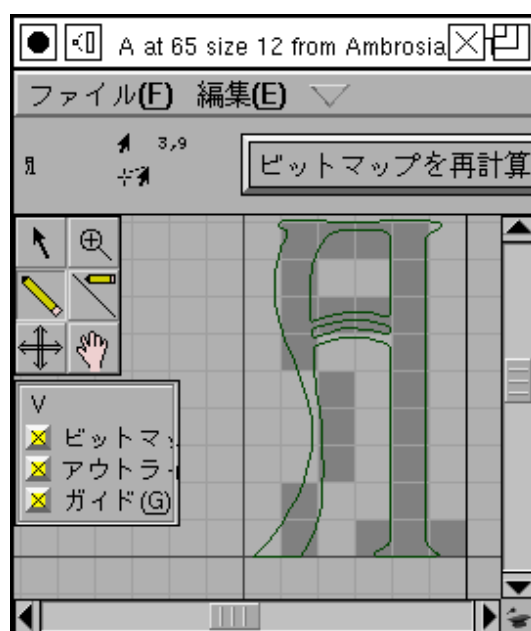
このダイアログは以下のような「問題点」を探することができます。

- 基準値のどれかに非常に近いが、正確には一致しない幅をもつステム
- 基準値のどれかに非常に近いが、正確には一致しない高さに置かれた点
- 水平または垂直にほとんど一致するが、わずかにずれているパス
- 通常ありえない位置に置かれた制御点
- ヒント位置にほとんど一致するが、わずかにずれている点
- 等々…

いちどきには同種の問題点だけを検証するのが最善です。そうしないと種類の異なる問題点をあれこれ調べることになって気が散るのです。

10.2. ビットマップ

この段階で、アウトライン・フォントに加えてビットマップ・フォントも欲しくなるかも知れません（これは必須ではありません）。メニュー項目の **【エレメント(L)】 ⇒ 【Bitmap Strikes Available】** を呼び出し、ビットマップを組み込みたいピクセルサイズを選択します（X と MS Windows ではピクセルサイズがポイントサイズと正確に一致しない場合があることにご注意ください）。次に、ビットマップエディター（ **【ウィンドウ(W)】 ⇒ 【ビットマップウィンドウを開く(B)】** ）を使用してビットマップ



を修整するか、ビットマップフォントを書き出してから誰か他の人のビットマップエディターを使用して修整することができます。

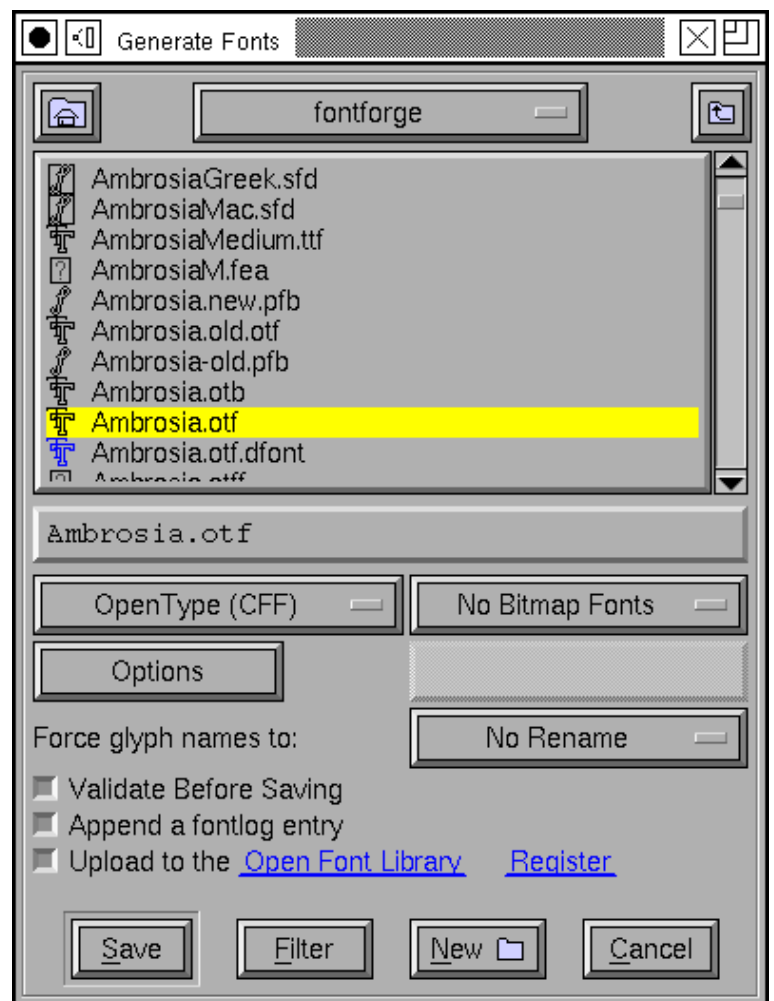
ビットマップについては、次章「11.ビットマップ・フォントを取り扱う」で詳しく説明します。

10.3. フォントの生成

作成したファイルを保存しても、（著者の知る限り）FontForge だけが理解できるフォーマットで保存されるだけです。フォントを使いたい場合には、これは役立たないでしょう。

作成したフォントを標準的フォントフォーマットのどれかに変換するためには、[[ファイル(F)] ⇒ [フォントを出力(G)...] を使用する必要があります。FontForge が表示するのはフォントフォーマットの膨大な羅列のように見えますが、実際には数種類の基本的なフォントフォーマット（PostScript Type 1、TrueType、OpenType、それに CJK フォントでは CID キー指定フォント）の変種があるだけです。

ビットマップのフォントの選択も可能です。FontForge は BDF³²（X で使います）、Mac NFNT³³（Mac で使います）、Windows FNT（Windows 2.0³⁴ で使うと思います）および TrueType（または OpenType）のラッパー³⁵内へのビットマップ組み込みに対応しています。



10.4. フォントファミリー

フォントを生成した後、おそらく一連の類似したフォントを生成する必要が生じるでしょう

32 BDF 形式： Bitmap Distribution Format。Adobe 社が所有するビットマップタイプのフォントを保存するためのファイル形式。X は、おそらく X Window Sytem（UNIX/Linux）

33 NFNT 形式： Macintosh New Font。

34 Windows 2.0： 英語版は 1988 年 5 月、日本語版は 1989 年 6 月発売。ということで、これ以降の記載内容の「時差」にもご注意ください。2024 年現在でのバージョンは「Windows 11」です。

35 wrapper： ある機能や処理を簡単に行なえるようにするために、元の機能を「包み込んで」（ラップして）異なる環境からでも利用できるようにするための或る種のプログラム。

う。ラテン、ギリシアおよびキリル文字のフォントでは「斜字体」（イタリック体またはオブリーク体）、「太字体」（ボールド体）、縦長体（コンデンスト）、幅広体（エクスパンデッド）の各フォント・スタイルは非常に一般的です。

同じフォント・ファミリーに属する異なるスタイルのフォントは、同じファミリー名を共有している必要があります（**【エレメント(L)】** ⇒ **【フォント情報(F)...】** ⇒ **【PS Names / フォント名】** タブで設定します）。フォント名はファミリー名の後ろにひとつ以上のスタイル名を、多くの場合はハイフンを挟んで付け加えたものです。ですからフォントファミリー「Helvetica」に属するフォントはすべて「Helvetica」というファミリー名が設定されていなければなりません。通常のスタイル（標準体）は単に「Helvetica」または「Helvetica-Regular」となり、ボールドスタイル（太字体）は「Helvetica-Bold」、オブリーク（斜字体）（Helvetica には真のイタリックはありません）は「Helvetica-Oblique」という具合に命名されることになります。

FontForge には **【エレメント(L)】** ⇒ **【スタイル(Y)】** というメニュー項目があり、標準体フォントをさまざまなスタイル（太字体、斜字体〔イタリック／オブリーク〕、縦長体、幅広体、小型大文字体など）に変換できるように設計されています。しかし、これらの変換はどれも完璧ではありません。必ず結果を確認してください。

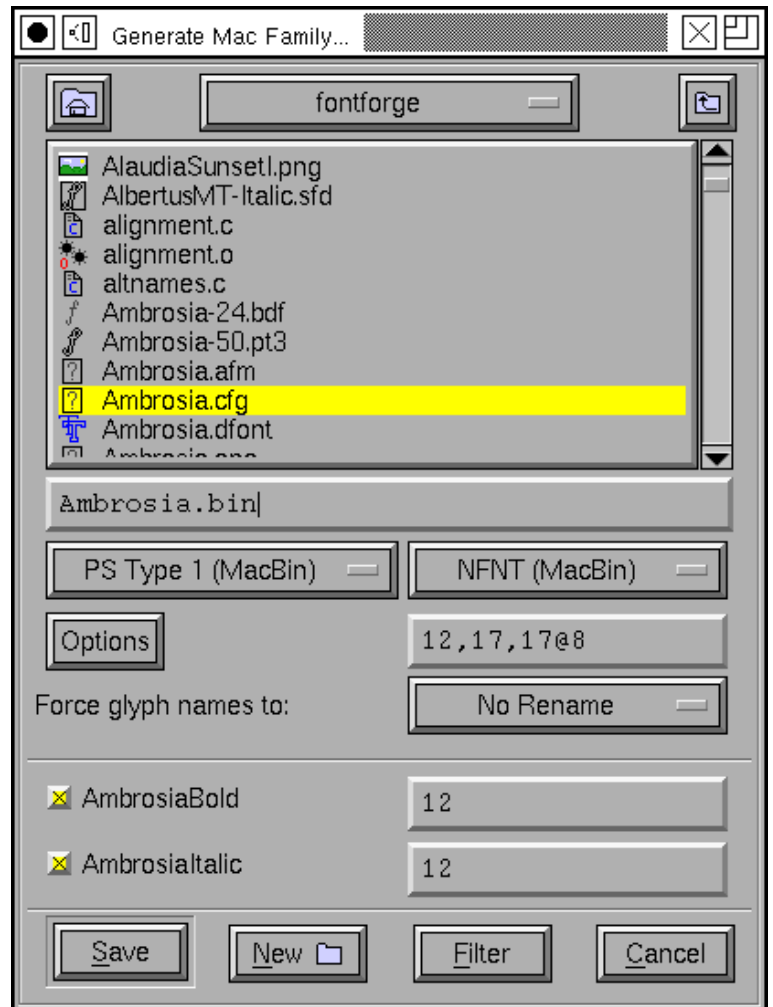
【エレメント(L)】 ⇒ **【変形(T)】** ⇒ **【変形(T)...】** ⇒ **【傾き】**³⁶ コマンドは通常フォントをオブリーク体のものに変換します。真のイタリック体の作成は一般的にはもうすこし複雑で、「a」の形が「*a*」へと劇的に変わったり、「f」にディセンダーがついて「*f*」となったり、「ilm」のセリフが「*ilm*」のように丸められたり、通常は他にも微妙な違いが生じます。また、フォントを傾斜した形に変更した後には、**【エレメント(L)】** ⇒ **【極大点の追加(X)】** を行なうべきでしょう。

あるフォントの「太字体 Bold」と「標準体 Plain」の両スタイルが既に揃っているならば（そして、それぞれのグリフが同じ順番で同じ個数の点を含むならば）、**【エレメント(L)】** ⇒ **【フォントの補間(L)...】** コマンドを使って「半太字体 DemiBold」スタイルを生成することができます。

TrueType フォント（と Windows）では、ステム〔縦軸線〕の太さに関する非常に細かいグラデーションに対応しています（Mac では「標準体 Plain」と「太字体 Bold」の二種類しか対応しません）。**【エレメント(L)】** ⇒ **【フォント情報(F)...】** ⇒ **【OS/2】** タブを選択すると、0 から 999 までの任意の値を太さとして設定することができます（「標準体」フォントの太さは一般的に「400 から 500」の間で、太字体 Bold は「700」くらいです）。TrueType は、文字幅の値にも広範囲に対応しています（Mac では「縦長体 condensed」、「標準体 plain」と「幅広体 expanded」しか対応していないのですが）。

36 Skew コマンド： 20230101 版では、この「Skew」コマンドが見当たらず、代わりに「エレメント⇒スタイル」の中に、**【イタリック(I)...】** と **【斜体(Q)...】** の二種類のコマンドがあります。

Windows PC では、フォント名が正しければ、システムはフォントがどのファミリーに属するかを導き出すことができます。しかし Mac では現状はより複雑です（が、これはもう過去の話かもしれません。現状は変化しつつあり、新しい拡張子について筆者はまだ理解していません）。Mac では限られた範囲の文字スタイル（標準体、斜字体、太字体、輪郭体 outline、縦長体、幅広体とこれらの組み合わせ）のみに対応し、これ以外のすべての物は別個のファミリーに分けられてしまいます。そのため、あるファミリーに属するさまざまなフォントへのポインターを保持する特別なテーブル（「FOND」といいます）を構築する必要があります。あるファミリーに含めたいすべてのフォントを開いてあれば（そしてそれらに正しい名前がついていれば）標準フォントから **【ファイル(E)】⇒【Mac ファミリーを出力(E)...】** を選択します。これにより、FontForge が現在のフォントと同じファミリーに属すると認識したすべてのフォントが列挙され、ファミリーに属するすべてのフォントファイルとともに、FOND 構造を作成できるようにします（場合により、すべてのフォントが同じファイルに含まれることも、そうでないこともあります。これは選択したフォント形式によって異なります）。



10.5. 最後のまとめ

これで新しいフォントが完成しました。しかしディスクに保存しておくだけでは役に立ちません。コンピューターにインストールする必要があります。システムによっては、フォントをコンピューターの「Fonts フォルダ」にドラッグするだけで簡単にインストールできるものもありますが、多くのさまざまな作業を必要とするものもあります。詳細は「[フォントのインストールに関する FAQ](#)」を参照してください。

「[FontForge のスクリプト処理の仕組みに関するチュートリアル](#)」（このリンク）は、第 12 章をご覧ください。

11. ビットマップ・フォントを取り扱う

Working with Bitmap Fonts

11.1. ビットマップ・ストライクの取り込み

FontForge は、もともとアウトライン・フォント・エディターですが、ビットマップ（およびグレースケール³⁷、またはアンチエイリアス³⁸）フォントを編集するための機能も備えています。ここでの便利な用語は「ストライク Strike」「ビットマップ画像」です。「ストライク」とは、ビットマップ・フォントの各文字サイズにおける文字画像です。FontForge がフォントを表示する場合、そのフォントにはアウトラインがひとつだけ含まれますが、潜在的には多くのビットマップ（やグレースケールの）「ストライク」が（いくつかの異なるピクセル・サイズに対して）含まれています。

最新のラスタイザーによる高品質な出力結果を考えると、アウトライン・フォントのほかにビットマップ・フォントも提供することが本当に必要なのかと疑問に思うかもしれません。ラテン文字（ギリシア文字、キリル文字）フォントの場合は必要ではないかもしれませんが、小さなピクセル・サイズの CJK 文字フォントの場合、最良のラスタイザーでも人間の目には敵いません。1980 年代に Apple 社が初めて PostScript フォントを採用したとき、アウトライン・フォントに合わせてビットマップ・フォントを用意することが不可欠であると考えていました。そのため、少なくともひとつのビットマップも用意されていない PostScript フォントは使用することさえ不可能でした。現在でも、Mac で Type1 フォントを使用する場合は、その文字を示すビットマップ・フォントが必要です。

もちろん、Type1 フォントは今では少し時代遅れ³⁹ですが、sfnt ファイル形式（OpenType フォントと TrueType フォントの両方で使用される形式）は、フォントのアウトライン・バージョンとともに埋め込みビットマップ・ストライクに対応しています。このストライクは、用意されているピクセル・サイズであればアウトラインをラスタイズするよりも優先して使用されます。

アウトライン・フォントとビットマップ・ストライクの両方を含む sfnt データがある場合は、**【ファイル(F)】⇒【開く(O)】** コマンドで直接開くことができ、FontForge がすべてのごまごまとした部分をロードします。

【ファイル(F)】⇒【開く(O)】 コマンドを使用してビットマップのみのフォントを直接開くこともできますが、同じ基本フォントの複数のストライクが別々のファイルにある場合は、

37 greymap font : bitmap font が白黒のドット・パターンだけで作られているのに対して、中間色（灰色）も入れているため小さい文字サイズでもなめらかな線を表示できるフォント。「アンチエイリアスフォント」「グレースケールフォント」とも呼ばれています。

38 anti-alias font : 斜線や曲線のギザギザを目立たなくするために、境界に中間色を補い（アンチエイリアシング）、低解像度でも滑らかに見えるよう工夫されたフォントです。

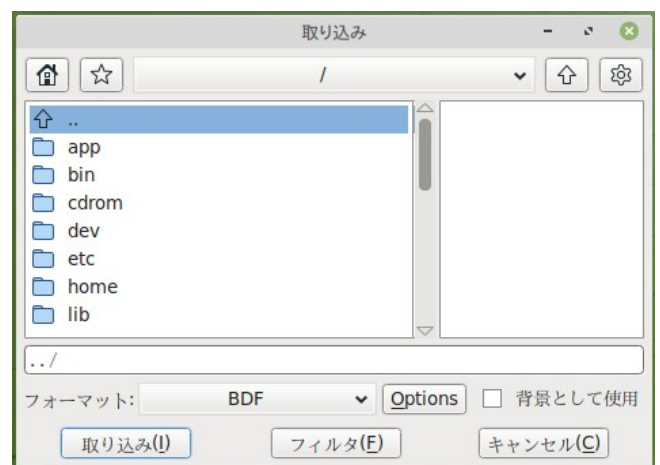
39 Type 1 フォント : Adobe 社、2023 年 1 月に、PostScript Type 1 フォントのサポート終了 [リンクはこちら]。

「開く(O)」で最初のストライクのみ（またはアウトライン・バージョンのみ）を開くことをお勧めします。そして、**【ファイル(F)】⇒【取り込み(I)】**を使用して、他のストライクをひとつの FontForge フォントに統合します。

たとえば、Ambrosia.pfb⁴⁰（アウトライン・フォント）、Ambrosia-12.bdf⁴¹、および Ambrosia-24.bdf（ふたつのビットマップ・ストライク、「-12」は高さ 12 ピクセル、「-24」は高さ 24 ピクセル）の三つのフォント・データがあり、それらをすべてひとつのフォントに纏めたいとします。まず最初に、**【ファイル(F)】⇒【開く(O)】** コマンドを使用して Ambrosia.pfb を開き、**【ファイル(F)】⇒【取り込み(I)】** を使用して両方のビットマップ・ストライクを選択します（「Shift キー」を押したまま）。

一方、Ambrosia-12.bdf と Ambrosia-24.bdf のふたつのビットマップしかない場合は、**【ファイル(F)】⇒【開く(O)】** を使用してどちらかのストライクを開き、**【ファイル(F)】⇒【取り込み(I)】** を使用して残りのひとつを取り込みます。

【取り込み(I)】 ダイアログに「☐ 背景として使用」と書かれたチェックボックスがあることに気づいたかもしれません。これにチェックを入れると、ビットマップ・フォントはフォントとしては読み込まれませんが、アウトライン・フォントの背景として読み込まれ、そこから**【エレメント(L)】⇒【自動トレース(R)】**を選択してビットマップ画像からアウトライン・フォントを生成できます。ただし、注意してください。トレースするフォントが大きく（100 ピクセル以上）ない限り、かなり悪い結果になります。



11.2. ビットマップ・ストライクの作成

ある所定のピクセル・サイズのビットマップ・ストライクをフォントに追加する場合は、**【エレメント(L)】⇒【Bitmap Strikes Available】**〔利用可能なビットマップ・ストライク〕を使用します。

ポイント・サイズの扱いに慣れている場合は、ポイント・サイズが画面の解像度に依存することに注意してください。このダイアログ・ウィンドウ〔次図〕では、ふたつの標準解像度のポイント・サイズとピクセル・サイズ間の変換が表示されます。解像度は、使用しているウィンドウ・システムによって異なります。

もちろん、ひとつ以上のピクセル・サイズを追加することもできますし、必要がなくなればそのピクセル・サイズを削除することもできます。

40 .pfb フォント： Adobe Type 1 フォントを格納するファイル形式。

41 .bdf フォント： Adobe のグリフのビットマップ配布形式。

通常、新しいストライクを作成すると、アウトライン・フォントからラスターライズされます。アウトライン・フォントがない場合、FontForge は利用可能な最も大きいストライクを新しいサイズに拡大または縮小します（ほとんどの場合、これは相当見苦しいものになりますが、ないよりはましでしょう）。

ダイアログの下部には、「☐ Create Rasterized Strikes (Not empty ones)」〔ラスターライズされたストライクを作成する（空のストライクではない）〕チェックボックスもあります。このチェックボックスを外すと、上記のラスターライズ作業が実行されず、グリフを含まない新しいストライクが作成されます（もちろん、後からグリフを追加することもできます）。



11.3. ビットマップの属性

X11⁴²上のフォントが bdf および pcf フォントであった昔は、各フォントに一連の属性が付加されていました。X11 のフォントが OpenType ビットマップに移行しても、このような属性が依然として役立つかどうかはわかりません。役立つかもしれません。いずれにしても、FontForge はそうした属性を BDF フォントと OpenType ビットマップ・フォントの両方で保存します。各ストライクには独自の属性セットがある場合があります。FontForge は、ストライクを作成するときに属性を自動的に生成します（し、属性を含むフォントを読み込むときにはその属性を保持します）。【**エレメント(L)**】⇒【**その他の情報**】⇒【**BDF 情報...**】ダイアログを使用して現在の属性値を上書きできます。

11.4. 新しいビットマップ専用フォントの作成

アウトラインを持たず、ビットマップのみの新しいフォントを作成したい場合は、【**ファイル(F)**】⇒【**新規(N)**】を使用して新しいフォントを作成し、次に【**エレメント(L)**】⇒【**Bitmap Strikes Available**】を選択して、そのフォントいくつかの（空の）「ストライク」を追加します。新しいフォントに「ストライク」が追加されると、そのフォントはビットマップのみのフォントになります。

Apple には sfnt ファイル形式（OpenType フォントと TrueType フォントの両方で使用される形式）のビットマップのみのバージョンがあり、X11 にもあります。もちろん、ふたつの形式は若干異なっています。Windows は、ビットマップのみの sfnt 形式に対応していません。

42 X11： 「X ウィンドウシステム・バージョン 11」の略称。

11.5. フォントビューのビットマップ

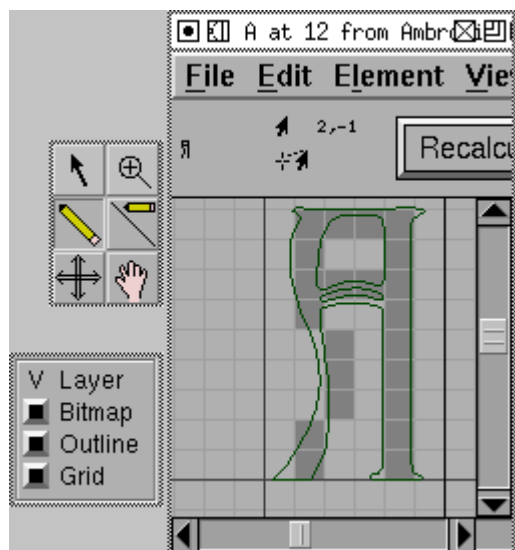
「フォントビュー」ウィンドウ（および「メトリックビュー」ウィンドウ）の **【表示(V)】** メニューには、フォント内のすべてのビットマップ・ストライクのリストが含まれており、フォントビューにどの「ストライク」を表示するのかが選択できます。また、 **【表示(V)】** ⇒ **【ビットマップの拡大(M)】** でストライクの表示倍率を変更することもできます（高解像度の画面では8ピクセル・フォントの詳細を確認するのが難しいため、実際のサイズの3倍で拡大表示するとよいでしょう）。

フォントビューにある多くのコマンドは、フォント内のすべてのストライク（およびアウトライン）に適用されます。たとえば、 **【編集(E)】** ⇒ **【コピー(C)】** を選択すると、アウトライン・グリフとすべてのビットマップ・グリフがコピーされます。時には、ひとつのストライクのみを操作したい場合があります。その場合は、 **【編集(E)】** ⇒ **【コピー元の指定(F)】** ⇒ **【すべてのフォント(A)】** の選択を解除します。

【エレメント(L)】 ⇒ **【組み立て(U)】** コマンドは、アクセント付きのアウトライン・グリフを形成するのと同じように、アクセント付きのビットマップ・グリフを形成します。

フォントビューにビットマップ・ストライクが表示されているときに文字をダブルクリックする⁴³と、FontForgeは（アウトライン編集ウィンドウを開くのではなく）現在のストライク内のその文字を表示する「**ビットマップ編集ウィンドウ**」を表示します。文字が現在のストライクに一致するグリフを持たない場合、FontForgeはフォントのアウトライン版のグリフからラスタライズすることによってグリフを作成します。

11.6. ビットマップ・フォントの編集



「ビットマップ編集ウィンドウ」は、ほとんど説明を要しないでしょう。ウィンドウは簡単なビットマップ編集画面です。「鉛筆ツール」を使って各ピクセルをクリックすると、ピクセルが変化します。「直線ツール」は、開始点と終着点の間を直線で結びます。

ビットマップの背景にアウトライン・グリフの輪郭線が見えます。

【ビットマップを再計算】 ボタンを押せば、（このアウトラインがラスタライズされて）ビットマップが再生成されます。

アンチエイリアス処理されたグレースケールマップを編集するには、「鉛筆ツール」で適用する色（実際には灰色の階調）を選択する必要があるため、さらに複雑になります。このため、考えられるすべて

⁴³ ダブルクリックする： 反応しない場合には、右クリックまたは **【ウィンドウ(W)】** メニューから各ウィンドウを開く必要があるかもしれません。

のグレー色を含む追加のパレットが用意されています。

11.7. 最小限のビットマップ・ストライク

場合によっては、ストライクにフルセットのビットマップ・グリフが必要ない場合があります。おそらく、ラスターライザーはひとつを除くすべてのグリフに対して適切に機能します。その場合は、空のストライクを作成し（**【エレメント(L)】** ⇒ **【Bitmap Strikes Available】**）を選択し、「☐ Create Rasterized Strikes (Not empty ones)」のチェックを外します）、気になるグリフをひとつ選択して、**【エレメント(L)】** ⇒ **【ビットマップの再生成(B)...】** を実行します。

あるいは、完全にラスターライズされたストライクを作成してから、不要なグリフを選択して、**【エレメント(L)】** ⇒ **【ビットマップを除去...】** でも構いません。



12. FontForge スクリプト チュートリアル

FontForge Scripting Tutorial

【注記】

FontForge は Python スクリプトを提供するようになりました。Python に精通している場合は、おそらく懸命な判断です。「python」については多くの情報が入手可能ですので、ここでは繰り返しません。FontForge による Python への独自の追加は「[ここに](#)」文書化されています。

説明をとてとても解りやすくするつもりではありますが、ここでプログラムを教えるつもりは**ありません**。

12.1. 簡単な事例

(PFB と AFM が一組みになった) Type1 形式の PostScript フォントがあって、それを TrueType フォントに変換することを考えましょう。この処理を行なうにはどのようなスクリプト⁴⁴を書けばいいのでしょうか？

これをプログラム UI⁴⁵上で実行する場合は、フォントを【**ファイル(F)**】⇒【**開く(O)**】で開き、続いて【**ファイル(F)**】⇒【**フォントを出力(G)...**】で TrueType フォントを書き出します。スクリプトを書く場合にも、基本的にはこれと同じ事を行なうことになります。

12.1.1. とりあえずの解決法

```
Open($1)
Generate($1:r + ".ttf")
```

大体において、メニューコマンドと同じ名前のスクリプト関数が存在します（まあ、メニュー項目の英語表記名と同じであるだけですが）。

「\$1」はおまじないです。これは「[スクリプトに渡す最初の引数](#)」の意味です。

「\$1:r + ".ttf"」はずっと複雑なおまじないです。その意味は、「最初の引数 (\$1) を取り、拡張子（おそらく「.pfb」）を削除し、文字列「.ttf」をファイル名に付け加える」というものです。

スクリプト・コマンド「Generate」は、与えられた拡張子によってどのタイプのフォン

44 Script： コンパイルを必要としない簡易的なプログラミング言語。

45 UI (user interface)： ユーザーインターフェース。

トを出力するかを決定します。この例では「ttf」なので「TrueType フォント」を指定しています。

AFM ファイルを読み込むように指定していないことに注意して下さい。これは、Open コマンドが pfb と同じディレクトリにある場合、afm を自動的に読み込むからです。

12.1.2. 実世界で考慮すべき事柄

スクリプトがどのようなものかについて説明しました。これを実用的な物にするには、おそらくこのデータを格納する専用のファイルが必要でしょう。そこで「convert.pe」という名前のファイルを作成し、その中に上記のスクリプトを格納します。

しかし、それを遥かに有用な物にするために、スクリプトの先頭にコメント行を追加する必要があります（コメント行とは文字「#」で始まる行のことです）。

```
#!/usr/local/bin/fontforge
Open($1)
Generate($1:r + ".ttf")
```

このように記述したならば、以下のようにタイプします。

```
$ chmod +x convert.pe
```

このコメントは FontForge にとって重要ではありませんが、Unix シェルでは意味があります。これについては次のセクションで説明します。

12.1.3. スクリプトの起動法と引数の指定方法

OK, これで基本的なスクリプトができました。使うにはどうするのでしょうか？

まあ、以下のようにタイプして FontForge に直接渡すこともできます。

```
$ fontforge -script convert.pe foo.pfb
```

が、前述のコメントを追加しているならば、以下のようにタイプすれば十分です。

```
$ convert.pe foo.pfb
```

この場合シェルは、スクリプトを処理するために FontForge を呼び出す必要があることを認識しています。

12.1.4. ループの使い方

これでどこにも問題は無いのですが、多数のフォントを変換したいとなった時には、これは面倒な作業となるでしょう。そういうわけでスクリプトを修正して、多数のファイル名を受け取り、一度にひとつずつ処理できるようにしましょう。

```
#!/usr/local/bin/fontforge
i=1
while ( i<$argc )
  Open($argv[i])
  Generate($argv[i]:r + ".ttf")
  i = i+1
endloop
```

ここでは変数「\$argc」と「\$argv」を導入しました。前者はこのスクリプトに渡される単純な引数の数を表し、後者はそれらの引数をすべて含む配列ですから、「\$argv[i]」は i 番目に指定した引数ということになります。

次は、これです。

```
i=1
```

これは、「i」というローカル変数を用いることを宣言し、それに値「1」を代入しています。

while ループは「while」キーワードと「endloop」キーワードの間にあるすべての文を、条件「(i<\$argv)」が真である限り実行します。言い替えれば、まだ変換を行なうべき引数が存在する限り、このループは実行され続けます。

これにより、このスクリプトを次のように、またはそれに似たもので、

```
$ convert.pe *.pfb
```

呼び出すことができます。

12.1.5. より複雑な事例

ここで、Type1 フォントから TrueType への変換に加え、TrueType フォントを OpenType フォントに変換することができるスクリプトが必要になったとしましょう。ではこのスクリプトをもっと複雑なものにしてみましょう。

```
#!/usr/local/bin/fontforge
i=1
format=".ttf"
while ( i<$argc )
  if ( $argv[i]=="-format" || $argv[i]=="--format" )
    i=i+1
    format = $argv[i]
  else
```

```

        Open($argv[i])
        Generate($argv[i]:r + format)
    endif
    i = i+1
endloop

```

このスクリプトの起動法は以下のようになります。

```
$ convert.pe --format ".ttf" *.pfb --format ".otf" *.ttf
```

ここでは、「format」という新しい変数を導入しています。これは、この後で使用する出力形式を表します。この値はまず TrueType の「.ttf」で初期化されますが、ユーザーが「--format」（または「-format」）という引数を指定したならば、どんな出力形式にでも、ユーザーが指定して切り替えることができます。

また、ここで新しく「if」を付け加えています。この文は、条件（ \$argv[i]=="-format" || \$argv[i]=="--format" ）が「真」である場合に「if」と「else」の間の各文を実行し、「偽」の場合には「else」と「endif」の間の各文を実行します。「||」演算子は「または」を意味しますので、この条件文は \$argv[i] が「-format」または「--format」のどちらかである時に真になります。

実際には、以下の事を確認するために、いくつかのエラーチェックを行なう必要があります。

- format 変数に格納するための次の引数が指定されているか
- 引数が適切な値（.ttf, .pfb, .otf, .svg, ...）を含んでいるか

```

#!/usr/local/bin/fontforge
i=1
format=".ttf"
while ( i<$argc )
    if ( $argv[i]=="-format" || $argv[i]=="--format" )
        i=i+1
        if ( i<$argc )
            format = $argv[i]
            if ( format!=".ttf" && format!=".otf" && \
                format!=".pfb" && format!=".svg" )
                Error( "Expected one of '.ttf', '.otf', '.pfb' or '.svg'
for format" )
            endif
        endif
    else
        Open($argv[i])
        Generate($argv[i]:r + format)
    endif
    i = i+1
endloop

```

長い行が含まれているため場合は、バックスラッシュ記号（\）を用いて二行に分割していることに注意してください。通常、行末は文の終わりを表しますので、バックスラッシュ

を用いて次の行まで文が続いていることを示す必要があるのです。

これで、TrueType から PostScript フォントを変換したい場合にも、正しく出力ができるようになりました。しかし、この変換をより改善することができます：

```
#!/usr/local/bin/fontforge
i=1
format=".ttf"
while ( i<$argc )
  if ( $argv[i]=="-format" || $argv[i]=="--format" )
    i=i+1
    if ( i<$argc )
      format = $argv[i]
      if ( format!=".ttf" && format!=".otf" && \
        format!=".pfb" && format!=".svg" )
        Error( "Expected one of '.ttf', '.otf', '.pfb' or '.svg'
for format" )
      endif
    endif
  else
    Open($argv[i])
    if ( $order==2 && (format=="otf" || format=="pfb" ))
      SetFontOrder(3)
      SelectAll()
      Simplify(128+32+8,1.5)
      ScaleToEm(1000)
    elseif ( $order==3 && format=="ttf" )
      ScaleToEm(2048)
      RoundToInt()
    endif
    Generate($argv[i]:r + format)
  endif
  i = i+1
endloop
```

TrueType はその性質上、PostScript よりも多くの輪郭点を含む傾向があります。しかし、**【エレメント(L)】** ⇒ **【単純化(S)】** ⇒ **【単純化(S)】** コマンドを用いれば、あるフォーマットから別のフォーマットへの変換を行なう際に、輪郭点の個数を減らすことができます。また、TrueType フォントが「em あたりユニット数」が「2 の冪乗」（通常、2048 または 1024）になっているのに対し、PostScript フォントは「em あたり 1000 ユニット」である必要があるので、この慣習への対応も必要です。最後に、TrueType フォントは輪郭点の座標に整数値（場合によっては半整数値）しか使用することができません。

12.2. その他の事例

12.2.1. Type1 フォントにアクセントつき文字を追加する

Unicode で定義されたアクセントつき文字を全部を含んだ Type1 フォントはほとんどありません。FontForge を使えば、Type1 フォントを読み込み、フォントが許す限り（フォントにオゴネク記号が含まれていなければ、FontForge は「オゴネク記号付きの A」文字を作成することはできませんが）、できるだけ多くのアクセントつき文字を追加することは簡単です。

```
#!/usr/local/bin/fontforge
Open($1)
Reencode("unicode")
SelectWorthOutputting()
SelectInvert()
BuildAccented()
Generate($1:r + ".otf")
```

12.2.2. Type1 フォントと Type1 エキスパートフォントを併合し、適切な GSUB テーブルを作成する

Adobe 社は、かつて通常フォントと「エキスパート」フォント⁴⁶（これには、小型大文字・小文字数字ノンライニング数字などが含まれていました）を含むフォントパックを出荷していました。現在では⁴⁷、それらはすべて、適切な GSUB 項目によってグリフ同士にリンクが張られたひとつの OTF フォントに詰め込まれています。

```
#!/usr/local/bin/fontforge
Open($1)
MergeFonts($2)
RenameGlyphs("AGL with PUA")
SelectAll()
DefaultATT("*")
```

12.2.3. より多くの事例

「[スクリプト処理](#)」に関するページ（章）を参照してください。



46 Adobe used to ship ... an “expert” font : 《訳注》いつ頃の話か特定できていません・・・。「エキスパート字形」という異体字表示（グリフ置換）の話であれば 2007 年頃の記事が WEB でヒットします。

47 Now-a-days : [通常 Nowadays と綴られるが…] 英語チュートリアル執筆時点だとすれば、2012 年頃と推定されます（著作権表示の年表示に拠る）。

13. 特定の用字系において特に考慮すべき点

Special thoughts for special scripts

13.1. 特定の用字系において特に考慮すべき点

Microsoft 社は、ワードプロセッサが特定の用字系に対してデフォルトでどの (OpenType) 機能を用意するべきかに関するいくつかの情報を提供しています。

【警告】 機能が文書化されていて、有益そうに見えるからといって、Uniscribe⁴⁸ が目的とする用字系でそれを使用するとは限りません。多くのラテン文字フォントは「init」、「medi」、「calt」などを必要としますが、Uniscribe においては、これらの機能はどれもラテン文字では使用可能になっていません。

【警告】 Uniscribe がその機能をサポートしているからと言って、すべてのアプリケーションがそれを使用可能なわけではありません。Uniscribe は (2005 年現在) ラテン文字で「liga」をサポートしていますが、Word も Office もそれを処理しません。

【警告】 Uniscribe (マイクロソフト社の Unicode テキストレイアウトルーチン) は、用字系によっては GPOS または GSUB テーブルを無視することがあり、GPOS/GSUB に正しいデータが含まれていないとフォント全体の使用を拒絶することすらあります。ヘブライ文字のフォントは GPOS と GSUB の両方を含んでいなければなりません。含まれていないとフォントは使用されません。ラテン文字のフォントはどちらも必要ありませんが、GSUB が含まれていないと GPOS は使用されません。ですから、今のところ、片方のテーブルが存在してもう片方が存在しないときには、FontForge は存在していない方のダミー版を生成します。

13.1.1. 共用文字

多くの文字がひとつ以上の用字系で使用されます。Unicode では、数字や句読点などは「Common [共用]」文字集合⁴⁹に属すると言われています。OpenType はこの用字系を認識しません。最も近いのは「DFLT」⁵⁰文字集合です。筆者の理解では、共通文字集合内の文

48 Uniscribe: 細かい文字体裁や複雑なスクリプトを処理するための高度な制御を可能にするマイクロソフト社の API (アプリケーション・プログラミング・インターフェース)。

49 文字集合: この章では、文字 (character) や用字系・文字集合 (script) のような類似の用語が混在していますが、すべて「文字」と訳すのが一番判り易いでしょう。ここでの character と script の違いは、「おおむね」用字系 (文字集合) が特定の言語を書き表すためのひとまとまりの文字や記号の集まりを指すのに対し、文字は個々の文字・記号を含意していることです。

50 DFLT: おそらく「デフォルト default」=「既定」の意味。

字には、OpenType によって隣接するテキストの文字が割り当てられると考えられます。

したがって、あるフォントがラテン文字、ギリシア文字、またはキリル文字に対応している場合、数字と句読点はこれら三つの文字のどれにも含まれており、そのような文字に適用されるすべての機能がすべての文字に存在しているはずです。たとえば、数字 9 が数字 1 とカーニングされる場合、そのカーニング・データはラテン文字だけでなくキリル文字、ギリシア文字でも存在するはずであるということです。

ただし、あるフォントの数字を別のフォントの漢字に交えて使用することは可能です（日本語では一般的だと思います）。これは、数字が漢字に対応していないフォントで使用されている可能性があることを意味します。ただし、OpenType ではそれらに漢字を割り当てます。したがって、「検索」は適用されません。アドビ社では、文字内のほとんどの機能を代替文字集合である「DFLT」にも含めることを推奨しています。この慣習に従っている人が他にいるのかどうかはわかりません。

13.1.2. ラテン文字

Uniscribe は以下の機能をサポートします。

ラテン文字においては特に複雑な点はありません。ラテン文字のフォントは一般的に 1 バイト符号化に収まり、フォント・テーブルはありません（あってもごく少数です）。構築できるアクセント付きグリフは多数ありますし、「位置記号から基底へ」の位置決めを使用することもできます。幾つかのグリフの組み合わせについてカーニングを作成するべきでしょう。作成が必要ないいくつかの合字があります（「f」に関する「ff」「fi」「fl」「ffi」「ffl」や恐らく「st」も——ただし、いくつかの言語〔トルコ語〕では、合字「fi」は作成すべきではありません）。

一組み小型大文字を追加したくなるかもしれません。その場合、Adobe 社はラテン文字の小型大文字用区画を私用領域に予約しています——これは現在「非推奨」⁵¹です。

いくつかの言語には、その言語独特の要求事項があります。

- [ポーランド語について](#)（リンク切れ）

13.1.3. ギリシア文字

Uniscribe は以下の機能をサポートしています。

ギリシア語にも複雑さはあまりありません。現代のギリシア文字フォントは通常、1 バイト符号化に適合しています。現代ギリシア文字では、構築する必要があるアクセント付きグリフは少しですが、多声調⁵²の〔古代〕ギリシア語では多くのグリフがあり、「位置記号から基底へ」および「位置記号から位置記号へ」の位置決めが選択できます。カーニングを生成する必要があります。筆者は現代ギリシア語の標準的な合字については判りません（ただ

51 原文 deprecated「価値下落」： deprecated「非推奨」「廃止予定」の誤記か？

52 polytonic ポリトニック： 多声調の。古代ギリシア語の、発音の異なる声調やピッチ（発音の高さ）を示すために、さまざまな発音区別符号（ダイアクリティカルマーク）で書き分ける表記体系。

し、古代ギリシア語には一部のグリフに合字や異体字がありました）。

ギリシア文字でも、小型大文字を用いる選択が可能です。筆者は、そのためのブロックを私用領域に予約していました——これも「非推奨」になりました。

13.1.4. キリル文字

Uniscribe は以下の機能をサポートしています。

キリル文字のフォントも 1 バイト符号化に適合しています。少数のアクセントつきグリフがあります。カーニングも作成するべきです。筆者は標準的な合字については判りません。

いくつかの言語は異体字グリフを必要とします（「loca」機能〔地域設定〕で指定します）。

- セルビア語/マケドニア語

13.1.5. アラビア文字

Uniscribe は以下の機能をサポートしています。

アラビア文字は語頭、語中、語尾および独立形の完全なセットを必要とします——Unicode はこのための領域を予約しています。また、アラビア文字は膨大な合字セットを必要とします——Unicode は多くの予約領域を設けていますが、場合によっては、追加の合字が必要になることもあるように思います。アラビア文字は母音記号を文字の上に配置するためのひと組の接続位置記号（母音記号から基底文字へ、母音記号から合字へ）を必要とします。アラビア文字では、グリフ分解テーブルも必要かもしれません。

アラビア語の優れた組版を実現するためには、ひとつのグリフに四つ以上の文字形態が必要であるとのことでした。これにどのように対応するべきかについてはよく判りません。

アラビア語は「右から左へ」書きます。

13.1.6. ヘブライ文字

Uniscribe は以下の機能をサポートしています。

ヘブライ文字は複数の語尾形をもちますが、これらのための特別なテーブルは必要ありません。ヘブライ文字はカーニングを必要な場合があります。ヘブライ文字には、文字の上に母音記号を配置するための一組の位置記号（母音記号から基底文字へ）のテーブルが必要です。ヘブライ文字はグリフ分解テーブルを必要かもしれません。必要な合字については判りません。

ヘブライ語も「右から左へ」書きます。

13.1.7. インド系の諸文字

Uniscribe は以下の機能をサポートしています。

インド系の諸文字には合字のセットが必要です。

（この用字系ではより多くの事柄を必要とするでしょうが、それが何なのか検討が付きません。）

13.1.8. 韓国ハングル文字

Uniscribe は以下の機能をサポートしています。

ハングルは「音標文字」から組み立てられた音節文字の組み合わせで構成されています。通常、フォントは予め組み合わされた音節文字から成り立っています。

複雑さは、これら全ての音節の膨大な組み合わせによる爆発的な規模によるものです。この複雑さは、PostScript では CID キー指定フォントによって緩和されています。

ハングル文字は、縦書きと「左から右へ」の横書きが併用されます。いくつかのグリフは縦書き時には異なる方向をもちます（例えば括弧など）。

13.1.9. 日本語と中国語（と韓国語の漢字）

Microsoft は、（著者の知る限り）これらの言語の用字系について記述していません。

これも膨大なグリフの集合が必要であり、PostScript では CID キー指定フォントを使ってこれを扱っています。

縦書きと左から右への横書きが併用されます。いくつかの文字（例えば括弧）は縦書き時には異なる方向を持ちます。

