

# Footy2 テキストエディタコントロール解説書

Copyright © 2004-2009 Shinji Watanabe

Copyright © 2009 Project. Footy

このドキュメントをご覧になる前に

- この文書はFooty2の機能を解説する文書です。C++開発者向けに記述されております。
- このファイルを印刷するとそのまま説明書になります。印刷してご覧になることを推奨します。
- 「しおり」を表示させる機能を使用することで、該当する項目に即座にジャンプすることが出来ます。
- このファイルを閲覧するときはAdobe Reader 8の使用を推奨します。
- 文書の無断転載は固く禁じます。ただし、引用の範囲であれば著作権法上問題はありません。
- この文書は誰でも入手することが出来るため、管理文書(*Confidential*)ではありません。

# 目次

1.最初に.....	4
1.1.ご挨拶.....	4
1.2.使用条件.....	4
1.3.このドキュメントの更新履歴.....	5
2.Unicode と文書管理.....	6
2.1.文字コードと Unicode.....	6
2.2.Unicode と Unicode 以外の文字コード.....	6
2.3.文字コードの自動判別について.....	7
2.4.BOM について.....	7
2.5.ANSI 系関数と WIDE 系関数について.....	8
3.バージョンの取得.....	9
3.1.バージョンの取得関数.....	9
4.ウィンドウ操作.....	10
4.1.Footy の ID 番号.....	10
4.2.Footy の生成手順.....	10
4.3.Footy の生成に関わる関数.....	11
4.4.Footy のウィンドウを操作するための関数.....	13
5.編集コマンド.....	20
5.1.編集コマンドとは.....	20
5.2.クリップボード関連機能.....	20
5.3.アンドウ、リドウの機能.....	22
5.4.選択のための機能.....	25
5.5.検索機能.....	27
6.ファイル操作.....	29
6.1.ファイルの新規作成.....	29
6.2.ファイル入出力.....	30
6.3.ファイルの情報取得.....	35
7.文字列処理.....	37
7.1.設定系関数.....	37
7.2.取得系関数.....	42
7.3.行全体関係.....	44

8.表示設定.....	48
8.1.強調表示の設定.....	48
8.2.フォントの設定.....	55
8.3.行アイコンの表示.....	58
8.4.数値設定.....	62
8.5.見えているものの取得.....	64
8.6.そのほか設定系.....	65
9.イベントの監視.....	69
9.1.イベントとは.....	69
9.2.イベントを登録するには.....	69
9.3.イベントを登録するための関数.....	71

# 1. 最初に

## 1.1. ご挨拶

さて、難産でしたがようやく二作目が仕上がりました。仕様変更次ぐ仕様変更で收拾がつかなくなってしまう前作 Footy と完全に決別し、全てを書き直した Footy2 を完成させることが出来ました。今回は Microsoft.net フレームワーク向けのラッパライブラリや、WindowsMobile 版の Footy2CE.dll も用意してのラインナップになります。あなたのアプリケーションをより華やかにできればと思います。

## 1.2. 使用条件

Footy シリーズはシェアウェアでしたが、諸般の事情によりフリーウェアとなりました。そのため個人、法人そのほか団体など形態を問わず自由にそのソフトウェアに組み込むことができます。Footy シリーズがアプリケーションに組み込まれていることを明記する必要はありませんが、してもかまいません。そのあたりは利用者に任せられています。

Footy2 を利用してソフトウェアを作成した場合、その販売、配布形態は作成者が自由に設定できるものとします。つまり、フリーソフトウェアであっても、シェアウェアであっても、パッケージングソフトであっても構いません。Footy2 を利用してソフトウェアを作成した場合、ソフトウェアに Footy2.dll およびその付属品を添付して再配布することが可能です。

他のフリーソフトウェアなどと同じくして、Footy2 はバグ修正やサポートを恒久的に行う保証はありません。また、このライブラリおよびこの文書が原因により発生した、直接的あるいは間接的な問題はいかなる場合であったとしても作者は責任を負いません。全て自己の責任において利用してください。特に Footy を利用してパッケージングソフトウェアを作成することを検討している企業の方々はこの点に留意してください。高度信頼性が求められる現場(医療、兵器開発、金融、原子力関係ほか、人命、地球環境あるいは財産に対して多大なる影響力を持ち絶対的な安定性が求められる現場)では使用されないことを勧めます。

## 1.3. オープンソース

Footy2 エディタコントロールは、Subversion 運用のオープンソースソフトウェアとして提供されています。どなたでも開発に参加することが可能ですので、興味のある方はご連絡ください。

<http://www.hpp.be:8080/trac/Footy/>

## 1.4. このドキュメントの更新履歴

---

年月日	執筆時最新 Footy Ver	備考
2007.07.07	Ver2.000	初稿
2007.08.20	Ver2.000	フリーウェア化に伴う変更、そのほか細かい修正
2007.09.04	Ver2.010	新しいバージョンに伴う修正
2007.09.18	Ver2.012	新しいバージョンに伴う修正 検索フラグに対する記載漏れがあったので追記
2008.12.28	Ver2.015	新しいバージョンに伴う修正
2009.01.11	Ver2.016	新しいバージョンに伴う修正
2009.03.16	Ver2.017	新しいバージョンに伴う修正 一部誤記があったので修正
2009.07.10	Ver2.018	新しいバージョンに伴う修正
2009.08.06	Ver2.019	新しいバージョンに伴う修正

表 1:ドキュメント更新履歴

## 2. Unicode と文書管理

エディタを扱う以上避けて通れないのが文字コードとの絡みです。ここではFooty2が標準で利用するUnicodeに対する理解を深めていただき、Unicodeとどのようにつきあったらよいのかを解説します。

### 2.1. 文字コードとUnicode

コンピューターは文字も数字として扱っています。その数字に文字を割り当ててあたかも文字であるかのように振る舞っているに過ぎません。コンピューターが発展したアメリカではご存じのように英語が使われており、アルファベットやそのほか特殊な記号を付加したとしてもさほど多くの種類はないため、7bitあれば全ての文字を表現することが出来ました。この表現方法をASCIIコードと呼びます(8bitを一文字とするという記述が多少見受けられますが、元来のASCIIコードは7bitが基本です)。ところが、世界中にコンピューターが普及するようになると日本や中国など特にアジア圏の大量に文字が存在する国や地域でビット幅が足りないという事態が起こり始めました。各国はそれぞれ独自の文字コードを制定し、様々な規格が乱立してしまいました。

そこで、UnicodeコンソーシアムはUnicodeという規格を用いて文章表現を行う方法を提案しました。このUnicodeを用いれば、様々な国の言語を統一的に使用することが出来ます。Windows 2000やWindows XPも内部的には全てUnicodeを使用して作られています。Footy2は今回、全ての内部処理をUnicodeで行うように仕様が大幅に変更されました。サロゲートペアにも対応し、充実したテキスト編集環境を提供することが出来るでしょう。

### 2.2. Unicode とUnicode以外の文字コード

Footy2は内部的に全てUnicode(UTF-16 LE)を使用するようにプログラムされています。しかし、Unicodeでない文書ファイルは今でも大量に存在しており、それらのファイルを読み込む機会が数多く存在することでしょう。そこで、Footy2は内部に文字コード自動判別と文字コード変換機能を組み込みました。これによりファイルの読み込みや保存を行うときにそれらを有効に使用することが出来ます。ユーザーがファイルを編集する場合を例にとって考えてみましょう。ファイルを編集する作業の流れは以下のようになります。

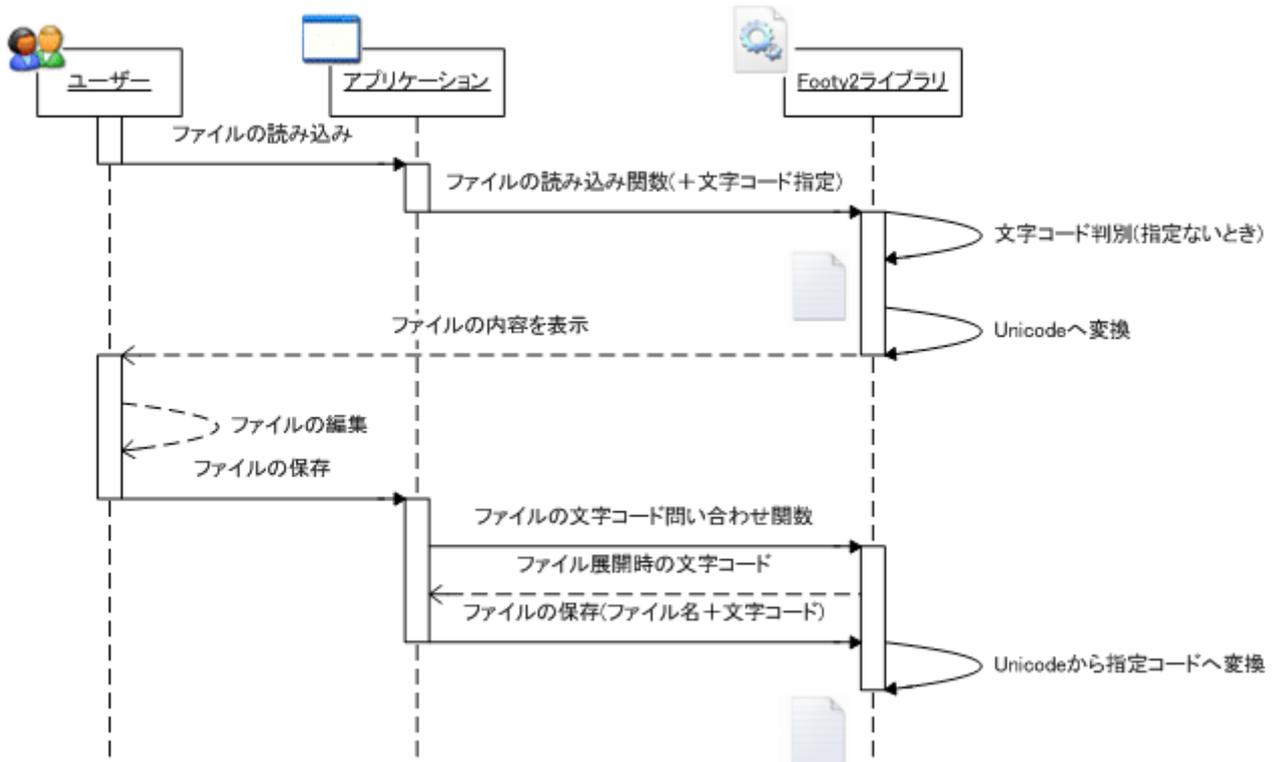


図 1:ファイル読み込みフロー

### 2.3. 文字コードの自動判別について

文字コードの自動判別機能が Footy には搭載されていますが、文字コードの自動判別は完璧でないことをご理解ください。これは、各文字コード間の仕様に使用されている文字コードの範囲が被っている箇所があるためです。

### 2.4. BOM について

BOM(Byte Order Mark)とは、Unicode 系の文字コードにおいて、そのファイルがどのようなエンコーディング形式で書かれているのかを定義するものです。通常はファイルの先頭の数バイトにこの情報を埋め込みます。この情報を埋め込むことで多くの場合、正常にファイルの形式を認識させることができます。Microsoft が提供する多くのソフトウェアは、この BOM が無いと正常にファイルを扱うことが出来ない可能性がありますのでなるべく付加しておくことをおすすめします。ただし、一部古いソフトウェアには BOM をつけることで逆にエラーとなる可能性があります(ファイル先頭に予期しない数バイトが存在するため)。特にテキストエディタなどを作成する場合は BOM を頭につけるかつかないかはユーザーに選択させるのが最良の設計といえます。

各定数において、後ろに BOM とついている場合は BOM を付加して書き込み処理を行います。ついていない場合は BOM をつけません。読み込みのときに文字コードを指定する場合、BOM の定数を使用しても使用しなくてもどちらでも正常に読み込むことが可能です。

## 2.5. ANSI 系関数と WIDE 系関数について

---

Footy2 では内部処理を Unicode で処理しているため、文字列を扱うすべての関数を ANSI 系関数と WIDE 系関数として提供しています。

※WindowsMobile 環境では、WindowsMobile の仕様上 ANSI 系関数を用意していません。すべて WIDE 系関数を使用することになります。

ANSI 系関数は、文字列の表現に ASCII 文字列と ShiftJIS を使用します。また、文字列を関数の間で受け渡すときに *const char\**型や *char\**型を使用することになります。今までのソフトウェアを Footy2 に移行するときには便利な機能といえるでしょう。WIDE 系関数を利用する場合は受け渡す文字列の表現に Unicode(UTF-16)を利用します。このときには *const wchar\_t\**型や *wchar\_t\**型を利用することになります。なお、Footy2 は内部的に Unicode 処理を行っていますので、当然変換処理が必要である ANSI 系関数よりも WIDE 系関数の方がメモリの消費も少なく高速です。もしもこれからソフトウェアを開発されるようでしたら多言語が使える有利な WIDE 系関数を利用することを強く推奨いたします。

ANSI 系関数は関数名の最後に A が、WIDE 系関数の関数名の最後には W がついているのが特徴的で、これは WindowsSDK がそのような命名規則になっているのに対応しています。また、C 言語では `_UNICODE` が宣言(*#define*)されているかいないかによって使用する関数を動的に変化させることも可能です。詳しくは付属のヘッダファイルをご覧ください。

## 3. バージョンの取得

### 3.1. バージョンの取得関数

---

このFooty2 ライブラリのバージョン情報を取得するための処理です。

#### 3.1.1. GetFooty2Ver

---

Footy2 のバージョンを取得します。値には 1000 がかけられた状態で返ってきます。たとえば Ver2.000 ならば 2000 が返ってきます。引数はありません。

#### 3.1.2. GetFooty2BetaVer

---

Footy2 のβバージョンを取得します。β版とは、そのバージョンがテスト目的で頒布されていることを示します。引数はなく、戻り値にはβ版の番号が返ってきます。たとえばβ1であれば1が返ります。もしもそれがβ版でない場合には代わりに0が返ります。

## 4. ウィンドウ操作

様々な機能はあれど、まずはエディタウィンドウが出ないことには話が始まりません。この章では、エディタコントロールの内部的な扱い、およびウィンドウの操作を行う関数について解説します。

### 4.1. Footy の ID 番号

Footy2 は、複数のコントロールを生成することが可能なライブラリです。一つのアプリケーション上で複数のエディタコントロールを扱うことが出来ます。それら一つ一つを識別するための数値が ID 番号 (*IDentifier*) です。これらの ID 番号は Footy エディタコントロールを生成するときに自動的に発行されますので、それをアプリケーション側が管理し、必要に応じて ID 番号とともに関数を呼び出す必要があります。Footy2 ライブラリが提供する関数は基本的にすべてこの ID 番号を第一引数に取りますので非常に重要な数値であると言えます。

### 4.2. Footy の生成手順

Footy2 コントロールに限らず、チャイルドコントロールを生成するときは手順に沿って生成しなければエラーとなる可能性があります。まず、チャイルドコントロールを生成する時には親ウィンドウが必要となります。親ウィンドウとは、そのコントロールを乗せるためのウィンドウのことです。これが、確実に生成された状態で呼び出さなければなりません。具体的には、ウィンドウプロシージャのコールバック関数で WM\_CREATE が呼ばれたときに確実にウィンドウが生成されている状態になります。そこで、アプリケーション起動時に Footy を生成するためには以下のように関数を呼び出します。

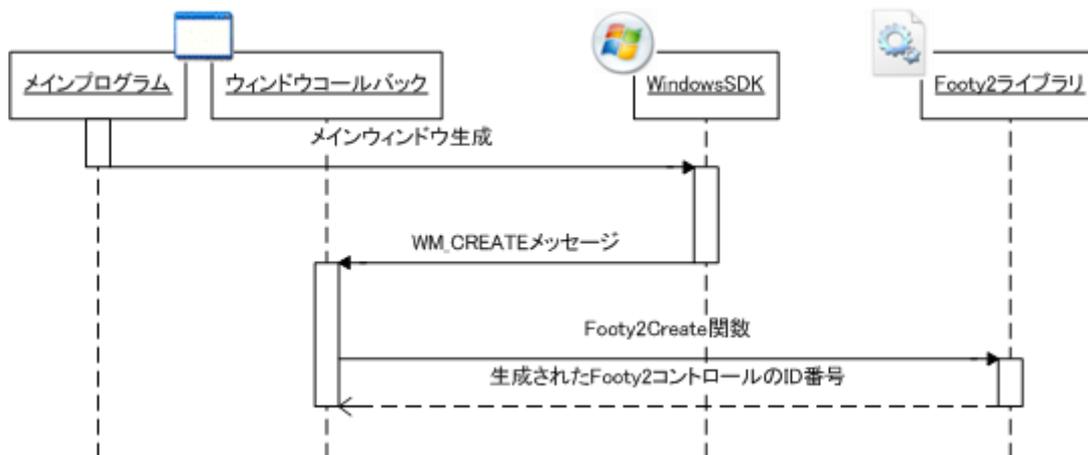


図 2:Footy 生成手順

### 4.3. Footy の生成に関わる関数

それでは、実際に生成するための関数を紹介していきます。

#### 4.3.1. Footy2Create

Footy コントロールを生成するための関数です。上記のように WM\_CREATE が呼び出されたあと、親ウィンドウハンドルが適切な状態であることを確認した上で呼び出してください。

名前	型	意味
hWnd	HWND	親ウィンドウのウィンドウハンドル。
x	int	生成する(親ウィンドウからの相対位置)x 座標。ピクセル単位。
y	int	生成する(親ウィンドウからの相対位置)y 座標。ピクセル単位。
dx	int	コントロールの幅。ピクセル単位。
dy	int	コントロールの高さ。ピクセル単位。
nMode	int	初期ビュー状態。下記参照。

表 2:Footy2Create関数の引数

Footy コントロールはビュー状態を指定できる機能があります。ビュー状態機能を利用することで、一つのドキュメントを複数のペインに分割して表示することが可能になります。以下の定数の中でどの状態で起動するかを選択することが出来ます。

値	意味
VIEWMODE_NORMAL	通常の状態。ひとつのドキュメントに一つのペインが表示されます。
VIEWMODE_VERTICAL	縦に分割して左右二つのペインを表示します。
VIEWMODE_HORIZONTAL	横に分割して上下二つのペインを表示します。
VIEWMODE_QUAD	上下左右に四分割して4個のペインを表示します。
VIEWMODE_INVISIBLE	Footy エディタコントロールを非表示にします。

表 3:ビュー状態一覧

Footy2Create 関数は成功すると作成された Footy の ID 番号を戻り値として返します。これは必ず 0 以上の数になり、基本的には生成された順序に従って 0 から順番についていきます(ただし、途中でコントロールが削除されるなどの影響で必ずしもこの法則は当てはまらないことがありますので、戻り値はアプリケーション側で保存するようにしてください)。もしも、失敗した場合は以下の定数を返します。

値	意味
FOOTY2ERR_ARGUMENT	引数のうちのいずれかに不備があったため関数が失敗した。 具体的には、hWnd が NULL である、dx または dy が負の値である。
FOOTY2ERR_MEMORY	メモリ不足により生成できなかった

表 4:Footy2Create関数の戻り値

#### 4.3.2. Footy2ChangeParent

この関数は Footy2 Ver2.015 より新規で追加された関数です。

Footy コントロールを、ドキュメントの内容を失うことなく、親ウィンドウだけ変更します。異なるウィンドウに載せ替える場合に利用してください。

名前	型	意味
hWnd	HWND	新しい親ウィンドウのウィンドウハンドル。

x	int	生成する(親ウィンドウからの相対位置)x座標。ピクセル単位。
y	int	生成する(親ウィンドウからの相対位置)y座標。ピクセル単位。
dx	int	コントロールの幅。ピクセル単位。
dy	int	コントロールの高さ。ピクセル単位。

表 5:Footy2ChangeParent関数の引数

もしも、失敗した場合は以下の定数を返します。

値	意味
FOOTY2ERR_NOID	指定された Footy の ID 番号は存在しなかった

表 6:Footy2ChangeParent関数の戻り値

#### 4.3.3. Footy2Delete

生成していた Footy コントロールを削除します。すべてのエディタ状態、アンドウバッファ、設定なども含めてすべて削除され、使用していたメモリを解放します。

名前	型	意味
nID	int	削除したい Footy の ID

表 7:Footy2Deleteの引数

この関数は以下の戻り値を返します。

値	意味
FOOTY2ERR_NONE	エラーなしで関数が成功した
FOOTY2ERR_NOID	指定された Footy の ID 番号は存在しなかった

表 8:Footy2Delete関数の戻り値

アプリケーション開発者はこの関数を明示的に呼び出してやることでメモリの消費量を抑えてやる事が出来ます。たとえば、ユーザーが文書を閉じるように命じた場合は非表示にするのではなくこの関数で削除してやるほうが有効です。

アプリケーション終了時にこの関数を呼ばなければメモリリークが発生するということはありません。アプリケーションが終了する直前、DLL はデタッチされたということを検知することが出来ます。Footy2.dll はデタッチされたことを検出すると自動的に確保されたすべてのメモリを解放して削除しますのでメモリリークは発生しません。

## 4.4. Footy のウィンドウを操作するための関数

### 4.4.1. Footy2Move

一度生成された Footy コントロールの位置や大きさを変更するための関数です。たとえば、メインウィンドウの大きさが変更されたとき(WM\_SIZE メッセージが送られたとき)、それに対応してメインウィンドウの大きさと Footy エディタコントロールの位置を合わせてやるという使い方が可能です。

名前	型	意味
nID	int	移動する Footy の ID 番号
x	int	移動先の(親ウィンドウからの相対位置)x 座標。ピクセル単位。
y	int	移動先の(親ウィンドウからの相対位置)y 座標。ピクセル単位。
dx	int	コントロールの幅。ピクセル単位。
dy	int	コントロールの高さ。ピクセル単位。

表 9:Footy2Move関数の引数

この関数は以下の戻り値を返します。

値	意味
FOOTY2ERR_NONE	エラーなしで関数が成功した
FOOTY2ERR_NOID	指定された Footy の ID 番号は存在しなかった

FOOTY2ERR_ARGUMENT	幅または高さが0以下だった
--------------------	---------------

表 10:Footy2Move関数の戻り値

#### 4.4.2. Footy2ChangeView

この関数では、Footy コントロールのビュー状態(11ページ参照)を変更させる処理を行います。この関数を使用することでコントロールを分割したり非表示にしたりすることが可能です。

名前	型	意味
nID	int	変更する Footy の ID 番号
nView	int	変更後のビュー状態。詳しくは 11 ページ参照。

表 11:Footy2ChangeViewの引数

この関数は以下の関数を返します。

値	意味
FOOTY2ERR_NONE	エラーなしで関数が成功した
FOOTY2ERR_NOID	指定された Footy の ID 番号は存在しなかった

表 12:Footy2ChangeView の戻り値

#### 4.4.3. Footy2SetFocus

指定した Footy エディタコントロールに対してフォーカスを与える処理を行います。フォーカスを与えられたウィンドウはキー入力を受け付けることが出来、Footy エディタコントロールではキャレットが表示されるようになります。

名前	型	意味
nID	int	変更する Footy の ID 番号
nViewID	int	どのペインにフォーカスを与えるのかを示すビュー ID 値。数値の意味は現在設定されているビュー状態(11ページ参照)によって変化する。

表 13:Footy2SetFuncFocusの引数

nViewID の値は以下のように設定します。

ビュー状態	0	1	2	3
VIEWMODE_NORMAL	メインのペイン	使用しません	使用しません	使用しません
VIEWMODE_VERTICAL	左のペイン	右のペイン	使用しません	使用しません
VIEWMODE_HORIZONTAL	上のペイン	下のペイン	使用しません	使用しません
VIEWMODE_QUAD	左上のペイン	右上のペイン	左下のペイン	右下のペイン
VIEWMODE_INVISIBLE	使用しません	使用しません	使用しません	使用しません

表 14:ビュー状態と nViewID

この関数は以下の値を返します。

値	意味
FOOTY2ERR_NONE	エラーなしで関数が成功した
FOOTY2ERR_NOID	指定された Footy の ID 番号は存在しなかった
FOOTY2ERR_ARGUMENT	与えられた nViewID の値が不正(0~3 ではない)

表 15:Footy2SetFocusの返り値

#### 4.4.4. Footy2GetWnd

---

この関数はFooty エディタコントロールのウィンドウハンドルを取得する関数です。この関数によってウィンドウハンドルを取得することが出来、この値を Windows に渡すことで様々な動作をさせることが可能ですが、やることによってはFooty 全体が不安定になったりクラッシュする可能性があります。十分にFooty の内部挙動を理解した上でお使いください。

名前	型	意味
nID	int	取得する Footy の ID 番号
nViewID	int	どのビュー ID(16ページ参照)のウィンドウハンドルを取得するの かの数値

表 16:Footy2GetWnd関数の引数

この関数が成功した場合、該当するウィンドウへのハンドルが返ります。もし該当する Footy の ID 番号が見つからない場合、該当するビュー IDが見つからない場合は *NULL* が返ります。

#### 4.4.5. Footy2GetWndVSplit

---

Footy2ChangeView 関数(15ページを参照)などを使用して分割表示を行ったときに使用される垂直分割バーコントロールのウィンドウハンドルを取得する処理です。注意事項としては上記と同様で、挙動を理解した上で利用しなければアプリケーションが不安定になる可能性があります。

名前	型	意味
nID	int	取得する Footy の ID 番号

表 17:Footy2GetWndVSplitの引数

この関数は、成功したときに垂直分割バーのウィンドウハンドルを返します。Footy の ID が不正である場合には *NULL* を返します。

#### 4.4.6. Footy2GetWndHSplit

---

上記 Footy2GetWndVSplit 関数とほぼ同等の機能と引数を持ちますが、こちらは垂直(*Vertical*)ではなく水平(*Horizontal*)バーを返します。

#### 4.4.7. Footy2GetActiveView

---

現在フォーカスを持っているアクティブなビュー ID(16ページ参照)を返す処理です。

名前	型	意味
nID	int	取得する Footy の ID 番号

表 18 : Footy2GetActiveView の引数

この関数は以下の値を返します。

値	意味
FOOTY2ERR_NOID	指定された Footy の ID 番号は存在しなかった
FOOTY2ERR_NOACTIVE	アクティブなビューはなかった(どれもフォーカスを失っている)

表 19:Footy2GetActiveViewの戻り値

もしもエラーなしで正常終了した場合はアクティブ状態にあるビュー ID が返ります。

#### 4.4.8. Footy2Refresh

---

この関数は Footy エディタコントロール全体(すべてのペイン)を再描画する処理を行います。エディタを再描画するとは、一度コントロール全体を背景色で塗りつぶした上でユーザーが今まで入力

していた文書を適切な表示位置へ書き直すことを示します。このFooty2Refresh 関数に限らず再描画処理は非常に重たい処理であり、多用するとアプリケーションのレスポンスを著しく低下させてしまいますのでご注意ください。

描画設定系の関数には再描画するかどうかのフラグを指定することが可能なものがあります。この再描画フラグを *false* として設定してやると再描画処理が行われません。複数の設定を連続で行う場合、その都度再描画を行っているユーザーがとても待たされてしまいますので複数行うすべての設定処理では再描画を行わないように設定しておき、最後の最後でこのFooty2Refresh 関数を呼び出すとレスポンスよく様々な設定処理を行うことができます。

この関数は以下の引数を持ちます。

名前	型	意味
nID	int	再描画する Footy の ID 番号

表 20:Footy2Refresh関数

この関数は以下の返り値を返します。

値	意味
FOOTY2ERR_NONE	エラーなしで関数が成功した
FOOTY2ERR_NOID	指定された Footy の ID 番号は存在しなかった

表 21:Footy2Refresh関数の戻り値

## 5. 編集コマンド

### 5.1. 編集コマンドとは

編集コマンドとは、通常のテキストエディタのメニューの「編集」の中に含まれるコマンドを指します。クリップボードからテキストを取得してcaret位置に挿入する「貼り付け」機能や、選択している部分をクリップボードへ保存する「コピー機能」を備えます。これらの機能は、全てFootyテキストエディタコントロールの関数を一つ呼び出すだけで簡単に利用することが出来ます。ユーザーが独自にルーチンを作成することも可能ですが、これらの機能を利用すれば開発の手間を省くことが出来るでしょう。

この章で解説する関数は他の章で解説する関数に比べると必要な操作も多くなく、また単純ですので開発者はすぐにこれらの機能をアプリケーションに組み込むことが出来るかと思います。

### 5.2. クリップボード関連機能

クリップボードに関する機能に「コピー」「切り取り」「貼り付け」があります。これらの機能は全てUnicodeとして動きます。例えば、ユーザーがInternet Explorerで韓国のページを見ていたとします。その中からハングル文字をコピーしたあとで、Footy2テキストエディタ上で貼り付け処理を行えば、そのままハングル文字を貼り付けることが出来ます。Unicodeに対応したエディタですので、このような事が可能なのです。ここで紹介するクリップボード機能を使用すれば、すぐにこれらの機能が実現できます。開発者は自分で実装しようとせずこれらの関数を利用することを推奨します。

#### 5.2.1. Footy2Copy

Footy2Copy関数はユーザーが現在選択しているテキストをクリップボードにUnicodeとしてコピーする関数です。クリップボードにコピーしたデータは他のアプリケーションでも使用することが出来ます。Windowsでは特別な理由が無い限りCtrl+Cキーを割り当てるように推奨されています。

名前	型	意味
nID	int	選択しているテキストをクリップボードへコピーしたいFootyのID

表 22:Footy2Copyの引数

値	意味
FOOTY2ERR_NONE	エラーなしで正常終了した
FOOTY2ERR_NOID	指定された Footy2 の ID 番号が見つからなかった
FOOTY2ERR_NOTSELECTED	テキストが選択されていないため操作が完了できなかった
FOOTY2ERR_MEMORY	メモリー不足が原因で操作が完了できなかった

表 23:Footy2Copyの返り値

### 5.2.2. Footy2Cut

Footy2Cut 関数はユーザーが現在選択しているテキストをクリップボードに Unicode として切り取る関数です。選択されていたテキストはコピーご削除されます。この機能を利用してテキストを削除した場合は Footy2Undo 関数(23ページ参照)によるアンドゥ操作の対象になります。クリップボードにコピーしたデータは他のアプリケーションでも使用することが出来ます。Windows では特別な理由が無い限り Ctrl+X キーを割り当てるように推奨されています。

名前	型	意味
nID	int	選択しているテキストをクリップボードへ切り取りたい Footy の ID

表 24:Footy2Cutの引数

値	意味
FOOTY2ERR_NONE	エラーなしで正常終了した
FOOTY2ERR_NOID	指定された Footy2 の ID 番号が見つからなかった
FOOTY2ERR_NOTSELECTED	テキストが選択されていないため操作が完了できなかった
FOOTY2ERR_MEMORY	メモリー不足が原因で操作が完了できなかった

表 25:Footy2Cutの返り値

### 5.2.3. Footy2Paste

Footy2Paste 関数はクリップボードから Unicode テキストを取得して現在のcaret位置へ貼

り付けます。また、このときにユーザーがテキストを選択していた場合は削除されてしまいます。この機能を利用してテキストを挿入、あるいは置換した場合はFooty2Undo 関数(23ページ参照)によるアンドウ操作の対象になります。クリップボードにコピーしたデータは他のアプリケーションでも使用することが出来ます。Windows では特別な理由が無い限り Ctrl+V キーを割り当てるように推奨されています。

名前	型	意味
nID	int	クリップボードからテキストを貼り付けたいFooty のID

表 26:Footy2Pasteの引数

値	意味
FOOTY2ERR_NONE	エラーなしで正常終了した
FOOTY2ERR_NOID	指定されたFooty2 のID 番号が見つからなかった
FOOTY2ERR_MEMORY	メモリー不足が原因で操作が完了できなかった

表 27:Footy2Pasteの返回值

### 5.3. アンドウ、リドウの機能

アンドウ(*Undo* : 元に戻す)、リドウ(*Redo* : やり直す)機能はテキストエディタの中でも重要視される機能のひとつです。アンドウは、直前に行った動作の一つを取り消して前の状態に戻す機能です。例えば、テキストを削除したあとでアンドウを行えば削除したテキストが復元されますし、逆に文字を入力してからアンドウを行えば入力されたテキストは削除されます。アンドウを行った事自体をキャンセルするにはリドウ機能を使用します。このアンドウとリドウ機能は共にメモリが許す限り無限に使用することが可能です。

また、アンドウとリドウ機能を装備した場合、テキストが編集されている状態であるかの判別が難しくなる場合があります。通常のテキストエディタでは、文字を入力した情報をアンドウして無かったことにしても「編集されている」という状態が維持されてしまい、エディタを終了しようとしたときにメッセージが発せられる事があります("この文章は編集されていますが保存しますか?"等)。Foot2 テキストエディタコントロールでは、アンドウ、リドウの状況まで考慮にいれてテキストが編集されているかどうかの状態を取得する事が出来ます。

### 5.3.1. Footy2Undo

---

Footy2Undo 関数はユーザーが行ったキーボード、マウス操作、あるいは関数呼び出しなどが原因で変更されたテキストを元に戻すための関数です。Footy2Cut 関数や Footy2Paste 関数もアンドウの対象になります。Windows では特別な理由が無い限り Ctrl+Z キーを割り当てるように推奨されています。

名前	型	意味
nID	int	アンドウを行いたい Footy の ID

表 28:Footy2Undoの引数

値	意味
FOOTY2ERR_NONE	エラーなしで正常終了した
FOOTY2ERR_NOID	指定された Footy2 の ID 番号が見つからなかった
FOOTY2ERR_NOUNDO	アンドウ操作がこれ以上行えなかった。

表 29:Footy2Undoの戻り値

### 5.3.2. Footy2Redo

---

Footy2Redo 関数は Footy2Undo 関数によってアンドウされたことを取り消し、もう一度編集された状態に戻します。たとえば、ユーザーがテキストを挿入したとします。この状態で Footy2Redo 関数を呼び出しても何も意味をなしません。ここで、Footy2Undo 関数を呼び出すと、ユーザーが入力したテキストは削除(アンドウ)されます。アンドウされたあとで、Footy2Redo 関数を呼ぶと、アンドウが無かったことになり、ユーザーが入力したテキストは復元されます。同様のことが削除など、ユーザーが行ったそのほかのテキスト編集操作にも当てはまります。Windows では、この操作を Ctrl+Y または Ctrl+Shift+Z のいずれかに対応させるように推奨されています。

名前	型	意味
nID	int	リドゥを行いたい Footy の ID

表 30:Footy2Redoの引数

値	意味
FOOTY2ERR_NONE	エラーなしで正常終了した
FOOTY2ERR_NOID	指定された Footy2 の ID 番号が見つからなかった
FOOTY2ERR_NOUNDO	リドゥ操作がこれ以上行えなかった。

表 31:Footy2Redoの戻り値

### 5.3.3. Footy2DeleteKey

この関数をコールすると、ただ単に Delete キーが押されたことになります。編集メニューに「削除」メニューを配置し、手動で呼び出す必要があるときに利用してください。

名前	型	意味
nID	int	リドゥを行いたい Footy の ID

表 32:Footy2DeleteKeyの引数

値	意味
FOOTY2ERR_NONE	エラーなしで正常終了した
FOOTY2ERR_NOID	指定された Footy2 の ID 番号が見つからなかった

表 33:Footy2DeleteKeyの戻り値

### 5.3.4. Footy2IsEdited

Footy2IsEdited 関数は、前回のセーブポイントと比較してテキストが編集されているかどうかを取得する関数です。この関数が true だった場合はメッセージボックスなどを表示して保存を促したりすることが出来ます。Footy2IsEdited 関数でセーブポイントとして扱われるのは以下の場面です。

- Footy2Create 関数(11ページ参照)が呼ばれた直後
- Footy2SetText 関数(40ページ参照)が呼ばれ、テキストがすべて変化した直後
- Footy2TextFromFile 関数(32ページ参照)が呼ばれ、テキストの内容がすべてファイルのデー

タで置き換わった直後

- Footy2SaveToFile 関数(35ページ参照)が呼ばれ、テキストの内容がファイルに保存された直後

これらの状態が最終のセーブポイントとなり、そのセーブポイントと比べて変化しているかどうか、をチェックします。たとえば、一度保存したあと(ここがセーブポイントになります)、ユーザーが文字列を入力します(セーブポイントと比べてテキストが変化しています)。そこで Footy2Undo 関数(23ページ参照)を呼び出して入力したテキストをアンドウします(セーブポイントと同様の状態に戻ったので Footy2IsEdited は *false* が返ります)。再び Footy2Redo 関数(23ページ参照)が呼び出され、テキストが復元された場合、Footy2IsEdited は *true* が返ります。ただし、アンドウ、リドゥに関してはすべてのテキストの状態をメモリに保存しているわけではありません。そのため、テキストを入力したとしてもユーザーがアンドウせずに自力でバックスペースキーなどで削除してファイルが保存された状態と比べて完全に同じ状態に戻したとしても Footy2IsEdited 関数は *true* を返してしまう点に注意してください。

名前	型	意味
nID	int	編集されているかどうかを調べたい Footy の ID 番号

表 34:Footy2IsEdited の引数

値	意味
<i>true</i>	テキストが編集されている
<i>false</i>	テキストは前回のセーブポイントから比べて編集されていない、あるいは指定された Footy2 の ID 番号が見つからなかった

表 35:Footy2IsEditedの返り値

## 5.4. 選択のための機能

### 5.4.1. Footy2SelectAll

この機能は、表示されている全てのテキストを選択状態にする機能です。

名前	型	意味
nID	int	全て選択したいFootyのID
bRedraw	bool	再描画処理を行うかどうか(規定値=true)

表 36:Footy2SelectAllの引数

この関数は以下の戻り値を返します。

値	意味
FOOTY2ERR_NONE	エラーなしで正常終了した
FOOTY2ERR_NOID	指定されたFooty2のID番号が見つからなかった

表 37:Footy2SelectAllの戻り値

#### 5.4.2. Footy2ShiftLock

この関数を使用すると、ユーザーがシフトキーを押すことなく常にシフトキーが押された状態となります。WindowsMobileなどのキー操作に難がある端末でこの機能を活用するとユーザビリティに優れたアプリケーションを構築できるかもしれません。デフォルトでは無効になっています。

名前	型	意味
nID	int	シフトロックの設定をしたいFootyのID
bLocked	bool	シフトロックを行うかどうか ( <i>true</i> = 行う / <i>false</i> = 行わない)

表 38:Footy2ShiftLockの引数

この関数は以下の値を返します。

値	意味
FOOTY2ERR_NONE	エラーなしで正常終了した

FOOTY2ERR_NOID	指定された Footy2 の ID 番号が見つからなかった
----------------	-------------------------------

表 39:Footy2ShiftLockの戻り値

### 5.4.3. Footy2IsShiftLocked

この関数は、Footy2ShiftLock 関数によってシフトキーがロックされているかどうかを取得するための関数です。以下の引数を持ちます。

名前	型	意味
nID	int	シフトロックされているかどうかを取得したい Footy の ID

表 40:Footy2IsShiftLockedの引数

この関数は以下の戻り値を返します。

値	意味
<i>true</i>	シフトロックされている
<i>false</i>	シフトロックされていない、あるいは指定された Footy2 の ID 番号が見つからなかった

表 41:Footy2IsShiftLockedの戻り値

## 5.5. 編集禁止設定

### 5.5.1. Footy2SetReadOnly

この関数は Footy2 Ver2.016 より新規で追加された関数です。この関数を利用すると、エディタ上でテキストの変更が可能かどうかを設定することが出来ます。

名前	型	意味
----	---	----

nID	int	編集可能かどうかを設定したい Footy の ID
bReadOnly	bool	true のときに、編集を無効化します。 false のときに、編集を許可します(デフォルト)。

表 42:Footy2SetReadOnlyの引数

この関数は以下の戻り値を返します。

値	意味
FOOTY2ERR_NONE	エラーなしで正常終了した
FOOTY2ERR_NOID	指定された Footy2 の ID 番号が見つからなかった

表 43:Footy2SetReadOnlyの戻り値

### 5.5.2. Footy2IsReadOnly

この関数は [Footy2 Ver2.016](#) より新規で追加された関数です。この関数を利用すると、エディタ上でテキストの変更が可能かどうかを取得することが出来ます。

名前	型	意味
nID	int	編集可能かどうかを設定したい Footy の ID

表 44:Footy2IsReadOnlyの引数

この関数は以下の戻り値を返します。

値	意味
true	編集禁止状態に設定されています
false	編集禁止状態ではありません、または、指定された Footy2 の ID 番号が見つかりません。

## 5.6. 検索機能

### 5.6.1. Footy2Search

エディタの多くは検索機能を持っています。これは、指定されたテキストを編集集中の文書から探し出す処理になります。初代 Footy では搭載されていませんでしたが、多くのアプリケーションが必要な機能ということで Footy2 から DLL 内部に検索機能が実装されました。アプリケーションによって実装すると Footy からテキストを全て吸い出す処理のオーバーヘッドが発生してしまったため今までは検索処理をある程度までしか高速化することが出来ませんでした。今回からは内部的に検索処理を実行しますので高速です。

この関数は ANSI 版と WIDE 版があり(詳しくは 8 ページ参照)、szText 引数の型がそれによって変化しますので注意してください。ANSI 版は Footy2SearchA、WIDE 版は Footy2SearchW という名前になります。

名前	型	意味
nID	int	検索処理を実行したい Footy の ID 番号
szText	const wchar_t* const char*	検索を行いたい文字列
nFlags	int	フラグを OR 結合したもの(以下を参照)

表 46:Footy2Search の引数

指定したフラグによって検索するとき様々な効果を発揮することが出来ます。以下の値を nFlags 引数に対して OR 結合してください。

名前	意味
SEARCH_FROMCURSOR	指定されているとき、現在のカーソル位置から検索を開始します。もしもこのフラグが指定されていないときは、文書の先頭か末尾から検索が開始されます。先頭か末尾かの指定は後述する

	SEARCH_BACK が指定されているか否かによって選択されます。
SEARCH_BACK	後ろ方向に検索処理を実行します。もし、このフラグが立っていないときは順方向に検索を行います。
SEARCH_NOT_REFRESH	検索終了時に再描画しません。
SEARCH_BEEP_ON_404	見つからなかったときにビープ音を鳴らします。 MessageBeep(MB_ICONEXCLAMATION)
SEARCH_NOT_ADJUST_VIEW	このフラグが立っていないとき、カーソル位置が中央に来るようにスクロール位置を調整します。
SEARCH_REGEX	正規表現( <i>Regular Expression</i> )による検索処理を行います。正規表現を使用すればより柔軟な検索処理が可能です。現在この機能は未実装です。
SEARCH_HIGHLIGHT	検索した文字列をハイライト表示にします。これによりユーザーは他のどこに検索した文字列があるかを視覚的に判断することが可能になります。現在この機能は未実装です。

表 47:nFlags の値

## 6. ファイル操作

前回の Footy では、DLL の関数としてファイル入出力は実装されていませんでしたが、今回の Footy2 ではエディタ全体が Unicode で処理されているということもあり、ファイル入出力の処理が複雑になり、アプリケーション開発者に多大なる負担をかけるとの判断からライブラリ内部に実装されることになりました。この章ではそれらの関数について解説します。

### 6.1. ファイルの新規作成

#### 6.1.1. Footy2CreateNew

この関数は、ファイルの「新規作成」に相当する部分の処理を行います。具体的には以下の処理が行われます。

- エディタのバッファ内容はすべて削除されます(文書が消えます)。
- アンドウのバッファ内容はすべて削除されます(Footy2Undo関数でアンドウできなくなります)。
- この状態がセーブポイントとして登録されます(Footy2IsEdited関数、24 ページ参照)。
- 設定されたすべての強調表示情報は削除されます(一切強調表示されなくなります)。
- 読み込まれたファイルのエンコード情報は削除されます。

この関数は以下の引数をとります。

名前	型	意味
nID	int	初期化したい Footy の ID 番号

表 48:Footy2CreateNew関数の引数

この関数は以下の値を返します。

値	意味
FOOTY2ERR_NONE	エラーなしで正常終了した
FOOTY2ERR_NOID	指定された Footy2 の ID 番号が見つからなかった

表 49:Footy2CreateNewの戻り値

## 6.2. ファイル入出力

この項目では、ファイルの入出力に関する関数を取り扱います。これらの関数を使用するときは図 1:ファイル読み込みフロー(7ページ)も併せてご覧ください。

### 6.2.1. Footy2TextFromFile

この関数によってテキストファイルからデータを読み込み、その内容をエディタに表示します。このとき、テキスト状態が読み込まれたときにアンドウバッファがすべて削除されますのでご注意ください。この関数は文字列を扱うため ANSI 版と WIDE 版があります(詳細は 8 ページ参照)。それぞれ *Footy2TextFromFileA* と *Footy2TextFromFileW* となり、*szPath* 引数の型が変化します。この関数は以下の引数を取ります。

名前	型	意味
nID	int	ファイルを読み込みたい Footy の ID 番号
szPath	const char* const wchar_t*	読み込みたいファイル(フルパスまたはカレントディレクトリからの相対パス)
nCharSets	int	読み込みたいファイルの文字コード設定

表 50:Footy2TextFromFileの引数

この中で、*nCharSets* 引数は以下の値を設定します。なお、`_BOM` で終了する定数は、BOM (Byte Order Mark)をつける設定、ないものについては BOM をつけない設定になります。詳細については 7 ページの「BOM について」をご覧ください。

定数名	文字コード名	解説
CSM_AUTOMATIC	自動判別	Footy2 に搭載された独自の自動判別エンジンによってファイルを解析し、適切な文字コードを利用します。ただし、この判別は完璧なものではありません。 <b>なお現状では ISO8859 系の自動判別は出来ません。</b>
CSM_PLATFORM	プラットフォーム依存	プラットフォームのデフォルトエンコード形式を指定します。標準的な日本版 Windows では、ShiftJIS(Codepage932)が利用されるはずですが、内部処理的には MultiByteToWideChar という API を使用して変換しています。なお、WindowsMobile 版ではこの値を利用することはできません。これは仕様です。
CSM_SHIFT_JIS_2004	Shift_JIS-2004	Windows Vista 環境において標準の文字コード。日本工業規格 JIS X 0213:2004 を使用できるように Shift_JIS を拡張したものです。一部の文字は Unicode の BMP 領域にマッピングされませんが Footy2 ならば自由に全ての文字を扱うことが出来ます(ただし、組み合わせ文字を除く)。
CSM_EUCJP_MS	EUC-JIS-2004	EUC( <i>Extended Unix Code</i> )-JP の拡張で、日本工業規格 JIS X 0213:2004 を使用できるようにしたものです。Footy2 は Internet Explorer7 ですら扱うことが困難な EUC-JP の 3バイト文字に対応しています。本来は EUC-JIS-2004 の規格外ですが互換性保持のため半角カナ文字にも対応しています。
CSM_UTF8 CSM_UTF8_BOM	UTF-8 (BOMなし、あり)	Unicode で使用できる符号化方式の一つ、UTF-8( <i>Unicode Translation Format-8</i> )。XML( <i>eXtensible Markup Language</i> )などを中心に多くのファイルで利用される符号化です。この符号化方式では、一文字を 1~6 バイトで表します。ただし、日本では漢字一文字に 3 バイトを使用することからファイルサイズが大きくなりがちな点に注意してください。純粋な ASCII 文字のみで構成すれば全く同じバイト配列となります。Footy2 ではサ

		<p>ロゲートペアを用いて表現可能な1文字1~4バイトの文字を扱うことが可能です。</p>
<p>CSM_UTF16_LE CSM_UTF16_LE_BOM</p>	<p>UTF-16 リトルエンディアン (BOMなし、あり)</p>	<p>Unicodeで利用できる符号化方式の一つ、UTF-16'(Unicode Translation Format-16)の、リトルエンディアンです。1文字を2バイト(16ビット)によって表す符号化方式であったはずでしたが、結局それだけでは世界中の文字を収めることは不可能であり、サロゲートペアという組を使用して表示するという可変長のUTF-16方式が誕生しました。Footy2ライブラリはこのサロゲートペアに対応しているため完全な状態でUTF16を扱うことが可能です(ただしフォントが必要)。この方式は、16ビットをリトルエンディアン(計算機的にバイトの順序を逆に処理していく)方式で、現在一般的に普及している多くのIntelおよびその互換プロセッサで効率よく文字を扱えるメリットがあります。この符号化方式はとても見分けがつきにくいのでBOMを挿入することを強く推奨します。</p>
<p>CSM_UTF16_BE CSM_UTF16_BE_BOM</p>	<p>UTF-16 ビッグエンディアン (BOMなし、あり)</p>	<p>上記UTF16のビッグエンディアン版です。UTF16リトルエンディアンと基本的には同じですが、奇数番地のバイトと偶数番地のバイトがひっくり返っているのが特徴です。これはSunのSPARCプロセッサなどビッグエンディアンで計算を行うマシンで効率よくファイルを扱えるメリットがあります。同様に、見分けがつきにくいのでBOMを挿入することを推奨します。</p>
<p>CSM_UTF32_LE CSM_UTF32_LE_BOM</p>	<p>UTF-32 リトルエンディアン (BOMなし、あり)</p>	<p>1文字4バイト固定で扱うことが出来るUnicodeの符号化方式です。Footy2ではUTF16のサロゲートペアで扱うことが可能な範囲であるU+0~U+10FFFFまでを利用することが出来ます。それ以外の文字は無視されます。</p>
<p>CSM_UTF32_BE CSM_UTF32_BE_BOM</p>	<p>UTF-32 ビッグエンディアン (BOMなし、あり)</p>	<p>上記UTF-32のビッグエンディアン版です。</p>

CSM_ISO8859_n	ISO8859 系	ISO8859 で制定された文字コードです。1 バイト 1 文字という考え方で最上位 1 ビットが立っている番地の文字の割り振り方で文字を変えていきます。n の中には 1～16 の値が入りますが、そのなかで ISO8859-12 は 1997 年に破棄されたため Footy2 ライブラリでは対応していません。
---------------	-----------	---

表 51:nCharSets の値

ファイルの改行コードについては、指定することが出来ませんが、これについては完全に自動判別を行います。CR、LF、CR と LF の組み合わせの三種類に現在対応しています。

この関数は以下の返り値を返します。

値	意味
FOOTY2ERR_NONE	エラーなしで正常終了した
FOOTY2ERR_NOID	指定された Footy2 の ID 番号が見つからなかった
FOOTY2ERR_404	ファイルが見つからなかったため関数が失敗した。あるいは、ファイルを展開するパーミッション(権限)が現在アプリケーションを実行中のユーザーにはない(制限ユーザーが他人のマインドキュメントを覗こうとしている、など)。内部的には fread 関数が失敗したらこの関数が返ります。
FOOTY2ERR_MEMORY	メモリ不足のため関数が実行できなかった
FOOTY2ERR_ENCODER	ファイルの文字コード自動判別に失敗した(バイナリである可能性が高い、または未対応の文字コード)。手動で設定してください

表 52:Footy2TextFromFileの戻り値

### 6.2.2. Footy2SaveToFile

現在のテキスト状態をファイルへの書き込む処理です。同様に ANSI 版と WIDE 版があり(8ページ参照)、それぞれ Footy2SaveToFileA と Footy2SaveToFileW になります。

名前	型	意味
nID	int	保存したい Footy の ID 番号
szPath	const char* const wchar_t*	書き込みたいファイル名(フルパスまたはカレントディレクトリからの相対パス)
nCharSets	int	保存するときの文字コード
nLineMode	int	保存するときの改行コード

表 53:Footy2SaveToFileの引数

nCharSets には保存したい文字コードを設定します。詳しくは 32 ページを参照してください。なお、これらの値の中で CSM\_AUTOMATIC を選んだ場合は以下のように処理されます。

- もしも、それ以前にそのコントロールに Footy2TextFromFile 関数でファイルが読み込まれていた場合、その読み込まれたときに設定されていた文字コード(もしもそのときも CSM\_AUTOMATIC によって自動判別されていた場合は、判別した結果の文字コード)を利用する。
- もしも、ファイルが読み込まれていない場合はデフォルトとして設定されている文字コードによって保存する。Windows 環境では CSM\_PLATFORM が選ばれた状態となり、WindowsMobile では CSM\_PLATFORM が利用できないため CSM\_SHIFT\_JIS\_2004 がデフォルトとなる。

nLineMode 引数は以下の値をとれます。

値	意味
LM_AUTOMATIC	自動的に改行コードを設定します。そのエディタで Footy2TextFromFile が読み出されていればそのときに設定された改行コードが書き込まれます。もしも Footy2TextFromFile で読み出されていない場合はデフォルトの改行コードである LM_CRLF が

	自動的に設定されます。
LM_CRLF	改行コードを CR+LF の組み合わせで表現します。Windows では標準です。
LM_CR	改行コードを CR(キャリッジリターン)のみで表現します。Mac で標準です。
LM_LF	改行コードを LF(ラインフィード)のみで表現します。Linux など UNIX 系のオペレーティングシステムでは標準です。

表 54:nLineMode の引数

この関数は以下の戻り値を返します。

値	意味
FOOTY2ERR_NONE	エラーなしで正常終了した
FOOTY2ERR_NOID	指定された Footy2 の ID 番号が見つからなかった
FOOTY2ERR_404	ファイルが書き込めなかった。あるいは、ファイルを書き込むパーミッション(権限)が現在アプリケーションを実行中のユーザーにはない(制限ユーザーが Program Files へ書き込みしようとする、など)。内部的には fread 関数が失敗したらこの関数が返ります。
FOOTY2ERR_MEMORY	メモリ不足のため関数が実行できなかった

表 55:Footy2SaveToFile の戻り値

## 6.3. ファイルの情報取得

### 6.3.1. Footy2GetCharSet

指定された Footy エディタコントロールの文字コードセットを取得します。この関数は、Footy2SaveToFile 関数において CSM\_AUTOMATIC(自動設定保存)を選択したときに利用される文字コードを取得します。つまり、そのエディタにおいて Footy2TextFromFile 関数を用いてファイルを読み込んだときは、その読み込んだときの文字コードが返り、何も操作を行っていないときはデフォ

ルトの値が返ります。

この関数は以下の引数をとります。

名前	型	意味
nID	int	値を取得したい Footy の ID 番号

表 56:Footy2GetCharSetの引数

この関数は以下の値を返します。

値	意味
CSM_ERROR	指定された Footy の ID 番号が見つからなかった
CSM_foobar で定義される値	設定されている文字コードの値

表 57:Footy2GetCharSetの戻り値

### 6.3.2. Footy2GetLineCode

---

この関数は上記と同様ですが改行コードを取得する処理になります。引数は同様ですが戻り値が以下のうちのいずれかを返します。

値	意味
LM_ERROR	指定された Footy の ID 番号が見つからなかった
LM_foobar で定義される値	設定されている改行コードの値

表 58:Footy2GetCharSetの戻り値

## 7. 文字列処理

文字列処理関数を利用すると、ユーザーに対して補助的な機能(定型文挿入など)を提供したり、ユーザーに入力させた文章をアプリケーションが処理させたりといった様々なことを行うことができます。ここではそれらの機能について解説します。

### 7.1. 設定系関数

#### 7.1.1. Footy2SetSelText

この関数を使用すると、選択しているテキストを指定された文字列で置き換えることができます。選択しているテキストがなかった場合はキャレット位置に指定された文字列が挿入されることとなります。この関数はFooty2Undo 関数(23ページ参照)を使用することで動作を取り消すことが可能です。この関数は文字列を扱うため ANSI と WIDE の二種類の関数(8ページ参照)が用意され、それぞれ Footy2SetSelTextA と Footy2SetSelTextW という名前になっています。この関数は以下の引数をとります。

名前	型	意味
nID	int	文字列挿入を行いたい Footy の ID 番号
pString	const char* const wchar_t*	挿入する文字列

表 59:Footy2SetSelTextの引数

この関数は以下の戻り値を持ちます。

値	意味
FOOTY2ERR_NONE	エラーなしで正常終了した
FOOTY2ERR_ARGUMENT	pString が NULL である
FOOTY2ERR_NOID	指定された Footy の ID が見つからない
FOOTY2ERR_MEMORY	メモリ不足のため操作を完了できなかった

表 60:Footy2SetSelTextの戻り値

### 7.1.2. Footy2SetText

この関数は指定された Footy エディタコントロール内部にあるテキストをすべて更新します。この操作はアンドウバッファを削除し、そこへセーブポイント(Footy2IsEdited関数を参照、24 ページ)を設定し、設定されていた文字コードをデフォルトへ戻します。この挙動が嫌な場合(アンドウバッファを残したい場合)は、Footy2SelectAll 関数(25ページ)ですべての文字列を選択した上で前述 Footy2SetSelText 関数を使用してください。この関数は文字列を扱うため ANSI 版と WIDE 版があり、それぞれ Footy2SetTextA と Footy2SetTextW という関数名になっています。

名前	型	意味
nID	int	文字を設定したい Footy の ID 番号
pString	const char* const wchar_t*	設定したい文字列

表 61:Footy2SetTextの引数

この関数の戻り値は前述 Footy2SetSelText 関数と同等なので割愛します。

### 7.1.3. Footy2GetCaretPosition

この関数は現在caretがある位置を取得するための関数です。

名前	型	意味
nID	int	caret位置を取得したい Footy の ID 番号
pCaretLine	int*	caretがある行を取得するための変数へのポインタ
pCaretPos	int*	caretがある桁を取得するための変数へのポインタ

表 62:Footy2GetCaretPositionの引数

この関数が終了したあと、pCaretLine、pCaretPosとして指定された変数はFooty2GetCaretPosition関数の内部で値が書き換えられ、それぞれcaretがある行と桁の数值が代入されます。この値はすべて0ベースであり、1行目が0としてカウントされ、1桁目が0としてカウントされます。値を取得したくない場合はこれら各ポイントへの値としてNULLを指定することも出来ます。

値	意味
FOOTY2ERR_NONE	エラーなしで正常終了した
FOOTY2ERR_NOID	指定されたFootyのIDが見つからない

表 63:Footy2GetCaretPositionの戻り値

#### 7.1.4. Footy2SetCaretPosition

この関数は指定された位置へcaretを移動する処理です。もしもテキストが選択されていた場合は選択が解除されます。

名前	型	意味
nID	int	caret位置を設定したいFootyのID番号
nCaretLine	int	caretの移動先行番号(0ベース、1行目が0になります)
nCaretPos	int	caretの移動先桁番号(0ベース、1桁目が0になります)
bRefresh	bool	設定時にエディタを再描画するかどうか(規定値はtrueです)

表 64:Footy2SetCaretPositionの引数

nCaretLine、nCaretPosとして設定する場合、Footy2GetLines関数(50ページ)やFooty2GetLineLength関数(49ページ)などを用いて適切な値の範囲を知ることが出来ます。この関数は以下の戻り値を返します。

値	意味
FOOTY2ERR_NONE	エラーなしで正常終了した

FOOTY2ERR_NOID	指定された Footy の ID が見つからない
FOOTY2ERR_ARGUMENT	引数が不正だったため関数が失敗した

表 65:Footy2SetCaretPositionの戻り値

### 7.1.5. Footy2GetSel

この関数は指定された Footy エディタの選択状態を取得する関数です。

名前	型	意味
nID	int	選択位置を取得したい Footy の ID 番号
pStartLine	int*	選択開始位置の行番号を取得するためのポインタ
pStartPos	int*	選択開始位置の桁番号を取得するためのポインタ
pEndLine	int*	選択終了位置の行番号を取得するためのポインタ
pEndPos	int*	選択終了位置の桁番号を取得するためのポインタ

表 66:Footy2GetSel関数の引数

この関数は pStartLine、pStartPos、pEndLine、pEndPos に指定された変数に対して関数内部で適切な値を設定します。取得したくない場合はこれらの値として `NULL` を指定することも出来ます。なお、設定される値は前述の Footy2GetCaretPosition 関数と同様にすべて 0 ベースとなります。この関数は以下の値を返します。

値	意味
FOOTY2ERR_NONE	エラーなしで正常終了した
FOOTY2ERR_NOID	指定された Footy の ID が見つからない
FOOTY2ERR_NOTSELECTED	テキストが選択されていないため関数が失敗した(Ver2.012以降、 <b>戻り値が変更になりましたのでご注意ください</b> )

表 67:Footy2GetSelの戻り値

## 7.1.6. Footy2SetSel

この関数はテキストを選択するための処理です。

名前	型	意味
nID	int	選択位置を取得したいFootyのID番号
nStartLine	int	選択開始位置の行番号(0ベース、1行目は0としてカウントします)
nStartPos	int	選択開始位置の桁番号(0ベース、1桁目は0としてカウントします)
nEndLine	int	選択終了位置の行番号(0ベース、1行目は0としてカウントします)
nEndPos	int	選択終了位置の桁番号(0ベース、1桁目は0としてカウントします)
bRefresh	bool	関数終了時に再描画処理を行うかどうか(規定値 <i>true</i> )

表 68:Footy2SetSel関数の引数

この関数で指定される nEndLine と nEndPos の位置にcaret位置が移動します。bRefresh が *true* のときはエディタ全体が再描画されます。連続して再描画が必要となるような処理を呼び出す場合は bRefresh を *false* にしてください。この関数は以下の値を返します。

値	意味
FOOTY2ERR_NONE	エラーなしで正常終了した
FOOTY2ERR_NOID	指定されたFootyのIDが見つからない
FOOTY2ERR_ARGUMENT	引数が適切ではない

表 69:Footy2SetSel関数の戻り値

## 7.2. 取得系関数

### 7.2.1. Footy2GetTextLength

文字列の長さを取得します。この関数は文字列をとりませんが ANSI 版と WIDE 版があり(8ページ参照)、それぞれ返す値が違うので注意してください。それぞれ Footy2GetTextLengthA と Footy2GetTextLengthW と名前が付けられています。この関数は以下の引数を持ちます。

名前	型	意味
nID	int	文字を設定したい Footy の ID 番号
nLineMode	int	改行コードをどのように扱うのか(後述)。値は Footy2SaveToFile 関数(35ページ)を参照してください。

表 70:Footy2GetTextLengthの引数

この関数の主目的はテキストを取得するときのバッファを計算することです。そのため、厳密には文字列の長さを取得するというものではありません。ANSI 版の関数を呼び出した場合、マルチバイト文字(日本語など)は2とカウントされてしまいますし、WIDE 版でも Unicode のサロゲートペア文字は2とカウントされてしまいます。これは仕様です。

nLineMode で渡す引数は、改行コードを何文字として計算するかという設定です。LM\_CRLF ならば2、LM\_CR や LM\_LF が設定されれば1として計上されます。LM\_AUTOMATIC の使用も可能です。この関数は以下の値を返します。

値	意味
FOOTY2ERR_NOID	指定された Footy の ID 番号が見つからなかった
0 以上の整数	エディタに入っている文字列の長さ

表 71:Footy2GetTextLengthの戻り値

この関数を使用して取得した値を *nValue* とすると、全体のテキストを取得するために必要なバッファサイズは以下の通りになります。

```
ANSI : sizeof(char) * (nValue + 1)
```

```
WIDE : sizeof(wchar_t) * (nValue + 1)
```

バッファサイズには終端文字(NULL文字)を付加する必要があるので純粋に nValue だけではだめであることに注意してください。また、使用する関数が ANSI 版であるか WIDE 版であるかで計算式が変わってくることに注意してください。

### 7.2.2. Footy2GetSelLength

この関数は Footy Ver2.010 以降でのみ使用可能です。Footy2GetTextLength とほぼ同等ですが、こちらは選択している文字列の長さを取得します。WIDE 版と ANSI 版があります。

名前	型	意味
nID	int	文字を設定したい Footy の ID 番号
nLineMode	int	改行コードをどのように扱うのか(後述)。値は Footy2SaveToFile 関数(35ページ)を参照してください。

表 72:Footy2GetSelLengthの引数

注意点や使用方法についても Footy2GetTextLength を参照してください。この関数は以下の値を返します。

値	意味
FOOTY2ERR_NOID	指定された Footy の ID 番号が見つからなかった
FOOTY2ERR_NOTSELECTED	選択されていない
FOOTY2ERR_MEMORY	ワーク用メモリが不足していた
0 より大きな整数	エディタに入っている文字列の長さ

表 73:Footy2GetSelLengthの戻り値

### 7.2.3. Footy2GetText



#### 注意

この関数は適切にバッファサイズの確保および設定を行わないまま呼び出すとアプリケーションがクラッシュするだけでなくバッファオーバーフローによるセキュリティホールの原因になります。

この関数は指定されたバッファにエディタの内容を書き込みます。この関数は必ず以下のフローに従って利用してください。適切にバッファを確保しなければバッファオーバーフローが発生してしまう危険性があります。

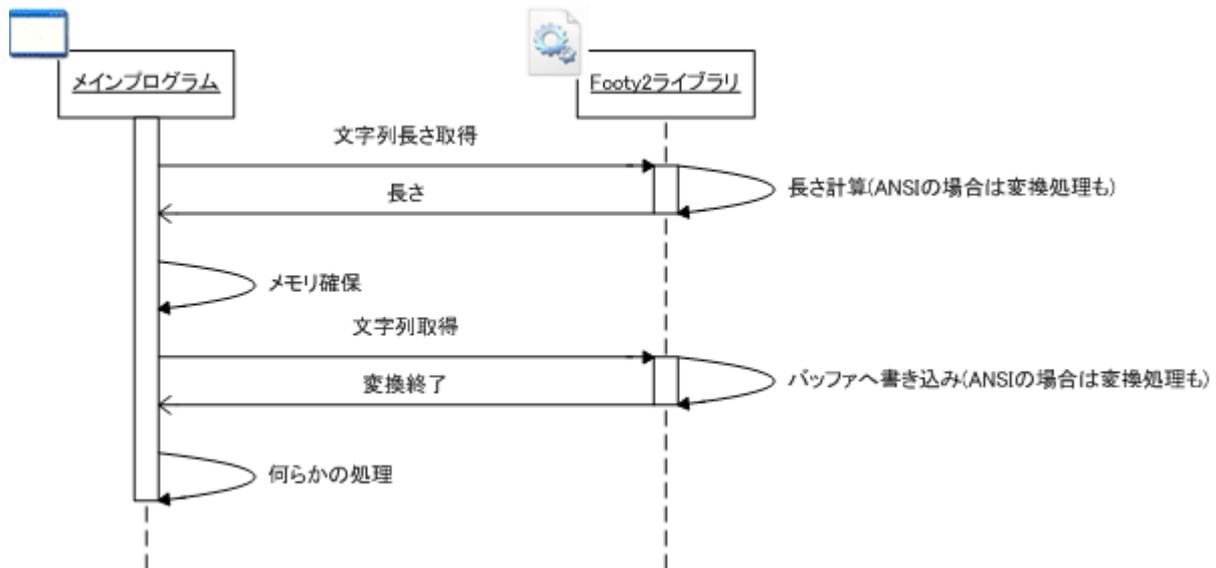


図 3: 文字列取得フロー

この関数は、文字列を扱うので ANSI 版と WIDE 版が存在します(8ページ参照)。どちらを使うかによってバッファを確保する方法が変わってきますのでご注意ください(詳しくは Footy2GetTextLength 関数、44 ページを参照)。この関数は以下の引数を持ちます。

名前	型	意味
nID	int	文字を設定したい Footy の ID 番号
pString	char*	書き込み先のバッファの先頭へのポインタ

	wchar_t*	
nLineMode	int	改行コードをどのように扱うのか(後述)。値は Footy2SaveToFile 関数(35ページ)を参照してください。
nSize	int	バッファへ書き込み可能なサイズ(バイト数ではなく配列数)

表 74:Footy2GetTextの引数

ここで、nSize は配列数であることに注意してください。ANSI 版では配列数とバイト数が一致しますが、WIDE 版では配列数はバイト数の半分になります。この関数は以下の値を返します。

値	意味
FOOTY2ERR_NONE	エラーなしで正常終了した
FOOTY2ERR_ARGUMENT	pString が NULL である、nSize が負である、など引数が不正
FOOTY2ERR_NOID	指定された Footy の ID が見つからない

表 75:Footy2GetTextの戻り値

#### 7.2.4. Footy2GetSelText

この関数は Ver2.010 以降でのみ利用可能です。Footy2GetText とほぼ同等の引数を持っていますが、こちらは選択している文字列が返されます。注意事項も同等ですので、詳しくは Footy2GetText をご覧ください。

### 7.3. 行全体関係

ここで紹介する関数は、Footy エディタコントロールが保持する行データへ直接アクセスが出来る関数です。

#### 7.3.1. Footy2GetLineW

この関数は行データを直接取得し、ポインタへのアクセスが可能な関数です。ただし、この関数を利用して得たポインタに対して直接操作などを行うと誤動作を招く可能性がありますので、あくまでもデータを取得するのみにしておいてください。この関数は Footy2GetText 関数などを利用してテ

キストへアクセスするよりも圧倒的に高速です。この関数は以下の引数を持ちます。

名前	型	意味
nID	int	文字を設定したい Footy の ID 番号
nLine	size_t	行番号(0 ベースです。1 行目を 0 としてカウントしてください)

表 76:Footy2GetLineW 関数の引数

この関数は以下の戻り値を返します。

値	意味
NULL	指定された Footy の ID 番号が見つからない、あるいは指定された行番号の行が見つからない
const wchar_t*型のポインタ	指定された行番号の行データの先頭へのポインタ

表 77:Footy2GetLineWの戻り値

### 7.3.2. Footy2GetLineA



#### 注意

この関数は適切にバッファサイズの確保および設定を行わないまま呼び出すとアプリケーションがクラッシュするだけでなくバッファオーバーフローによるセキュリティホールの原因になります。

行データを直接取得するためには、Footy エディタコントロールが内部処理で使用している Unicode を利用した方が賢明ですが、とりあえず ANSI 版の関数も用意しました。WIDE 版とは引数が大きく異なり、使用しにくいものとなっています。この関数はバッファを利用して行データを取得することになります。バッファサイズの計算には後述の Footy2GetLineLength 関数を利用してください。

名前	型	意味
nID	int	文字を設定したいFootyのID番号
nLine	size_t	行番号(0ベースです。1行目を0としてカウントしてください)
pString	const char*	データを取得するためのバッファ
nSize	int	バッファのサイズ

表 78:Footy2GetLineAの引数

この関数は以下の戻り値を返します。

値	意味
FOOTY2ERR_NONE	エラーなしで正常終了した
FOOTY2ERR_ARGUMENT	pStringがNULLである、nLineの値がおかしいなど引数の不正
FOOTY2ERR_NOID	指定されたFootyのIDが見つからない

表 79:Footy2GetLineAの戻り値

### 7.3.3. Footy2GetLineLength

この関数は指定した行の長さを取得する関数です。この関数はANSI版とWIDE版の二種類があり(8ページ参照)、それぞれ返す長さが変わります。それぞれFooty2GetLineLengthAとFooty2GetLineLengthWという名前になっています。引数に行番号がつく以外はFooty2GetTextLength関数(44ページ参照)と同様です。

名前	型	意味
nID	int	長さを取得したいFootyのID
nLine	size_t	行番号(0ベースです。1行目を0としてカウントしてください)

表 80:Footy2GetLineLengthの引数

この関数は以下の値を返します。

値	意味
FOOTY2ERR_NOID	指定された Footy の ID 番号が見つからなかった
FOOTY2ERR_ARGUMENT	指定された行番号の行が見つからなかった
0 以上の整数	指定された行の長さ

表 81:Footy2GetLineLengthの戻り値

### 7.3.4. Footy2GetLines

この関数は指定された Footy コントロールに含まれる行の数を取得する関数です。以下の引数を取ります。

名前	型	意味
nID	int	行の数を取得したい Footy の ID

表 82:Footy2GetLinesの引数

以下の戻り値を返します。

値	意味
FOOTY2ERR_NOID	指定された Footy の ID 番号が見つからなかった
0 以上の整数	エディタに入っている行の数

表 83:Footy2GetLinesの戻り値

## 8. 表示設定

表示設定項目では、Footy2 コントロールの表示状態を設定することが出来ます。表示状態を設定することで、ユーザーに対してグラフィカルで分かりやすい編集環境を提供することが出来るでしょう。

### 8.1. 強調表示の設定

強調表示を設定すると、特殊な文書(プログラミングやスクリプトなど)を編集するアプリケーションを作成する上で大きく有利になります。強調表示設定は、指定された語句や、指定された語句の間などだけの色を変えることで、文書を見やすく効率的に編集できるようになります。以下の流れにそって設定処理を行ってください。

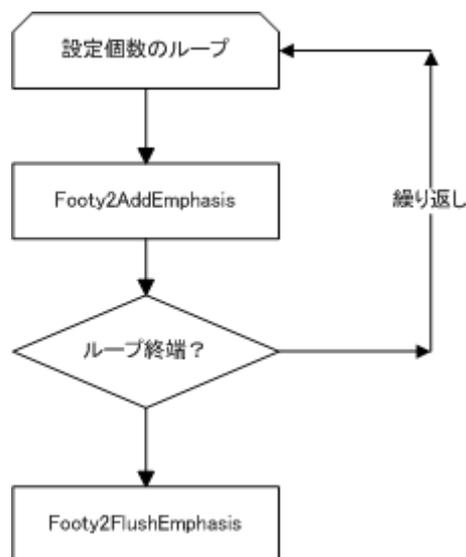


図 4: 強調表示設定手順

#### 8.1.1. Footy2AddEmphasis

この関数は指定された Footy コントロールに対して強調表示の文字を設定する関数です。これを一度だけ呼び出しただけでは適用されません。最終的には後述する Footy2FlushEmphasis 関数(57ページ参照)を呼び出してすべての強調表示文字を確定してやる必要があります。また、Footy2AddEmphasis 関数を呼び出しただけでは再描画されません。この関数は ANSI 版と WIDE 版があり(8ページ参照)、それぞれ Footy2AddEmphasisA と Footy2AddEmphasisW 関数という名前になっています。pString1 と pString2 の型が const char\*か const wchar\_t\*型という違いがあります。

この関数は以下の引数を取ります。

名前	型	意味
nID	int	強調表示設定したい Footy の ID
pString1	const char* const wchar_t*	設定する文字列 1(設定文字列、あるいは開始文字列)。NULL 設定不可
pString2	const char* const wchar_t*	設定する文字列 2(終端文字列)。フラグによっては NULL 可。
nType	int	どのように色分け設定を行うのか
nFlags	int	フラグとして設定されている値の OR 結合。わかりにくい場合は 0 を設定するとよい。
nLevel	int	設定される文字の色分けレベル(1~31)。わかりにくい場合は 1 を設定するとよい。
nPermission	int	この色分けがどのレベルの色分け最中で実行されるのかを示す。わかりにくい場合は場合は PERMIT_LEVEL(0)を設定するとよい。
nIndependence	int	単語独立度合い。わかりにくい場合は EMP_IND_ALLOW_ALL を設定するとよい。
nColor	COLORREF	色分けしたときの色。RGB マクロで設定可能

表 84:Footy2AddEmphasis関数の引数

この関数はかなり複雑で詳細に設定することが可能です。まず、pString1 と pString2 の値はそれぞれ色分けするための文字列を設定します。これは前述のように ANSI 版と WIDE 版で引数が異なります。この値の意味は nType で設定された色分け種類によって変化します。NType は以下の値をとることができます。

値	意味
EMP_WORD	単語モード、 pString1 の文字列のみを色分け。 pString2 は無視

EMP_LINE_AFTER	行末モード、pString1以降、行末までを色分け。pString2は無視。
EMP_LINE_BETWEEN	単一行間モード。pString1～pString2までを色分け、同一行に限る
EMP_MULTI_BETWEEN	複数行間モード。pString1～pString2までを色分け

表 85:nTypeの値

nFlagsには、色分けのフラグ設定を行います。フラグとしては以下の値が設定でき、すべてOR結合で組み合わせることが可能です。何も指定したくない場合は0を入れてください。

値	意味
EMPFLAG_BOLD	強調表示するときには太字にする
EMPFLAG_NON_CS	大文字と小文字の区別を行わない。レスポンスが悪くなるので多用しないように。
EMPFLAG_HEAD	行頭にあるときのみ色分け処理を実行する

表 86:nFlagsのフラグ一覧

次に現れるnLevelは、この色分けで使用される色分けレベルを設定します。レベルは、0～31までの値を使用することができ、この値に大小の関係はありません。この値はnPermissionと組み合わせて使用して、他の語句との優先順位を設定することになります。設定する際は似通った語句を同じレベルとして設定しておいてください。たとえば、C言語の色分けを行う場合、以下のように設定しておくのが理想的です。

レベル	含まれる語句
1	int, float, double, char, …など
2	#define, #ifdef, #ifndef, #endif, …など
3	if, for, else, while, …など

表 87:nLevelの例

次に現れるnPermissionでは、どのような状況においてこの語句を強調表示するのかという設定

を行います。nType に EMP\_WORD 以外の文字列を指定した場合、他に強調表示した文字が色分けの最中に出現するということが起こりえます。たとえば C 言語の色分け設定では、以下のような状況です。

```
int nDefaultVaule;      //このnDefaultValue という値はint型です。
printf("this is int value¥n");
```

上記の例では、ユーザーはテキストが非常に見にくくなってしまいます。コメント中に現れている int という語句が、色分けで設定されているため、コメントであるにもかかわらず色分けされてしまっているからです。また、下の行ではダブルクォートで囲まれている本来は文字列として認識すべき箇所まで int という色分けがなされており、これも非常に見えにくくなっています。本来、この文書は

```
int nDefaultVaule;      //このnDefaultValue という値はint型です。
printf("this is int value¥n");
```

と表示されるべきです。つまり、int を色分けするときには「この色分けは他に何も色分けをしていない最中のみ色分けする」という設定が必要であることが分かるかと思います。「何も色分けされていない状態」というのは Footy2 ライブラリではレベル 0 として定義されています。他のどのような色分けであってもレベル 0 に設定することは不可能であるので、レベル 0 と設定すれば「地の文」である状態であるという見分けがつかず、引数 nPermission には、PERMIT\_LEVEL マクロを用いて設定処理を行います。つまり、地の文のみを色分けしたい場合は

```
Footy2AddEmphasis(..(省略)...,PERMIT_LEVEL(0),...);
```

と設定すればよいこととなります。レベル 0 の他にレベル 1 の最中でも色分けを許可したい場合は、これを変更して

```
Footy2AddEmphasis(..(省略)...,PERMIT_LEVEL(0) | PERMIT_LEVEL(1),...);
```

フラグのように OR で結合してやれば、レベル 0 またはレベル 1 のときに色分けするという設定

になります。これらの機能を使いこなすことで、「HTML 文書において、タグの中だけ a や href を色分けする」などの高度な設定も可能になります。どのような状況であっても色分けを行うという設定は nPermission 引数に -1 を入れてください。

nIndependence には、独立性を設定できます。独立性によって、色分けするかしないかの判断を前後の文字からも設定することが出来ます。たとえば、C 言語の色分け設定時に以下のような状況が発生する可能性があります。

```
int internationalValue; //国際的に取引されている…
```

どこにでもあるありふれた変数名 international などの最初には C 言語の型名である int が含まれており、これによって色分けが起こってしまいます。こうしてしまっただけではユーザーにとってはとても不便です。nIndependence に値を設定してやり、前後にアルファベットがあるときは無視してやればこれを回避できます。

nIndependence には、以下の値の中から前後の文字として許可するものを選択し、それを OR で結合します。面倒な場合は EMP\_IND\_ALLOW\_ALL などを利用すればすべての文字を前後のものとして認めるといった設定になります。

値	意味
EMP_IND_PARENTHESIS	前後に丸括弧()があることを許可する
EMP_IND_BRACE	前後に中括弧{}があることを許可する
EMP_IND_ANGLE_BRACKET	前後に山形括弧<>があることを許可する
EMP_IND_SQUARE_BRACKET	前後に各括弧[]があることを許可する
EMP_IND_QUOTATION	前後にコーテーション"があることを許可する
EMP_IND_UNDERBAR	前後にアンダーバー_があることを許可する
EMP_IND_OPERATORS	前後に演算子 + - * / % ^ = があることを許可する
EMP_IND_OTHER_ASCII_SIGN	前述のものを除く全ての ASCII 記号 # ! \$ &   ¥ @ ? , .
EMP_IND_NUMBER	前後に(半角)数字を許可する
EMP_IND_CAPITAL_ALPHABET	前後に大文字アルファベットを許可する

EMP_IND_SMALL_ALPHABET	前後に小文字アルファベットを許可する
EMP_IND_SPACE	前後に半角スペースを許可する
EMP_IND_FULL_SPACE	前後に全角スペースを許可する
EMP_IND_TAB	前後にタブを許可する
EMP_IND_JAPANESE	日本語
EMP_IND_KOREAN	韓国語
EMP_IND_EASTUROPE	東ヨーロッパ言語
EMP_IND_OTHERS	上記以外
以下の値は、上記に挙げた定数の省略形です	
EMP_IND_ASCII_SIGN	全ての ASCII 記号列を許可する
EMP_IND_ASCII_LETTER	全ての ASCII 文字を許可する(数字とアルファベット)
EMP_IND_BLANKS	全ての空白文字列を許可する
EMP_IND_OTHER_CHARSETS	全てのキャラクタセットを許可する(日本語、韓国語…)
EMP_IND_ALLOW_ALL	全て

表 88:nIndependence として設定可能な値

この関数は以下の戻り値を返します。

値	意味
FOOTY2ERR_NOID	指定された Footy の ID 番号が見つからなかった
FOOTY2ERR_ARGUMENT	設定された引数のうちどれかがおかしい
FOOTY2ERR_MEMORY	メモリー不足により操作が完了できなかった
FOOTY2ERR_NONE	エラーなく関数が終了した

表 89:Footy2AddEmphasisの戻り値

### 8.1.2. Footy2FlushEmphasis

---

この関数は、Footy2AddEmphasis 関数によって設定された色分け情報を適用する処理です。なぜこの関数を呼び出す必要があるのかというと、Footy2AddEmphasis 関数は大量に連続して呼び出される可能性があるからです。Footy2AddEmphasis は詳細な設定が可能なので、設定されるごとに大幅に色分け設定が狂ってきます。そこで、Footy2AddEmphasis 関数を呼び出すだけでは色分け設定を保持することなく、最終的に Footy2FlushEmphasis 関数を呼び出すことで色分け情報を確定するという仕様になりました。この関数は全ての色分け情報を確定したあと、エディタ全体を再描画します。この関数には以下の値が設定できます。

名前	型	意味
nID	int	色分け情報を確定したい Footy の ID 番号

表 90:Footy2FlushEmphasis関数の引数

この関数は以下の値を返します。

値	意味
FOOTY2ERR_NOID	指定された Footy の ID 番号が見つからなかった
FOOTY2ERR_NONE	エラーなく関数が終了した

表 91:Footy2FlushEmphasis関数の戻り値

### 8.1.3. Footy2ClearEmphasis

---

この関数は、設定された色分け情報を全て破棄し、エディタを再描画する処理を行います。色分け情報を全て破棄することで色分け用に利用していたメモリを解放することが出来ます。この関数は以下の引数をとります。

名前	型	意味
nID	int	色分け情報を破棄したい Footy の ID 番号

表 92:Footy2ClearEmphasis関数の引数

この関数は以下の値を戻り値として返します。

値	意味
FOOTY2ERR_NOID	指定された Footy の ID 番号が見つからなかった
FOOTY2ERR_NONE	エラーなく関数が終了した

表 93:Footy2ClearEmphasis関数の戻り値

## 8.2. フォントの設定

Footy で表示するためのフォントを設定する処理です。

### 8.2.1. Footy2SetFontFace

この関数は、指定された種類のフォントを変更することが出来ます。この関数は ANSI 版と WIDE 版が存在し、それぞれ Footy2SetFontFaceA と Footy2SetFontFaceW という名前になっています。

名前	型	意味
nID	int	フォントを設定したい Footy の ID 番号
nFontMode	int	設定したいフォントの種類
pString	const char* const wchar_t*	フォントの名前
bRefresh	bool	再描画を行うかどうか(規定値 <i>true</i> )

表 94:Footy2SetFontFaceの引数

nFontMode は以下の値をとります。

定数名	意味	デフォルト書体名
-----	----	----------

FFM_ANSI_CHARSET	ANSI 文字の描画に利用されるフォントを設定	Courier New
FFM_BALTIC_CHARSET	バルト系言語	Courier New
FFM_BIG5_CHARSET	繁体中国語	MingLiU
FFM_EASTEUROPE_CHARSET	東ヨーロッパ	Courier New
FFM_GB2312_CHARSET	簡体中国語	SimSun
FFM_GREEK_CHARSET	ギリシャ文字	Courier New
FFM_HANGUL_CHARSET	ハングル文字	GulimChe
FFM_RUSSIAN_CHARSET	キリル文字	Courier New
FFM_SHIFTJIS_CHARSET	日本語	MS Gothic
FFM_TURKISH_CHARSET	トルコ語	Courier New
FFM_VIETNAMESE_CHARSET	ベトナム語	Courier New
FFM_ARABIC_CHARSET	アラビア語	GulimChe
FFM_HEBREW_CHARSET	ヘブライ語	Courier New
FFM_THAI_CHARSET	タイ語	Tahoma

表 95:デフォルトフォント一覧

bRefreshが *true* のときは再描画処理が行われます。一気に初期設定を行う場合は *false* に設定して、最終的に Footy2Refresh 関数(18ページ参照)を呼び出してください。この関数は以下の戻り値を返します。

値	意味
FOOTY2ERR_NOID	指定された Footy の ID 番号が見つからなかった
FOOTY2ERR_MEMORY	メモリー不足により処理が実行できなかった
FOOTY2ERR_ARGUMENT	引数のいずれかが不正、フォントが見つからなかったなど
FOOTY2ERR_NONE	エラーなく関数が終了した

### 8.2.2. Footy2SetFontSize

全体のフォントのサイズを設定する処理です。この関数を使用すると、全ての上記のタイプに関係なく全てのフォントのサイズが変更されます。

名前	型	意味
nID	int	フォントサイズを設定したいFootyのID番号
nPoint	int	フォントのサイズ(ポイント)
bRefresh	bool	再描画を行うかどうか(規定値 <i>true</i> )

表 97:Footy2SetFontSizeの引数

nPoint は一般的なフォントダイアログなどで使用されるフォントの大きさを指定しますが、これはデバイスの DPI(*Dot Per Inchi*)の値によって大きさが左右されます。通常のパソコンなどでは大きく大きさが変化することはないでしょうが、WindowsMobile 端末など解像度が極端に大きな端末で(特に W-ZERO3 のように小さい画面に解像度が極限まで高められた製品)、大きさがアプリケーション開発者の意図しないものになってしまう場合がありますので注意してください。大きさのデフォルト値は Windows 版の Footy2.dll では 11、WindowsMobile 版の Footy2CE.dll では 9 に設定されています。

bRefresh 引数は設定後再描画を行うかどうかを設定します。初期設定などでこの関数を呼び出す場合は *false* で指定しておいて最終的に Footy2Refresh 関数(18ページ参照)を呼び出して再描画してください。

この関数は以下の戻り値を返します。

値	意味
FOOTY2ERR_NOID	指定されたFootyのID番号が見つからなかった
FOOTY2ERR_ARGUMENT	引数のいずれかが不正、フォントが見つからなかったなど
FOOTY2ERR_NONE	エラーなく関数が終了した

表 98:Footy2SetFontSizeの戻り値

### 8.3. 行アイコンの表示

行アイコン機能を使用すると、行ごとにアイコンを設定することが出来るようになります。これにより、統合編集環境ではブレークポイントや現在実行している行を表示したり、通常のエディタであればブックマークを表現したりということが可能になります。この行アイコンは行番号表示領域に表示されることになり、プログラマのアイディアで幅広い用途に使える機能と言えます。たとえば、この行アイコンを設定しておき、それを元にアイコン取得関数などを使用して検索していけばブックマーク機能が簡単に実現できるでしょう。アプリケーション側で「何行目にアイコンがあるか」などを保存しておく必要は一切ありません。

アイコンは一つの行に複数設定することが出来ます。アイコンを複数設定できるように、Footy2ではアイコンの定数をフラグ式にしてOR結合で繋ぐようにして設定します。アイコンの定数としては以下の値が使用可能です。なお、以下示すアイコンのうち紫で描かれている部分は透過色になりますので実際には背景色と同化します。

定数名	アイコン	解説
LINEICON_ATTACH		クリップのようなアイコン
LINEICON_BACK		戻るアイコン
LINEICON_CHECKED		チェックボックスのチェックがはいたもの
LINEICON_UNCHECKED		チェックボックスのチェックがないもの
LINEICON_CANCEL		キャンセルアイコン(ブレークポイントなどに使えそう)
LINEICON_CLOCK		時計アイコン
LINEICON_CONTENTS		コンテンツを含むアイコン
LINEICON_DB_CANCEL		データベース用、キャンセルアイコン

LINEICON_DB_DELETE		データベース用、削除アイコン
LINEICON_DB_FIRST		データベース用、最初の項目アイコン
LINEICON_DB_EDIT		データベース用、編集アイコン
LINEICON_DB_INSERT		データベース用、追加アイコン
LINEICON_DB_LAST		データベース用、最後の項目アイコン
LINEICON_DB_NEXT		データベース用、次の項目アイコン
LINEICON_DB_POST		データベース用、チェックアイコン
LINEICON_DB_PREVIOUS		データベース用、前の項目アイコン
LINEICON_DB_REFRESH		データベース用、再描画アイコン
LINEICON_DELETE		削除アイコン
LINEICON_EXECUTE		実行アイコン
LINEICON_FAVORITE		お気に入りアイコン
LINEICON_BLUE		青色フラグアイコン
LINEICON_GREEN		緑色フラグアイコン
LINEICON_RED		赤色フラグアイコン
LINEICON_FORWARD		前へ進むアイコン
LINEICON_HELP		ヘルプアイコン
LINEICON_INFORMATION		情報アイコン
LINEICON_KEY		暗号化アイコン

LINEICON_LOCK		ロックアイコン
LINEICON_RECORD		記録アイコン
LINEICON_TICK		チェックアイコン
LINEICON_TIPS		Tips アイコン
LINEICON_WARNING		警告アイコン(コンパイルエラー時には是非)

表 99:行アイコンの一覧

### 8.3.1. Footy2SetLineIcon

この関数を使用して行アイコンを設定することが出来ます。アイコン定数と見本は上記をご覧ください。この関数は以下の引数を取ります。

名前	型	意味
nID	int	行アイコンを設定したいFootyのID番号
nLine	int	行アイコンを設定したい行番号(0ベース)
nIcons	int	アイコンのリスト(定数値 OR 結合)
bRefresh	bool	再描画を行うかどうか(規定値 <i>true</i> )

表 100:Footy2GetLineIconの引数

nLine に関しては、行番号は0ベースで設定します。1行目を0として関数に渡してください。使用可能な行番号の最大値はFooty2GetLines関数(50ページ参照)にて調べることが出来ます。アイコンのリストはOR結合で設定します。アイコンの表示を解除したい場合は専用の関数はありません。nIconsに0を指定してこの関数を呼び出すと全てのアイコンが解除されることとなります。bRefresh関数を *true* で渡すと即座に再描画処理が実行されます。初期設定などでつけたい場合は *false* を使用して最終的にFooty2Refresh関数(18ページ)を呼び出してください。この関数は以下の戻り値を返します。

値	意味
FOOTY2ERR_NOID	指定された Footy の ID 番号が見つからなかった
FOOTY2ERR_ARGUMENT	指定された行番号の行が存在しない
FOOTY2ERR_NONE	エラーなしで関数成功

表 101:Footy2SetLineIconの戻り値

### 8.3.2. Footy2GetLineIcon

この関数では、指定された行番号に設定されたアイコンを取得することが出来ます。この関数は以下の引数を持ちます。

名前	型	意味
nID	int	行アイコンを取得したい Footy の ID 番号
nLine	int	行アイコンを取得したい行番号(0 ベース)
pIcons	int*	アイコンリストを取得したい変数へのポインタ

表 102:Footy2GetLineIconの引数

pIcons には設定されたアイコンのリストが OR 結合で設定されます。このポインタは NULL を指定することも出来ますが、その場合は何も取得できません(この関数を呼び出す意味がなくなります)。このアイコンの定数は上記アイコンの見本とともに参照してください。この値の戻り値によって、特定のアイコンが含まれているかどうかを調べるためには以下のように記述します。

```
bool bBlueIncluded = ((nIconValue & LINEICON_BLUE) != 0);
```

この関数は以下の戻り値を返します。

値	意味
---	----

FOOTY2ERR_NOID	指定された Footy の ID 番号が見つからなかった
FOOTY2ERR_ARGUMENT	指定された行番号の行が存在しない
FOOTY2ERR_NONE	エラーなしで関数成功

表 103:Footy2GetLineIconの戻り値

## 8.4. 数値設定

この数値設定系関数では、数字のみに対して設定処理を行う場合の関数を紹介します。数字を設定するためだけのために関数を用意するのは開発者としても管理が大変になるのを防ぐ目的に作られています。今後拡張されていく予定です。数値として設定できるものには以下のものがあります。

値	意味
SM_LAPEL_COLUMN	折り返し位置を設定します
SM_LAPEL_MODE	折り返しモードを設定します
SM_MARK_VISIBLE	記号の表示しているものの定数 OR 結合
SM_LINENUM_WIDTH	行番号幅(0 のとき非表示、設定時負の数でデフォルトに戻す)
SM_RULER_HEIGHT	ルーラー高さ(0 のとき非表示、設定時負の数でデフォルトに戻す)
SM_UNDERLINE_VISIBLE	行下線の表示状態(0 のとき非表示、それ以外るとき表示)

表 104:数値設定で使用できる定数

SM\_MARK\_VISIBLE のとき、数値には以下の値の組み合わせが入ります。

値	意味
EDM_HALF_SPACE	半角スペース
EDM_FULL_SPACE	全角スペース
EDM_TAB	タブ文字

EDM_LINE	改行マーク
EDM_EOF	[EOF]マーク(ファイルの終端、 <i>End Of File</i> )
EDM_SHOW_ALL	フラグの省略形、全てのマーク表示
EDM_SHOW_NONE	フラグの省略形、全てのマーク非表示

表 105:SM\_MARK\_VISIBLEのマーク定数

### 8.4.1. Footy2SetMetrics

この関数は上記の定数と数字を指定することで Footy エディタコントロールの設定を行う処理です。以下の引数を取ります。

名前	型	意味
nID	int	数値を設定したい Footy の ID 番号
nObject	int	設定する箇所(上記定数が入る)
nValue	int	設定する数値(nObject によって意味が異なる)
bRefresh	bool	設定後すぐに再描画するか(規定値 <i>true</i> )

表 106:Footy2SetMetrics関数の引数

この関数は以下の戻り値を返します。

値	意味
FOOTY2ERR_NOID	指定された Footy の ID 番号が見つからなかった
FOOTY2ERR_ARGUMENT	指定された nObject が上記で定義されている値ではない
FOOTY2ERR_NONE	エラーなしで関数成功

表 107:Footy2SetMetrics関数の引数

## 8.4.2. Footy2GetMetrics

この関数はFooty2GetMetrics関数などで設定した値を取得するための関数です。以下の引数を持ちます。

名前	型	意味
nID	int	数値を設定したいFootyのID番号
nObject	int	設定する箇所(上記定数が入る)
pValue	int*	数値を取得するための変数へのポインタ

表 108:Footy2GetMetrics関数の引数

この関数は以下の戻り値を返します。

値	意味
FOOTY2ERR_NOID	指定されたFootyのID番号が見つからなかった
FOOTY2ERR_ARGUMENT	指定されたnObjectが上記で定義されている値ではない、またはpValueがNULLである
FOOTY2ERR_NONE	エラーなしで関数成功

表 109:Footy2GetMetrics関数の戻り値

## 8.5. 見えているものの取得

この項では、見えている行数や桁数を取得するための関数を紹介します。これらの関数は以下の引数を持っています。

名前	型	意味
nID	int	数値を設定したいFootyのID番号
nView	int	取得したいビューID番号(詳細は16ページ参照)

表 110: GetVisible 系関数の引数

これらの関数は以下の戻り値を返します。

値	意味
FOOTY2ERR_NOID	指定された Footy の ID 番号が見つからなかった
FOOTY2ERR_ARGUMENT	指定されたビュー ID が正しくない(0~3 でない)
0 以上の値	それぞれの関数に対応した値

表 111: GetVisible 系の戻り値

### 8.5.1. Footy2GetVisibleColumns

この関数は、指定されたビューが表示可能な桁数を返します。Footy2Move 関数(14ページ参照)を呼び出した直後にこの関数で桁数を取得し、Footy2SetLepal 関数(68ページ参照)で折り返しを設定すれば常に画面端で折り返すということが可能になります。

### 8.5.2. Footy2GetVisibleLines

この関数は指定されたビューが表示可能な行数を返します。

## 8.6. そのほか設定系

### 8.6.1. Footy2SetLepal

この関数は、折り返しの設定を行うことができます。

名前	型	意味
nID	int	行アイコンを設定したい Footy の ID 番号
nColumns	int	折り返し桁数

nMode	int	折り返しモードフラグ(定数値 OR 結合)
bRefresh	bool	設定後すぐに再描画するか(規定値 <i>true</i> )

表 112:Footy2SetLepa関数

nMode にはどのように折り返しを行うかを設定できます。以下の値を OR 結合で指定してください。何も考えずにその位置で折り返す場合は LAPELFLAG\_NONE のみを指定します。

値	意味
LAPELFLAG_WORDBREAK	英文ワードラップ
LAPELFLAG_JPN_PERIOD	日本語の句読点を行頭に持ってこない
LAPELFLAG_JPN_QUOTATION	日本語のカギ括弧を行頭に持ってこない

表 113:折り返しモード

値	意味
FOOTY2ERR_NOID	指定された Footy の ID 番号が見つからなかった
FOOTY2ERR_ARGUMENT	折り返し桁数が不正
FOOTY2ERR_NONE	エラーなしで関数成功

表 114:Footy2SetLepa関数の戻り値

### 8.6.2. Footy2SetColor

この関数はエディタの各部位の色を設定する処理です。以下の引数を持ちます。

名前	型	意味
nID	int	色を設定したい Footy の ID 番号
nPosition	int	色を設定したい部位
nColor	COLORREF	設定したい色(RGB マクロで指定)
bRefresh	bool	設定後すぐに再描画するか(規定値 <i>true</i> )

表 115:Footy2SetColorの引数

この nPosition に設定可能な定数は以下の通りです。

値	意味
CP_TEXT	通常テキスト色
CP_BACKGROUND	背景色
CP_CRLF	改行マーク色
CP_HALFSPACE	半角スペース色
CP_NORMALSPACE	全角スペース色
CP_TAB	タブ文字色
CP_EOF	[EOF]マーク色( <i>End of File</i> )
CP_UNDERLINE	キャレット位置下線
CP_LINENUMBORDER	行番号との境界線
CP_LINENUMTEXT	行番号テキスト
CP_CARETLINE	行番号のキャレットがある行の強調背景
CP_RULERBACKGROUND	ルーラーの背景色
CP_RULERTEXT	ルーラーのテキスト
CP_RULERLINE	ルーラーの線
CP_CARETPOS	ルーラーのキャレット桁位置強調背景
CP_URLTEXT	URL のテキスト
CP_URLUNDERLINE	URL の下線
CP_MAILTEXT	メールアドレスのテキスト
CP_MAILUNDERLINE	メールアドレスの下線
CP_HIGHLIGHTTEXT	ハイライト表示のテキスト(未実装)
CP_HIGHLIGHTBACKGROUND	ハイライト表示の背景(未実装)

表 116:色設定可能な数値

bRefreshがtrueの場合はすぐに再描画処理が行われます。初期設定などでまとめて色の設定などを行う場合はこのbRefreshをfalseとして指定してください。その後、最終的にFooty2Refresh関数(18ページ)を呼び出すことで効率的に再描画を行えます。この関数は以下の値を返します。

値	意味
FOOTY2ERR_NOID	指定されたFootyのID番号が見つからなかった
FOOTY2ERR_ARGUMENT	nPositionが不正
FOOTY2ERR_NONE	エラーなしで関数成功

表 117:Footy2SetColorの戻り値

## 9. イベントの監視

### 9.1. イベントとは

イベントとは、ユーザーがコントロールに対して行った操作を通知する機能です。イベント駆動型のアプリケーションにおいては、イベント処理を行うことは必須といえます。Footy エディタコントロールもまた、ユーザーがキーを押したりマウスを動かすごとに処理を行っています。この章では、Footy エディタコントロールが発するイベントをどのように捕まえるか(ハンドリングするか)について解説します。

### 9.2. イベントを登録するには

全てイベントは、コールバック関数という形で捕まえることが可能です。コールバック関数とは、決められた一定の状態になったときに自動的に呼び出される関数の事です。C++で開発をした事があれば、ウィンドウのメッセージを受け取るときにウィンドウプロシージャコールバック関数を利用しますが、その感覚に近いです。関数を登録するときは、関数ポインタという値を使用します。この関数ポインタは、その関数がメモリ上のどの位置に存在するかを表す値です。この機能は各言語によって使用する方法が異なります。

まず最初に、コールバック関数として使用したい関数をクラスの内部関数でないようにします。これは、コールバック関数がクラスのインスタンスを利用できないからです。もしもクラスの内部関数として利用したい場合はstaticをつけてインスタンス無しでも呼び出すことが出来るようにしなければなりません。この場合はコールバック関数内部でstaticでないメンバ変数やメンバ関数にアクセスすることが出来なくなります(コンパイルエラーになります)。

```
class CTest{
public:
    void SetFunc(){
        Footy2SetFuncFocus(0,FocusCallback,NULL); //詳細は 74 ページ参照
    }

    static void FocusCallback(int nID,void *pData,int nViewID,bool bGotFocus){
        if (bGotFocus)
```

```

        m_nTestNumber++; //エラー!!!
    }

private:
    int m_nTestNumber;
};

```

そこで、static 関数でもメンバ変数や関数にアクセスするための機能が搭載されています。後述するデータ関数は、pData という void ポインタの引数を渡すことが可能です。このポインタは、関数ポインタが呼び出されるときにそのまま引き渡されます。

```

class CTest{
public:
    void SetFunc(){
        Footy2SetFuncFocus(0,FocusCallback,(void*)this);
    }

    static void FocusCallback(int nID,void *pData,int nViewID,bool bGotFocus){
        CTest *pInstance = (CTest*)pData;
        if (bGotFocus)
            pInstance->m_nTestNumber++;
    }

private:
    int m_nTestNumber;
};

```

## 9.3. イベントを登録するための関数

イベントを登録するための全ての関数は、以下の引数によって成り立っています。

名前	型	意味
nID	int	イベントを受け取りたい Footy エディタコントロールの ID
pFunc	各関数ポインタ	コールバック関数へのポインタ。型は以下に紹介する登録関数ごとに異なります。
pData	void*	コールバック関数へ渡したいデータを指定します。NULL にしておく事も可能です。

表 118: イベント登録関数の引数

### 9.3.1. Footy2SetFuncFocus

この関数は、Footy エディタコントロールがフォーカス(キー入力に対する優先)を得た、あるいは失った瞬間に発生するイベントを設定します。第二引数のイベントには Footy2FuncFocus 型を指定します。この関数がエディタ側から呼び出されたとき、アプリケーションは以下の引数をエディタ側から取得することが出来ます。

名前	型	意味
nID	int	フォーカスを得た、あるいは失った Footy のエディタ ID が通知されます。
pData	void*	ユーザーが関数を登録するときに設定したデータがそのまま通知されます。
nView	int	フォーカスを得た、あるいは失ったビュー ID(16ページ参照)
bFocused	bool	フォーカスを得たときは <i>true</i> 、失ったときは <i>false</i>

表 119: Footy2FuncFocus関数の引数

### 9.3.2. Footy2SetFuncMoveCaret

---

この関数によって設定されたイベントは、caret(文字入力する位置を示すバー)が移動したときに発生します。第二引数として Footy2FuncMoveCaret 型を指定します。このイベントが発生したときには以下の引数が通知されます。

名前	型	意味
nID	int	caretが移動された Footy のエディタ ID が通知されます。
pData	void*	ユーザーが関数を登録するときに設定したデータがそのまま通知されます。
nCaretLine	size_t	新たに設定された caret の行番号を示します(0 ベース)
nCaretColumn	size_t	新たに設定された caret の桁を示します(0 ベース)

表 120:Footy2FuncMoveCaretの引数

### 9.3.3. Footy2SetFuncTextModified

---

ユーザーの操作によってテキストが編集されたときに発生するイベントを設定します。この Footy2SetFuncTextModified 関数には第二引数として Footy2FuncTextModified 型を指定します。この関数型は以下の引数を持ちます。

名前	型	意味
nID	int	テキストが編集された Footy のエディタ ID が通知されます。
pData	void*	ユーザーが関数を登録するときに設定したデータがそのまま通知されます。
nCause	int	テキストが編集された要因を通知されます。

表 121:Footy2FuncTextModifiedの引数

この中で、nCause はこのイベントが呼び出された原因を示す定数で、Footy2.h に宣言されている以下の定数の中から適切な物が通知されます。

定数名	意味
MODIFIED_CAUSE_CHAR	文字が入力された(IME オフ)
MODIFIED_CAUSE_IME	文字が入力された(IME オン)
MODIFIED_CAUSE_DELETE	Delete キーが押下された
MODIFIED_CAUSE_BACKSPACE	BackSpace が押下された
MODIFIED_CAUSE_ENTER	Enter キーが押下された
MODIFIED_CAUSE_UNDO	元に戻す処理が実行された
MODIFIED_CAUSE_REDO	やり直し処理が実行された
MODIFIED_CAUSE_CUT	切り取り処理が実行された
MODIFIED_CAUSE_PASTE	貼り付け処理が実行された
MODIFIED_CAUSE_INDENT	インデント処理が行われた(Ver 2.010 以降)
MODIFIED_CAUSE_UNINDENT	逆インデント処理が行われた(Ver 2.010 以降)
MODIFIED_CAUSE_TAB	タブキーが押された(Ver 2.010 以降)
MODIFIED_CAUSE_SETSELTEXT	Footy2SetSelText が呼び出された(Ver 2.019 以降)
MODIFIED_CAUSE_SETLINETEXT	Footy2SetSelLine が呼び出された(Ver 2.019 以降)

表 122:nCause の内容

#### 9.3.4. Footy2SetFuncInsertModeChanged

この関数は、挿入/上書きモードが変化したときに発生します(INsert キーが押されたときに発生します)。この関数の第二引数は Footy2FuncInsertModeChanged 型になります。このイベントではエディタライブラリから以下の値が渡されます。

名前	型	意味
nID	int	イベントが発生した Footy のエディタ ID が通知されます。
pData	void*	ユーザーが関数を登録するときに設定したデータがそのまま通知さ

		れます。
bInsertMode	bool	挿入モードのときに <i>true</i> 、上書きモードのとき <i>false</i>

表 123:Footy2FuncInsertModeChangedの引数