Oracle Tuxedo のグローバリゼーション機能：アジア太平洋地域でのマルチバイト・サポート

Oracle ホワイト・ペーパー
2008 年6 月更新

ORACLE

Oracle Tuxedo のグローバリゼーション機能：アジア太平洋地域でのマルチバイト・サポート

はじめに 3
Oracle Tuxedo のグローバリゼーション強化 3
ソフトウェア・グローバリゼーションの展望 4
　国際化とローカリゼーション 4
　コードセットとエンコーディング 5
　マルチバイト・エンコーディング変換：過去と現在 5
　典型的な変換シナリオ 6
Oracle Tuxedo のマルチバイト変換機能 6
　エンコーディング変換の管理 6
　クライアント側の処理 7
　サーバー側の処理 9
　カスタマイゼーション 11
　エイリアス名のエンコーディング 11
フィールド操作言語バッファのマルチバイト・データ 12
結論 12
付録 1：マルチバイト・データに関連する API 13
付録 2：ソフトウェアの例 14
　マルチバイト・データ変換例 14
　　クライアント側のアプリケーション 14
　　サーバー側のアプリケーション 17
　FLD_MBSTRING の使用 19
　　クライアント側のアプリケーション 20
　　サーバー側のアプリケーション 22
　カスタム変換機能 24

Oracle Corporation 発行「Oracle Tuxedo Globalization Features: Multibyte Support for the Asia Pacific Reg」の翻訳版です。

Oracle Tuxedo のグローバリゼーション機能：アジア太平洋地域でのマルチバイト・サポート

国際化とローカリゼーションは、Oracle Tuxedo システムの重要な要素です。アジア太平洋地域では、ソフトウェアがマルチバイト文字のコードセット処理を完全にサポートしており、開発者は言語の制約なしでソリューションを開発できます。

はじめに

国際化とローカリゼーションは、Oracle Tuxedo システムの重要な要素です。アジア太平洋地域では、ソフトウェアがマルチバイト文字のコードセット処理を完全サポートしており、開発者は言語の制約なしでソリューションを開発できます。Oracle Tuxedo は、国際化に対応するために必要なソフトウェアのカスタマイズを排除することで、企業が自社のアプリケーションを簡単に拡張し、社員やパートナーへ多言語で提供できるようにします。

このホワイト・ペーパーでは、いくつかの例とともに Oracle Tuxedo のグローバリゼーション機能を紹介します。

Oracle Tuxedo のグローバリゼーション強化

Oracle Tuxedo は、次に挙げる国際化の機能強化をおこなっています。

- ユーザー・データにおけるマルチバイト文字のバッファ・タイプをサポート
- API、または中国語、日本語、韓国語のコードセット・エンコーディング間の自動変換を使用したオンデマンドのプログラム変換機能
- プログラムおよび手動でコードセット・エンコーディング情報を"取得"および"設定"し、自動変換のオンとオフを切り替える機能
- カスタム変換機能による変換ライブラリの簡易交換をサポート

これらの特定の強化は、MBSTRING と呼ばれるバッファ・タイプや、FLD_MBSTRING と呼ばれるフィールド・タイプ、マルチバイト文字転送と変換 API など、いくつかのシステム機能を使用します。

Oracle Tuxedo であれば、プログラマは環境変数の TPMBENC と TPMBACONV を使用して、手動でエンコーディング変換を実施するか、新しい API 関数を使用してプログラムで実施するかのどちらかを選択できます。オンとオフをプログラムで自動変換する機能を使用することで、必要な場合にだけ変換を実行するようアプリケーションを制御でき、変換にかかわるパフォーマンスをコントロールできるようになります。

Oracle Corporation 発行「Oracle Tuxedo Globalization Features: Multibyte Support for the Asia Pacific Reg」の翻訳版です。

Oracle Tuxedo システムが自動コードセット・エンコーディング変換に設定されていれば、異なるコンピュータ・プラットフォーム上で実行されているプロセス間に MBSTRING バッファ（または FML32 バッファの FLD_MBSTRING フィールド）が転送されると、基盤システムはあるコードセット・エンコーディングから別のコードセット・エンコーディングへと変換されます。具体的には、受信側が MBSTRING バッファを送信側のコードセット・エンコーディングの表現から受信側の表現へと自動変換します。環境変数の TPMBENC と TPMBACONV を使用して、自動コードセット・エンコーディングが手動で設定されていない場合、送信側または受信側のアプリケーションは、場合に応じて、変換 API によりコードセット・エンコーディング変換を要求できます（詳細は付録 1 を参照）。

GNU iconv 変換ライブラリを使用すれば、UNIX や Windows プラットフォームで一般的なコードセット変換機能を利用できます。Oracle Tuxedo バッファ・タイプを使用すれば、テストやパフォーマンス・チューニングなどのカスタム機能を簡単に変換ライブラリへ置き換えることができます。

ソフトウェア・グローバリゼーションの展望

オペレーティング・システム、ライブラリ、開発ツールなど、ほとんどのソフトウェア製品はまったく異なる言語や文化、プレゼンテーション要求をもつ国際的な環境で使用できるよう設計および開発されています。たとえば、東京に本社を置き、ニューヨークとソウルに支店をもつ大企業では、英語、日本語、韓国語を組み合わせたソフトウェア環境が必要です。さらに、こうした国際的に分散するコンピューティング環境では、地域ごとに異なる時間、数値、日付、貨幣価値、コードセット・エンコーディング・スキーマなどをサポートする必要もあります。これらのすべての要件は、トランザクションが世界各国に広がる中で、自発的に（アプリケーションを再起動することなく）サポートされなければなりません。こうした要件を満たすソフトウェアのことを、グローバル対応ソフトウェアと呼びます。

国際化とローカリゼーション

国際化は、異なる言語や文化の地域間でソフトウェアを移植可能にします。ローカリゼーションは、国際化されたプログラムの特定のバージョンを地理的または政治的な地域内で使用できるようにします。

ソフトウェアのグローバル化は、国際化とローカリゼーションの両要件に取り組むことで達成できます。国際化は、異なる言語や文化の地域間でソフトウェアを移植可能にします。国際化されたソフトウェアを作成するために、開発者は言語や文化に依存するプログラムの一部を切り離しています。たとえば、エラー・メッセージは、ロケールで使用する言語に訳しやすくするために切り離されています。ロケールとは、同一の言語や慣習をもつ地理的または政治的地域のことです。国際化されたプログラムは、システムの初期化の際に、ロケールに依存する個所を取り出すよう設計されているか、適用するよう作られています。

ローカリゼーションは、ロケール依存個所に対してロケール固有のバージョンまたはパッケージを作成するためのプロセスです。ローカリゼーションには、ユーザー・インタフェースのラベル、エラー・メッセージ、オンライン・ヘルプなどのテキストを翻訳する作業が含まれます。また、時間、貨幣価値、日付、数字など、文化固有の形式にデータ項目を修正する作業も含まれます。オラクルは、ローカリゼーションが必要なお客様にパッケージを提供しています。

コードセットとエンコーディング

*キャラクタ・セット*とは、指定された言語を表す要素のセットです。英語のアルファベットはキャラクタ・セットになります。文字間には黙示的な順序関係がある場合もありますが、特定の値が割り当てられているわけではありません。たとえば、英語のアルファベットは*a*、*b*、*c*で始まり、普通に読み上げていくと*x*、*y*、*z*で終わります。確かに、このような黙示的な順序は存在しますが、それを示す数値的な関係は文字間にはありません。コードセットは、このような数値的関係を提供し、コンピュータ・プログラムがキャラクタ・セットを操作するメカニズムを与えます。

さらに、*コードセット*はコード化キャラクタ・セットとも呼ばれています。これは、コンピュータ・ベースでマッピングされたキャラクタ・セットのことで、負の整数ではない独自のものです。コードセットの独自のバイナリ値をマッピングしたものは、コードセットのエンコーディングと呼びます。米国では、ほとんどのコンピュータ・キーボードのキャラクタ・セットは、ASCIIとUnicodeの 2 つのコードセットを採用しています。ASCIIもエンコーディングです。特定のコードセットが複数のエンコーディングをもつこともできます。たとえば、日本のコンピュータ・ベンダーは、日本語のコードセットである漢字に対して、少なくとも 3 つのエンコーディングをサポートしています。EUC-JP、Shift-JIS（SJIS）、ISO-2022-JPです。ほとんどのUNIXベンダーはEUC-JPをサポートしており、SJISをサポートしているベンダーもいます。Windows、OS/2、MacintoshはSJISをサポートします。韓国ではKSC5601 エンコーディングが幅広く使用されており、中国ではGBKが使用されています。JavaはネイティブのUnicodeのほか、多くの外部エンコーディングをサポートします。

Oracle Tuxedo は、マルチバイト・コードセット処理機能をサポートします。標準的な英語は 8 ビット（単一バイト）コードセット・エンコーディング・スキーマで対応できますが、中国語、日本語、韓国語は、マルチバイト・コードセット・エンコーディング・スキーマが必要です。

マルチバイト・エンコーディング変換：過去と現在

標準的な英語を含む欧州言語のアルファベット文字は、8 ビット（単一バイト）コードセット・エンコーディング・スキーマで対応できます。しかし、大規模な記号（または表意文字）のセットを基盤とする中国語、日本語、韓国語では、マルチバイト・コードセット・エンコーディング・スキーマが必要です。Oracle Tuxedo は、これらのアジア太平洋地域のキャラクタ・セットをマルチバイト・コードセット処理でサポートします。

Oracle Tuxedo が登場する前、アプリケーション開発者は、グローバリゼーション機能を得るためにカスタム変換ソリューションを作成する必要がありました。しかし、カスタム変換は非常に限られたユースケースでしか扱えません。たとえば、あるカスタム・ソリューションは SJIS と EUC-JP 間の変換をおこない、別のソリューションは SJIS と ISO-2022-JP 間の変換をおこないます。Oracle Tuxedo であれば、カスタム変換の種類をそれぞれ開発する必要はありません。

重要なのは、Oracle Tuxedo のコードセット変換機能は、コードセットのエンコーディング変換をおこなうために設計されたということです（たとえば、Unicode コードセットの UTF-8 と UTF-16BE エンコーディング間）。コードセット間（たとえば ASCII と Unicode 間）の変換や言語間の翻訳をおこなうのではなく、同一言語の異なるエンコーディングを変換します。

典型的な変換シナリオ

もっとも一般的なマルチバイト変換シナリオは、異なるプラットフォーム上で異なる漢字エンコーディング・スキーマが実行されているというものです（たとえば、クライアント/サーバーシステム）。クライアントとサーバーが異なるエンコーディング・スキーマを使用するプラットフォーム上でホストされている場合、プラットフォーム間でエンコーディング変換を実施する必要があります。図 1 の例は、日本の分散コンピューティング環境でのシナリオを示したものです。この例では、クライアント側は SJIS をサポートする Windows マシンで、Oracle Tuxedo サーバー・マシンは EUC-JP をサポートする UNIX ベースとなっています。

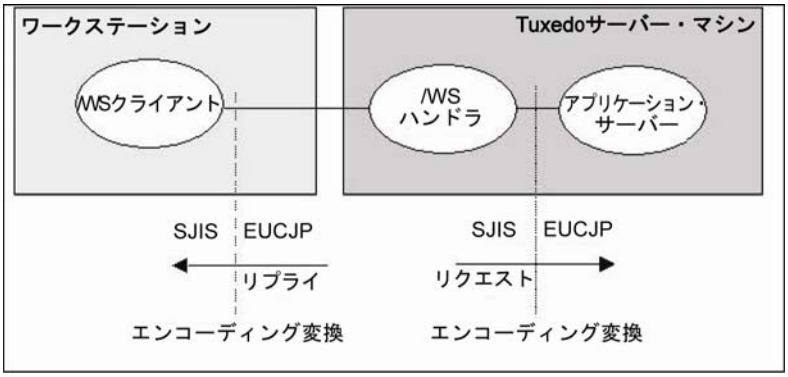

図1：分散コンピューティング環境の典型的なシナリオ

Oracle Tuxedo のマルチバイト変換機能

前の例では、マルチバイト文字のアプリケーション・データが MBSTRING と呼ばれるバッファ・タイプを使用して、Windows クライアントと Oracle Tuxedo サーバーのプロセス間で転送されました。このバッファ・タイプに関連する API 関数のセットは、GNU iconv ライブラリを使用してコードセット・エンコーディング名を決定し、マルチバイト・データ変換を実行します。

Oracle Tuxedo の開発者は、バッファ・タイプのスイッチ構造（tm_typesw）に新しいエントリを追加して、MBSTRING バッファを実装します。これにより、Oracle Tuxedo は各バッファ・タイプに対してどのルーチンを呼び出すかを決定できます。MBSTRING バッファについては、システムが内部関数の_mbsconv()を呼び出して、自動コードセット・マルチバイト・データ変換を実行します。そのあと、この内部関数は GNU ライブラリのルーチンを使用してユーザー・データを変換します。

変換は、本質的にパフォーマンス面を犠牲にします。パフォーマンスに悪影響を与えないようエンコーディングを変換するために、Oracle Tuxedo は変換を手動およびプログラムの両面で制御できるようにします。

エンコーディング変換の管理

エンコーディング変換を制御するには、2 つの方法があります。1 つは、環境変数の TPMBENC と TPMBACONV を使用して手動でエンコーディング変換を実施する方法で、もう 1 つは API 関数を使用してプログラムで実施する方法です。自動変換で環境変数が設定されている場合、受信側の Oracle Tuxedo システムは、バッファ内のデータをあるエンコーディングから別のエンコーディングへ変換します。もしくは、プログラム・インタフェースの tuxsetmbaconv()を使用して、アプリケーションを再起動することなく自動エンコーディングをオンとオフに切り替えるようにします。これにより、必要な場合にのみ変換をおこなうよう制限できます。そうしないと、変換は各ホップ時に実施され、パフォーマンスを劣化させてしまいます。

図 2 は、図 1 で示したものと同じ例を使用していますが、Oracle Tuxedo がどのようにマルチバイト・データを扱うかをより詳細に図示しています。環境変数の TMPBENC と TPMBACONV は各マシンに設定されており、エンコーディングや自動エンコーディング変換の状態（オンまたはオフ）を識別します。この例では日本ロケールが設定されており、SJIS エンコーディングをサポートする Windows クライアントと、EUC-JP エンコーディングをサポートする UNIX サーバーが描かれています。バッファ・タイプのヘッダーは、バッファを MBSTING タイプと識別し、エンコーディングとデータ長の情報を提供します。バッファ自体は、ヘッダー内にエンコーディングで識別されたユーザー・データを保持しています。クライアントは、バッファ格納に対して SJIS エンコーディングのデータをリクエストし、サーバーは EUC-JP エンコーディングのデータを返します。

アプリケーションを設計する際、2 つの点を考慮します。1 つ目は、変換は本質的にパフォーマンス面を犠牲にするという点です。この例のように自動変換を使用すると、メッセージに対して変換は 2 回おこなわれることになります。1 回目はサーバーでリクエストを受信したとき、2 回目はクライアントがリプライを受信したときです。2 つ目は、バッファ内のユーザー・データのサイズが変換に応じて変わるという点です。この例のクライアント側では、バッファのサイズは変換後も同じか、または小さくなります。サーバー側では、バッファのサイズは同じか、または大きくなります。

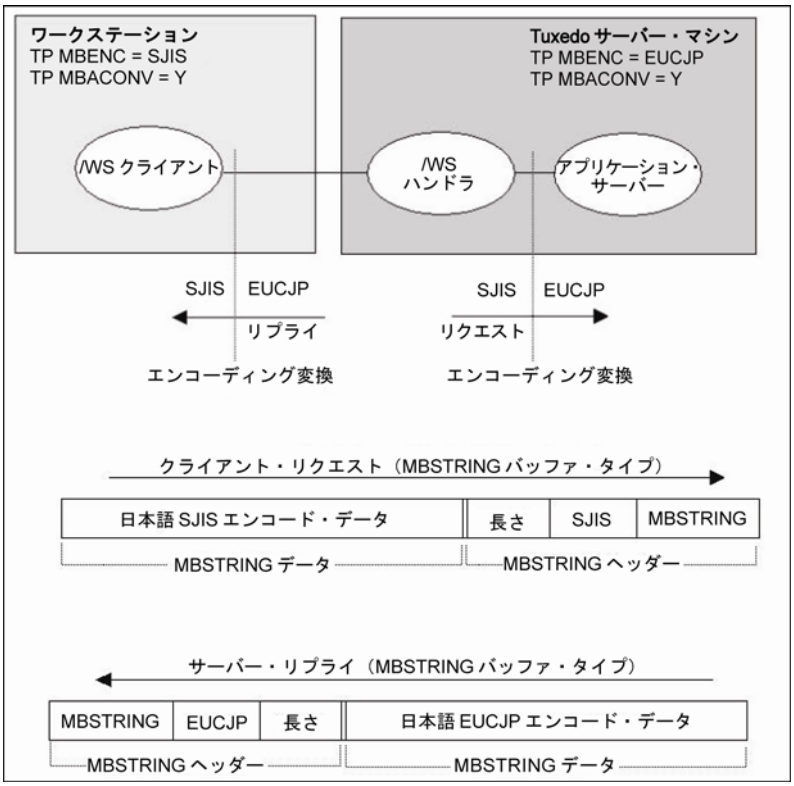

図 2 : MBSTRING バッファ・タイプによるデータ変換

クライアント側の処理

クライアント処理が実行されると、実行されるマシンでサポートするコードセット・エンコーディング名が取得または設定されます。たとえば、環境変数の TMPBENCを使用してエンコーディング名のセットを取得する場合、クライアン

トはtuxgetmbenc()を呼び出して、TPMBENC=*encodingName*の形式で記述された文字列の環境リストを検索します。文字列が存在すれば、クライアントがOracle Tuxedoのtpalloc()を呼び出して新しいMBSTRINGバッファを割り当てるときに、ユーザー・データとともにエンコーディング名が受け渡されます。そのあと、エンコーディング名はキャッシュされ、バッファ・タイプのスイッチ関数が一度起動して処理されたら、再度呼び出す必要がないようにします。環境変数のTMPBENCが定義されていない場合、または処理時に再設定したい場合、アプリケーションはAPI関数を使用してこれを実施できます。

クライアントが一度 tpalloc を呼び出せば、Oracle Tuxedo は図 3 に示したとおり、バッファ割当てとデータ変換を提供するようになります。基盤の Oracle Tuxedo システムは新しい MBSTRING バッファに対してメモリを割り当てて、tuxgetmbenc() 関数の内部バージョンを使用します。これにより、設定されていれば、TMPBENC 環境変数に定義されたエンコーディング名が取得されます。Oracle Tuxedo は、MBSTRING バッファにエンコーディング名を追加し、クライアントへ割り当てられたバッファを返します。

そのあと、たとえば tpsend()や tpcall()を使用して MBSTRING バッファをクライアントが送信すると、Oracle Tuxedo は再度介入し、次の項の"サーバー側の処理"で解説するように、受信側で変換を実施します。

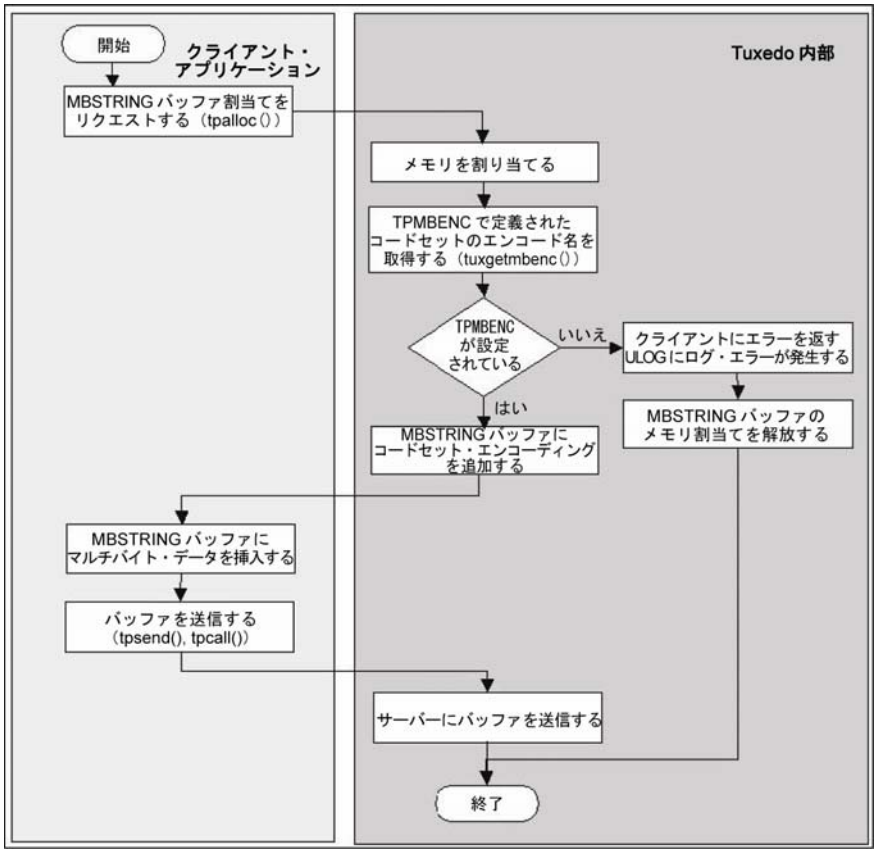

図3：バッファの割当てとデータ変換を提供する Oracle Tuxedo でのクライアント処理

サーバー側の処理

次の図 4 に示すダイアグラムは、クライアントがサーバーへ MBSTRING バッファ・タイプを含むリクエストを送信した際の基盤の Oracle Tuxedo 処理を示したものです。図 4 には、クライアントがリプライを受信した際と同じ手順も示してあります。サービスにメッセージをわたす前に、Oracle Tuxedo は MBSTRING バッファを受信します。Oracle Tuxedo は環境変数の TPMBACONV を見て、自動変換が設定されているかを確認します。設定されていなければ、エンコーディング変換を実行せずに MBSTRING バッファをサーバーへ受け渡します。自動変換が設定されていれば、Oracle Tuxedo は TPMBENC 内に定義されているエンコーディング名を取得します。Oracle Tuxedo は、エンコーディング値が設定されているかを確認するため、いくつかのエラー・チェックをおこないます。設定されていない場合は、変換がおこなえないからです。値が設定されていない場合、エラーをログに残してサーバーへ処理をわたします。

TPMBENC 環境変数が設定されている場合、Oracle Tuxedo のタイプ・スイッチ要素は自動的にクライアントのエンコーディング名とサーバーのエンコーディング名を比較し、異なる場合は、Oracle Tuxedo が受信したメッセージのエンコーディングを、GNU iconv ベースのライブラリ・ルーチンまたはユーザー作成のカスタム変換ルーチンを使用して、サーバー・マシンでサポートされるエンコーディングへ自動変換します（カスタム変換ルーチンの作成については、"カスタマイゼーション"の項を参照）。Oracle Tuxedo は、変換データをサービスに提供し、制御をわたします。

サーバー側およびクライアント側のマルチバイト変換アプリケーションの例は、このドキュメントの最後にある付録 2 を参照してください。

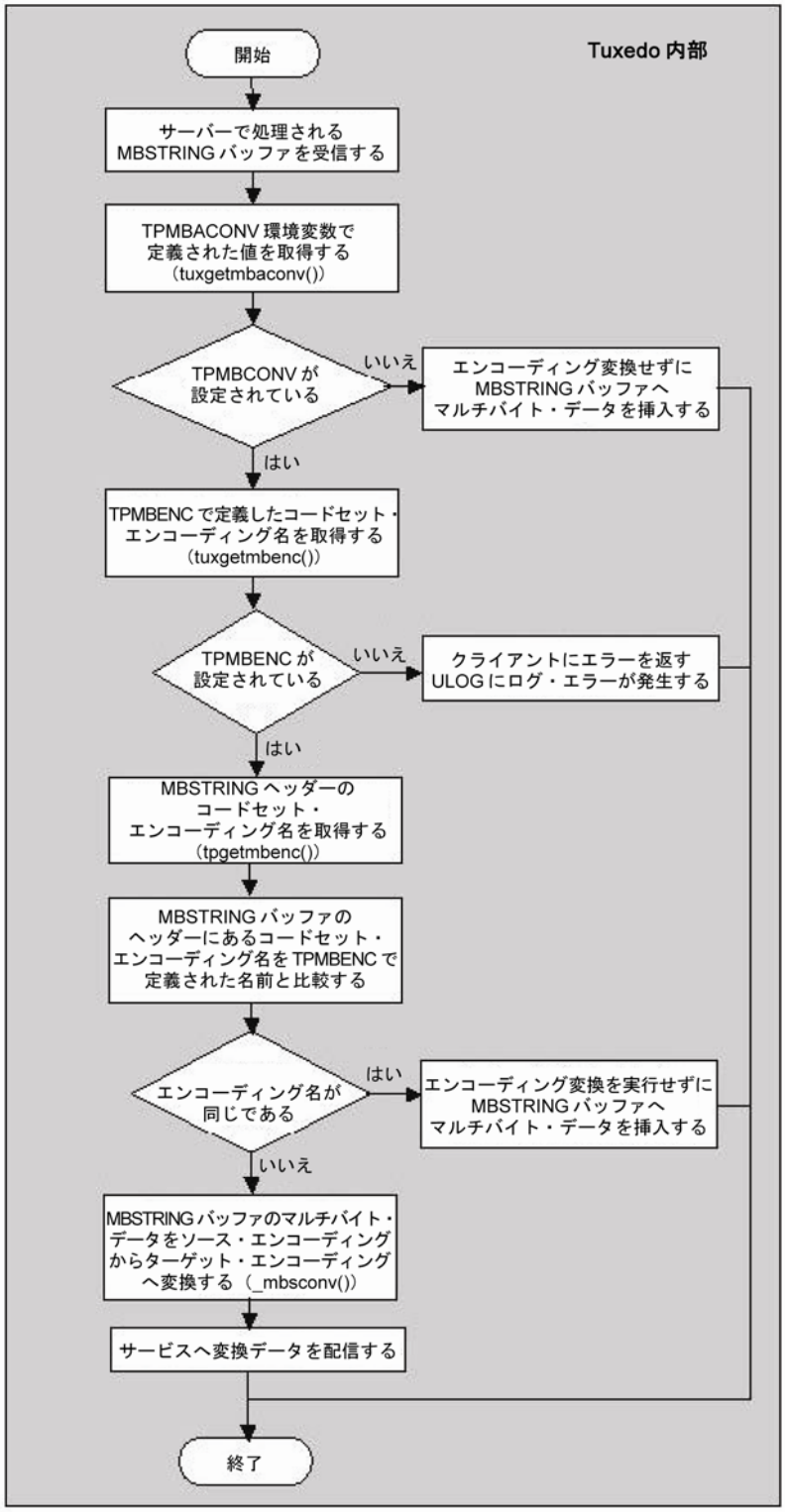

図4：クライアントが MBSTRING バッファ・タイプを含むリクエストをサーバーへ送信した際に発生する、サーバー処理の基盤となる Oracle Tuxedo 処理

カスタマイゼーション

MBSTRING のカスタム自動変換ルーチンは、デフォルトの変換関数名を MMSTRING で定義されたカスタム関数名に置き換えることで、簡単にインストールできます。

開発者は、たとえば変換関数を使用したりデバッグしたりする際によりよいパフォーマンスを得たい場合、または提供ライブラリで扱っていないカスタム文字がある場合、カスタム変換関数を作成しようとします。MBSTRING のカスタム自動変換ルーチンは、デフォルトの変換関数名を MMSTRING で定義されたカスタム関数名に置き換えることで、簡単にインストールできます。MBSTRING は、tmtypesw.c ファイルに定義されています。これは、Oracle Tuxedo バッファ・タイプがプロセス・バッファ・タイプ・スイッチ（tm_typesw）に追加する場所です。バッファは、アプリケーション・レベルで tpconvmb()関数を使用して、自動変換機能とは別に変換できます。

次の tmtypesw.c ファイルの一部は、MBSTRING のカスタム定義を示したものです。最後の行は、デフォルトの変換関数名である _mbsconv がカスタム関数名の CUSTmbconv に置き換えられたことを示しています。これにより、Oracle Tuxedo が MBSTRING タイプ・データで自動エンコーディング変換を実施するとき、デフォルト関数の代わりにカスタム変換ルーチンが呼び出されます。

```
"MBSTRING",     /* type */
"*",            /* subtype */
0,              /* dfltsize *
/_mbsinit,      /* initbuf */
NULL,           /* reinitbuf */
NULL,           /* uninitbuf */
NULL,           /* presend */
NULL,           /* postsend */
NULL,           /* postrecv */
NULL,           /* encdec *
/NULL,          /* route */
NULL,           /* filter */
NULL,           /* format */
NULL,           /* presend2 */
CUSTmbconv      /* customized multi-byte codeset conversion */
```

CUSTmbconv コードは変換で通常使用する関数により構成されていますが、UNIX オペレーティング・システムで通常使用する iconv コールへと削減されています。カスタマイゼーション関数の例は、付録 2 を参照してください。

エイリアス名のエンコーディング

GNU iconv 仕様は、charset.alias ファイルの仕様を制限します。ユーザーは、このファイルを使用して既存のエンコーディング名のエイリアスを定義します。これは、SJIS、SHIFT_JIS、SHIFT-JIS、MS_KANJI、CSSHIFTJIS など、独自のエンコーディングを指定する際に使用する一般的な名前が記載された GNU のビルトイン・リストに追加される機能です。ただし、この機能は提供されてはいますが、パフォーマンスの問題があるため、推奨されません。代わりに、エンコーディングで使用する名前を GNU iconv 仕様から選択します。

フィールド操作言語バッファのマルチバイト・データ

Oracle Tuxedo において、フィールド操作言語 32（FML32）バッファは、コードセット識別されたマルチバイト・データとして FLD_MBSTRING フィールド・タイプを受け入れます。Fmbpack32()関数と Fmbunpack32()関数は、処理する際に必要な情報が入ったフィールドを提供します。このパック形式データは、FML32 バッファと FML32 バッファのレシーバとともに送信されます。TPMBACONV 環境変数が設定されていれば、自動的に FML32 バッファ・タイプ・スイッチ変換関数（_fmbconv32）を実行します。この関数は、FML32 バッファの FLD_MBSTRING フィールドをチェックし、フィールド情報内のエンコーディング名がローカルの TPMBENC 環境変数と異なる場合は、変換を実施します。_mbconv 関数と同様に、ユーザーは tmtypesw を再定義してカスタマイズできます。アプリケーションは、FML32 API 関数と FLD_MBSTRING フィールド・タイプを使用して FML32 バッファの変換されたパック形式データにアクセスします。データは、Fmbunpack32() 関数を使用してアンパックします。

FML32 バッファで MBSTRING と FLD_MBSTRING を使用する例は、付録 2 を参照してください。

結論

異なる言語や慣習をもつ地域間で移植できる国際化されたソフトウェア・アプリケーションは、グローバル企業の事業成功において重要です。

異なる言語や慣習をもつ地域間で移植できる国際化されたソフトウェア・アプリケーションは、グローバル企業の事業成功において重要です。Oracle Tuxedo が登場する前、アプリケーション開発者はグローバリゼーション機能を得るために、カスタム変換ソリューションを作成する必要がありました。しかし、カスタム変換は非常に限られたユースケースでしか扱えません。Oracle Tuxedo が提供するグローバリゼーション機能は、次のとおりです。

- ユーザー・データにおけるマルチバイト文字のバッファ・タイプのサポート
- API、または中国語、日本語、韓国語のコードセット・エンコーディング間の自動変換を使用したオンデマンドのプログラム変換機能
- プログラムまたは手動でコードセット・エンコーディング情報を"取得"および"設定"し、自動変換のオンとオフを切り替える機能
- カスタム変換機能による変換ライブラリの簡易交換のサポート

こうしたグローバリゼーション機能を使用することで、アプリケーション管理者は複数言語のアプリケーションをさらに容易に管理できるようになります。

付録 1：マルチバイト・データに関連する API

この付録の表は、MBSTRING と FLD_MBSTRING コマンドに関連する関数を示します。

tpconvmb()	入力バッファとともに受け渡される文字のエンコーディングを、指定されたターゲット・エンコーディングへ変換する。
tpgetmbenc()	クライアントまたはサーバーが MBSTRING バッファからコードセット・エンコーディング名を処理もしくは再設定する。tpsetmbenc()は、エンコーディング名が設定されているかどうかを示す値を返す。アプリケーションで必要なエンコーディング名が MBSTRING バッファの一部として指定されたものと異なる場合は、tpsetmbenc()を使用する。
tuxsetmbaconv()	クライアントまたはサーバーが TPMBACONV 環境変数を取得または設定する。get 操作で TPMBACONV を示す値が返された場合、バッファ・タイプ・スイッチ関数によってコードセット・データ変換は自動実行される。tuxsetmbaconv()関数を実行すると、TPMBACONV 関数を設定または設定を解除する。
tuxsetmbenc()	クライアントまたはサーバーがTPMBENC環境変数を取得または設定する。アプリケーションはこの関数を使用して、TPMBENCの設定または再設定をおこなう。get関数は、TPMBENC=*value*の形式で記載された文字列の環境リストを検索する。リストがある場合、現在の環境に値へのポインタを返す。

表 1：関数に関連する MBSTRING

Fmbpack32()	FML32 API 関数の入力として使用するバイト・ストリームを作成する。入力には、コードセット・エンコーディング名、コードセット・マルチバイト・データ、入力データ長を使用する。上記の入力を含む出力データのポインタを、FML32 が使用できる形式で返す。
Fmbunpack32()	FLD_MBSTRING の FML32 API 関数アクションの出力を取得し、アプリケーションが使用できる情報へ変換する。バイト数と FML32 関数の結果であるパック形式のバイト・ストリームを入力する。コードセット・エンコーディング名、マルチバイト・ユーザー・データ、および返されたデータ長を返す。
tpconvfmb32()	アプリケーション開発者が、バッファ・タイプ・スイッチ関数とは別にマルチバイト・データ変換を実施できるようにする。入力 FML32 バッファ、出力 FML32 バッファ、およびターゲット・コードセット・エンコーディング名を取得する。入力 FML32 バッファを検証し、ターゲット・コードセット・エンコーディング名の引数と異なるコードセット・エンコーディング名を含む FLD_MBSTRING フィールド・タイプを更新する。

表 2：関数に関連する FLD_MBSTRING

付録２：ソフトウェアの例

マルチバイト・データ変換例

この例では、MBSTRING に関連する API 関数の使用方法を簡単な変換シナリオで示します。"カスタマイゼーション"の項で解説したマルチバイト・データの変換例は、この例のアプリケーションからの観点を示したものです。

クライアント側のアプリケーション

```
/* #ident "@(#)apps:simpapp/simpclmb.c 1.1" */

#include <stdio.h>
#include "Uunix.h"
#include "atmi.h" /* TUXEDO  Header File */
#if defined(__STDC__) || defined(__cplusplus)
main(int argc, char *argv[])
#else
main(argc, argv)
int argc;
char *argv[];
#endif
{
/*

    ************************************************************
    この例では、入力文字列を TOUPPERMB サービスへ送信します。TOUPPERMB
    サービスは文字を大文字に変換し、クライアントへ結果を返します。このク
    ライアント側の処理のエンコーディング名は UTF-16LE として定義され、
    送信するバッファは UTF-8 エンコーディングとして再定義されて、サーバー
    側のエンコーディングは UTF-16BE になります。両方で自動変換がオンに
    なっている場合、サーバー処理は MBSTRING を TOPUPERMB サービスへ受け
    渡す前に UTF-8 から UTF-16BE へ変換します。サービスが終了して MBSTRING
    が返されると、このクライアント処理は結果バッファを tpcall rcvbuf
    引数としてアプリケーションに渡す前に、UTF-16BE から（この処理で定義
    されたエンコーディングである）UTF-16LE へ変換します。最後に、rcvbuf
    は再度 UTF-8 エンコーディングに変換され、プリントアウトされます。
    UTF-16LE の手順は必要ありませんが、API の使用方法を示すために追加し
    ました（たとえば、UTF-8<=>UTF-16BE は、自動変換で対応できます）。
    ************************************************************
*/
    char *sendbuf, *rcvbuf;
    long sendlen, alloclen, rcvlen;
    int ret,iolen;
    if(argc != 2) {
        (void) fprintf(stderr, "Usage: simpclmb string¥n");
        exit(1);
    }
    /* Attach to System/T as a Client Process */
    if (tpinit((TPINIT *) NULL) == -1) {
        (void) fprintf(stderr, "Tpinit failed¥n");
        exit(1);
    }

/*
```

```
    ************************************************************
    自動マルチバイト変換を"OFF"にすることが望ましい場合は、次の 6 行をコ
    メント・アウトするか削除します。
    tuxsetmbaconv は、このクライアント処理のみを制御します。サーバー処
    理は、独自の環境変数を設定するか、独自の tuxsetmbaconv()関数を実
    行する必要があります。
    注：この 2 行を使用する以外に、TPMBACONV 環境変数を設定する方法があ
    ります
    （export TPMBACONV="YES"など）。
    ************************************************************
*/
    ret = tuxsetmbaconv(MBAUTOCONVERSION_ON,0);
    if(ret == -1) {
        (void) fprintf(stderr, "tuxsetmbaconv failed¥n");
        exit(1);
    }
    (void) fprintf(stderr, "tuxsetmbaconv ON done.¥n");
    /*
    ************************************************************
    注：次の 6 行を使用する以外に、TPMBENC 環境変数を設定する方法があり
    ます（export TPMBENC="UTF-16LE"など）。
    ************************************************************
    */
    ret = tuxsetmbenc("UTF-16LE",0);
    if(ret == -1) {
        (void) fprintf(stderr, "tuxsetmbenc failed¥n");
        exit(1);
    }
    (void) fprintf(stderr, "tuxsetmbenc UTF-16LE done.¥n");
    sendlen = strlen(argv[1]);
    /*

    ************************************************************
     注: この例では、ASCII 入力文字列を使用しています。顧客独自のエンコー
     ディングは、文字定義内の組込み NULL の関係で、OS の文字列関数で動作
     しない可能性があります。通常、メモリまたは wcstring 関数は、コード
     セット・エンコーディングが組込み NULL をもっているかどうかを意識せ
     ずに使用できます。よって、この例では NULL ターミネータとして sendlen
     に 1 を追加していません。正確な bytecnt のみが使用されています。文
     字列関数を使用して、NULL ターミネータを送信データ長に追加するかは、
     開発者次第です。
    ************************************************************
    */
    (void) fprintf(stderr,"Input:%s, Length: %d¥n", argv[1],
    sendlen);
    /* Allocate MBSTRING buffers for the request and the
    reply */ alloclen = sendlen * 4; /*max size buf ensures
    min # iconv iterations*/ if((sendbuf = (char *)
    tpalloc("MBSTRING", NULL, alloclen)) == NULL) {
   (void) fprintf(stderr,"Error allocating send buffer: %s¥n",
    tpstrerror(tperrno));
         tpterm();
         exit(1);
    }
    if((rcvbuf = (char *) tpalloc("MBSTRING", NULL,alloclen))
    == NULL) {
         (void) fprintf(stderr,"Error allocating receive
     buffer¥n");
    tpfree(sendbuf);
    tpterm();

     exit(1);
```

```
     }
/*
     ********************************************************
     新規の tpalloc の sendbuf で使用するデフォルトのエンコーディングは、
     UTF-16LE です（上記で tuxsetmbenc()を実行したため）。しかし、このクライアントに入力されるデータは、UTF-8 エンコーディングです（UTF-8 の argv[1]など）。よって、sendbuf エンコーディングを UTF-8 に再設定する必要があります。
     ********************************************************
*/
     ret = tpsetmbenc(sendbuf,"UTF-8",0);
     if(ret == -1) {
     (void) fprintf(stderr, "tpsetmbenc UTF-8 failed¥n");
     (void) fprintf(stderr, "Tperrno = %d¥n", tperrno);
     exit(1);
     }
     (void) fprintf(stderr, "tpsetmbenc UTF-8 done.¥n");
     (void) memcpy(sendbuf, argv[1], (size_t)sendlen);
     /* Request the service TOUPPERMB, waiting for a reply */
     ret = tpcall("TOUPPERMB", (char *)sendbuf, sendlen, (char
     **)&rcvbuf, &rcvlen, (long)0);

     if(ret == -1) {
          (void) fprintf(stderr, "Can't send request to
          service TOUPPERMB¥n");
          (void) fprintf(stderr, "Tperrno = %d¥n", tperrno);
          tpfree(sendbuf);
          tpfree(rcvbuf);
          tpterm();
          exit(1);
     }
     (void) fprintf(stdout, "Returned rcvbuf Length %d¥n",
     rcvlen);

/*
     ********************************************************
     rcvbuf は、この処理が TOUPPERMB サービスからリプライ・バッファを受信したとき、（最初に tuxsetmbaconv()を使用したことから）自動的に UTF-16BE から UTF-16LE へ変換されました。しかし、このアプリケーションは UTF-8 エンコーディングでプリントアウトすることを要求しているので、UTF-16E から UTF-8 へ変換を強制する必要があります。
     ********************************************************
*/
     iolen = (int)rcvlen;
     ret = tpconvmb(&rcvbuf, &iolen, "UTF-8", (long)0);
     if(ret == -1) {
          (void) fprintf(stderr, "Can't execute tpconvmb.¥n");
          (void) fprintf(stderr, "Tperrno = %d¥n", tperrno);
          tpfree(sendbuf);
          tpfree(rcvbuf);
          tpterm();
          exit(1);
     }

/*
     ********************************************************
     注：tpconvmb は、出力に対して rcvbuf を再利用し、UTF-8 へ変換された iolen バイトを返します。これを文字列として正しく出力するには、rcvbuf に NULL ターミネータを追加するか、別の char*と strncpy を使用する必要があります。
     ********************************************************
```

```
*/
    *(rcvbuf + iolen) = '¥0';
    /*output received buf from TOUPPERMB service converted to
    UTF-8 */ (void) fprintf(stdout, "simpclmb output string
    is: %s, Length %d¥n", rcvbuf, iolen);

    /* Free Buffers & Detach from System/T */
    tpfree(sendbuf);
    tpfree(rcvbuf);
    tpterm();
    return(0);
}
```

サーバー側のアプリケーション

```
/* #ident "@(#)apps:simpapp/simpservmb.c 1.0" */

#include <stdio.h>
#include <ctype.h>
#include <atmi.h> /* TUXEDO Header File */
#include <userlog.h> /* TUXEDO Header File */

/* tpsvrinit is executed when a server is booted, before it
begins processing requests. It is not necessary to have this
function. Also available is tpsvrdone (not used in this
example), which is called at server shutdown time.
*/

#if defined(__STDC__) || defined(__cplusplus)
tpsvrinit(int argc, char *argv[])
#else
tpsvrinit(argc, argv)
int argc;
char **argv;
#endif
{
    int ret,iolen;

  /* userlog writes to the central TUXEDO message log */
    userlog("Welcome to the simpservmb server");

  /* Some compilers warn if argc and argv aren't used. */
    argc = argc;
    argv = argv;
/*

    ********************************************************
    自動マルチバイト変換を"OFF"にすることが望ましい場合は、次の 6 行をコ
    メント・アウトするか削除します。
    tuxsetmbaconv は、このクライアント処理のみを制御します。サーバー処
    理は、独自の環境変数を設定するか、独自の tuxsetmbaconv()関数を実
    行する必要があります。
    注：この 2 行を使用する以外に、TPMBACONV 環境変数を設定する方法があ
    ります（export TPMBACONV="YES"など）。

    ********************************************************
*/
    ret = tuxsetmbaconv(MBAUTOCONVERSION_ON,0);
    if(ret == -1) {
        (void) fprintf(stderr, "tuxsetmbaconv failed¥n");
        exit(1);
```

```
    }
     userlog("tuxsetmbaconv ON done");
/*

    ********************************************************
    注：次の 6 行を使用する以外に、TPMBENC 環境変数を設定する方法があり
    ます（export TPMBENC="UTF-16BE"など）。
    ********************************************************
*/
    ret = tuxsetmbenc("UTF-16BE",0);
    if(ret == -1) {
         (void) fprintf(stderr, "tuxsetmbenc failed¥n");
         exit(1);
    }
         userlog("tuxsetmbenc UTF-16LE done");
         return(0);
}

/* This function performs the actual service requested by the
client. Its argument is a structure containing among other
things a pointer to the data buffer, and the length of the data
buffer.
*/
#ifdef __cplusplus
extern "C"
#endif
void
#if defined(__STDC__) || defined(__cplusplus)
TOUPPERMB(TPSVCINFO *rqst)
#else
TOUPPERMB(rqst)
TPSVCINFO *rqst;
#endif
{
    int i,ret;
    char myenc[80];
    char *en=&myenc[0];

    userlog("TOUPPERMB Input Length: %d", rqst->len);
    /*
    ********************************************************
    前述の tpsvrinit を見ると、自動変換がオンになっており、すでにサーバー
    処理ではバッファをこのサービスにわたす前に、定義されたエンコーディン
    グへ変換しています。rqst データは UTF-16BE エンコーディング内にあり、
    rqst 長は UTF-8 エンコーディング時の倍の長さになっています。
    ********************************************************
    */
    ret = tpgetmbenc(rqst->data,en,0);
    if(ret == -1) {
         (void) fprintf(stderr, "tpgetmbenc failed.¥n");
         (void) fprintf(stderr, "Tperrno = %d¥n", tperrno);
         tpreturn(TPFAIL, 0, rqst->data, 0L, 0);
    }
    if(strcmp(en,"UTF-16BE") !=0) {
         (void) fprintf(stderr, "tpgetmbenc not==UTF-
     16BE.Got: %s¥n",en);
         (void) fprintf(stderr, "Tperrno = %d¥n", tperrno);
         tpreturn(TPFAIL, 0, rqst->data, 0L, 0);
    }
     userlog("tpgetmbenc check==UTF-16LE done");
```

```
/*
    ********************************************************
    自動マルチバイト変換を"OFF"にすることが望ましい場合、ただし、このア
    プリケーションにおいてオンデマンドで変換を実施する必要がある場合は、
    コード内の次の 12 行をサンプルとして使用してください。自動変換が実施
    されず、tpconvmb が実行されない場合、rqst データ・バイトはクライア
    ント処理と同じエンコーディング名（UTF-8 など）で定義されます。
    ********************************************************

    if(tuxgetmbaconv(0) == MBAUTOCONVERSION_OFF) {
           ret = tpconvmb(&rqst->data, &iolen, "UTF-16BE",
    (long)0);
           if(ret == -1) {
              (void) fprintf(stderr, "Can't execute
    tpconvmb.¥n");
              (void) fprintf(stderr, "Tperrno = %d¥n",
     tperrno);
    tpreturn(TPFAIL, 0, rqst->data, 0L, 0);
    }
    userlog("tpconvmb new mbstring length: %d",iolen);
    }
*/
    for(i = 0; i < rqst->len; i++) {
         if(rqst->data[i]) {
         userlog("TOUPPERMB index: %d, char: %c", i, rqst-
         >data[i]);
         rqst->data[i] = toupper(rqst->data[i]);
         } else {
    /*

    ********************************************************
    注：独自の判断で取得したデータに対して文字列/プリント関数を使用する
    のは危険です。もとのデータが UTF-8 で送信されても、受信した変換デー
    タは UTF-16B になっており、組込み NULL をもっています。LIBC 文字列/
    プリント関数は正確に動作しないか、停止してしまいます。
    ********************************************************
    */
    userlog("TOUPPERMB skip index: %d",i);
   }
}
/* Return the transformed buffer to the requestor. */
tpreturn(TPSUCCESS, 0, rqst->data, rqst->len, 0);
}
```

FLD_MBSTRING の使用

最後の例では、FLD_MBSTRIN の使用を示します。

クライアント側のアプリケーション

```
#include <stdio.h>
#include <stdlib.h>
#include <atmi.h>
#include <userlog.h>
#include <fml.h>
#include <fml32.h>
#include "fmltbl32.h"
/*
  ******************************************************
  fmltbl32.h を生成するのに使用した fmltbl32 ヘッダー・ファイルは、単
  一のフィールド定義です。
  # name    number  type    flags   comments
  FLD4   112    mbstring -      -
  ******************************************************
*/
#define BUFLEN 1024
#ifdef _TMPROTOTYPES
main(int argc, char *argv[])
#else
main(argc, argv)
int    argc;
char   *argv[];
#endif
{
    FBFR32*fmlptr;
    long rlen;
    int ret;
    char *fldmbio;
    FLDLEN32 packedlen;
/*
  ************************************************************
  この例では、同一のフィールドの 2 つの出現を UTF-8 パック形式データに設定
  し、FML32SRV サービスへ送信します。サービスはフィールドとともに UTF-16BE
  形式でバッファを返し、バッファはローカルで UTF-8 へ変換されます。
  ************************************************************
*/
    /* Attach to System/T as a Client Process  */
    if (tpinit((TPINIT *)NULL) == -1) {
    (void) fprintf(stderr,"tpinit failed: %s¥n",
tpstrerror(tperrno));
    exit(1);
    }
    ret = tuxsetmbaconv(MBAUTOCONVERSION_ON,0);
    if(ret == -1) {
(void) fprintf(stderr, "tuxsetmbaconv failed¥n");
    exit(1);
    }
(void) fprintf(stderr, "tuxsetmbaconv ON done.¥n");
/*
  ************************************************************
  自動変換がオンになっているので、処理環境のためにエンコーディングを設定
  する必要があります。これは、tpcall へのリプライがアプリケーション・コー
  ドで利用可能になる前に UTF-8 へ変換されるからです。
  ************************************************************
*/
    ret = tuxsetmbenc("UTF-8",0);
    if(ret == -1) {
     (void) fprintf(stderr, "tuxsetmbenc failed¥n");
     exit(1);
    }
    (void) fprintf(stderr, "tuxsetmbenc UTF-8 done.¥n");
```

```
    /* allocation for fml32 buffer */
    if ( (fmlptr = (FBFR32 *) tpalloc("FML32", NULL, BUFLEN) ==
NULL ) {
     (void) fprintf(stderr,"tpalloc failed: %s¥n",
     tpstrerror(tperrno));
     tpterm();
     exit(1);
    }

     /* create and pack datastream input for FLD_MBSTRING
     fields */ packedlen = 256;/*excessive space, actual bytes
     used is very little*/ fldmbio =
     (char*)malloc((size_t)packedlen);
     if ( Fmbpack32("UTF-8", "hello", 5, fldmbio,
     &packedlen,0) < 0 ) {
          (void) fprintf(stderr,"Fmbpack32 on hello failed:
          %d¥n", Ferror32);
          exit(1);
    }
    /*set 1st occurence of FLD_MBSTRING field FLD4*/
    if ( Fchg32(fmlptr, FLD4, (FLDOCC32)-1, fldmbio, packedlen)
< 0 ) {
          (void) fprintf(stderr,"Fchg on FLD4,0 failed: %d¥n",
Ferror32);
          exit(1);
    }
    userlog("Fchg on FLD4,0 passed. packedlen: %d", packedlen);

    packedlen = 256;
    if ( Fmbpack32("UTF-8", "world", 5, fldmbio, &packedlen,0)
< 0 ) {
          (void) fprintf(stderr,"Fmbpack32 on bobf failed:
%d¥n", Ferror32);
          exit(1);
    }
    /*set 2nd occurence of mbstring field FLD4*/
    if ( Fchg32(fmlptr, FLD4, (FLDOCC32)-1, fldmbio, packedlen)
< 0 ) {
          (void) fprintf(stderr,"Fchg on FLD4,1 failed: %d¥n",
Ferror32);
          exit(1);
    }
    userlog("Fchg on FLD4,1 passed. packedlen: %d", packedlen);
/*
  ************************************************************
  注：すべてのフィールドは、同一のエンコーディングで定義されているので、
  各エンコーディングを個別に設定するには、FML32 バッファで tpsetmbenc
  (UTF-8)を設定してから、フラグ引数に FBUFENC の Fmbpack32()を、エン
  コーディング引数に NULL を使用します。これにより、使用するバッファの総
  サイズを削減できます。
  ************************************************************
*/
    puts("The FML32 buffer sent : -");
    Fprint32(fmlptr);
    userlog("Fchg32 : successful");
    /*send the FML32 buffer to the FMLSRV32 service*/
     if(tpcall("FMLSRV32",(char*)fmlptr,0,(char**)&fmlptr,&rlen,
     TPNOTIME) == - 1 ) {
          (void) fprintf(stderr,"tpcall failed: %s¥n",
          tpstrerror(tperrno));
          exit(1);
```

```
    }

    puts("The FML32 buffer got : -");
    Fprint32(fmlptr);

    tpfree((char *)fmlptr);
    tpterm();
    exit(0);
}
```

サーバー側のアプリケーション

```
#include <stdio.h>
#include <ctype.h>
#include <atmi.h>   /* TUXEDO Header File */
#include <userlog.h>   /* TUXEDO Header File */
#include <fml.h>
#include <fml32.h>
#include "fmltbl32.h"

/* tpsvrinit is executed when a server is booted, before it
begins processing requests. It is not necessary to have this
function. Also available is tpsvrdone (not used in this
example), which is called at server shutdown time.
*/

#if defined(__STDC__) || defined(__cplusplus)
tpsvrinit(int argc, char *argv[])
#else
tpsvrinit(argc, argv)
int argc;
char **argv;
#endif
{
    int ret=0;
    /* Some compilers warn if argc and argv aren't used. */
    argc = argc;
    argv = argv;

    /* userlog writes to the central TUXEDO message log */
    userlog("Welcome to the simple server");

    ret = tuxsetmbaconv(MBAUTOCONVERSION_ON,0);
    if(ret == -1) {
        (void) fprintf(stderr, "tuxsetmbaconv failed¥n");
        exit(1);
    }
    userlog("tuxsetmbaconv ON done");

    ret = tuxsetmbenc("UTF-16BE",0);
    if(ret == -1) {
     (void) fprintf(stderr, "tuxsetmbenc failed¥n");
     exit(1);
    }
    userlog("tuxsetmbenc UTF-16BE done");

    return(0);
}

/* This function performs the actual service requested by the
client. Its argument is a structure containing among other
```

```
things a pointer to the data buffer, and the length of the data
buffer.
*/

#ifdef __cplusplus
extern "C"
#endif
void
#if defined(__STDC__) || defined(__cplusplus)
FMLSRV32(TPSVCINFO *rqst)
#else
FMLSRV32(rqst)
TPSVCINFO *rqst;
#endif
{
    char buf[1024];
    char odata[1024];
    char pckdata[1024];
    char encname[256];
    char *bufptr = (char *)(rqst->data);
    int i=0,occ=0;
    FLDLEN32 odatalen=0,packedlen=0,buflen=0;
    userlog("Welcome to the fml32srv server");
/*
  ************************************************************
  FML32 バッファで自動変換がオンになっているため、FMLSRV32 サービスで受
  信するバッファは、ローカルのエンコーディングに変換されます（UTF-16BE
  など）。次のコードは、fml32 バッファからフィールドを取得し、ユーザー・
  データを抽出して、データを操作し（大文字に変換するなど）、再パックして、
  フィールドを変更してからクライアントへ fml32 バッファを返します。
  ************************************************************
*/

     for (occ = 0;occ < 2; occ++) {
        buflen = 1024;
           /*get FLD_MBSTRING field from FML32 buffer*/
        if ( Fget32((FBFR32 *)bufptr, FLD4, occ, buf, &buflen)
           == -1 ) { userlog ("Fget32 FLD4,%d failed: %d", occ,
           Ferror32); tpreturn(TPFAIL, 0, rqst->data, 0L, 0);
        }
        userlog("FMLSRV32 Fget32 FLD4,%d passed buflen: %d",
           occ, buflen);
        odatalen = 1024;
           /*unpack the field into user data and encoding
           info*/
        if ( Fmbunpack32(buf, 20, encname,odata,&odatalen,0) ==
           -1 ) { userlog ("Fmbunpack32 FLD4,%d failed", occ);
            tpreturn(TPFAIL, 0, rqst->data, 0L, 0);
        }
        userlog("FMLSRV32 FLD4,%d encname: %s", occ, encname);
             /*change relevant bytes to uppercase*/
      for(i = 0; i < odatalen; i++) {
           if(odata[i]) {
            userlog("FMLSRV32 FLD4,%d index: %d, char:
            %c",occ,i,odata[i]); odata[i] = toupper(odata[i]);
           } else {
            userlog("FMLSRV32 FLD4,%d skip index: %d", occ, )
            i};
           }
      }
      packedlen = 1024;
```

```
          /*pack encoding name and user data into field data*/
        if ( Fmbpack32("UTF-
        16BE",odata,odatalen,pckdata,&packedlen,0) < 0 ) {
          userlog("Fmbpack32 on FLD4,%d failed: %d", occ,
          Ferror32);
          tpreturn(TPFAIL, 0, rqst->data, 0L, 0);
        }
          /*set the FLD_MBSTRING with new packed data*/
        if (Fchg32((FBFR32
        *)bufptr,FLD4,(FLDOCC32)occ,pckdata,packedlen) < 0) {
          userlog("Fchg32 on FLD4,%d failed: %d", occ,
          Ferror32);
          tpreturn(TPFAIL, 0, rqst->data, 0L, 0);
        }
        userlog("Fchg32 on FLD4,%d passed. packedlen: %d ",
        occ, packedlen);
    }
    userlog("Successfully done with the fml32srv server");

    /* Return the transformed buffer to the requestor. */
    tpreturn(TPSUCCESS, 0, rqst->data, 0L, 0);
}
```

カスタム変換機能

```
/*
 * CUSTmbconv
 *
 * This function will convert characters from a source
encoding, defined in the TCM, to a targe codest.
 *
 * INPUT
 *     *iptr       - pointer to an input buffer
 *     ilen        - Length of input buffer
 *     *target_enc   - Characters in iptr will get converted
                     to the code set defined by this name.
 *     *flags     - valid values are TMUSEIPTR or TMUSEOPTR.
Where results put.
 *
 * OUTPUT
 *     *optr     - pointer to an output buffer. If null, use
iptr to output.
 *     olen      - Length of output buffer. If optr is null, use
ilen.
 *     *flags    - valid values are TMUSEIPTR or TMUSEOPTR.
Where results put.
 *
 * RETURNS
 *      -1             - Failure (check errno for reason)
 *     positive #      - Success. Return val is num of bytes
used in result.
 *     negative #      - Not enough space. Value is -1*(guess
of bytes needed)
 */
/*ARGSUSED*/
long
#ifdef _TMPROTOTYPES
_TMDLLENTRY
CUSTmbconv(char _TM_FAR *iptr, long ilen, char _TM_FAR
*target_enc, char _TM_FAR
     *optr, long olen, long _TM_FAR *flags)
#else

CUSTmbconv(iptr, ilen, target_enc, optr, olen, flags)
char *iptr;
```

```
long ilen;
char *target_enc;
char *optr
;long olen;
long *flags;
#endif
{
    iconv_t cd;
    char   *tptr;
    char   *to = (char *)NULL;
    char  *fptr;
    size_t  ileft, oleft, ret,used=0;
    char  encname[56];
    char  *src_enc = &encname[0];

    if ( (target_enc == NULL) || (*target_enc == '¥0') ) {
         /* missing target encoding argument */;
         return(-1);/*WILL NEED TO SET TPERRNO for return -1*/
    }

    if(tpgetmbenc(iptr,src_enc,0) < 0) {
         /* missing source encoding name */;
         return(-1);
    }

    /* convert characters from source encoding to target
    encoding format */
    cd = iconv_open((const char *)target_enc, (const char
    *)src_enc);
    if (cd == (iconv_t)-1) {
         /* iconv_open failure */
         return (-1);
    }

    if(optr == NULL) {
         /* If no output buf given and if conversion fails due
         to insufficient*/
         /* buf size then the input buf would be unusable when
         sent back for  */
         /* a retry attempt. So use tmp buffer for output until
         conversion is */
         /* clean and copy it back to the input buffer upon
         successful conv*/
         if(olen == 0) {
            /*probably will throw E2BIG error with correct size
            to use*/
            olen = ilen;
         }
         /*if olen!=0 then it should be the max size of the iptr
buffer*/
         if ((to = (char *)malloc((size_t)olen)) == NULL) {
            return (-1);
         }

    } else {
         to = optr;
    }
    tptr = to;
    fptr = iptr;
    ileft = ilen;
    oleft = olen;

    (void) iconv(cd,NULL,NULL,NULL,NULL); /* go to the initial
```

```
state */
    for ( ;; ) {
        ret = iconv(cd, &fptr, &ileft, &tptr, &oleft);
        if (ret != (size_t)-1) {
           /* iconv succeeded. NOTE: Some characters may not
have needed */
          /* conversion and are the same as input value
          representation */
          used = used + (olen - oleft);
          olen = oleft;
          if(ileft != 0) {
               /* iconv not done, execute again*/
               continue;
          }
          if(optr == NULL) {
               /* no output buffer given,copy tptr buf back to
               input buffer*/
               (void) memcpy(iptr,to,used);/*DANGER:iptr len
               must be >= used*/
               free(to);
               *flags |= TMUSEIPTR;
           } else {
               /* characters from iconv exec in output buffer,
optr */
               *flags |= TMUSEOPTR;
           }
           (void) iconv(cd,NULL,NULL,NULL,NULL);/* reset to
           initial state */
           (void) iconv_close(cd);
           return (used);/*return #bytes used in output
           buffer*/
        } else {
           /* iconv failed */
           (void) iconv(cd,NULL,NULL,NULL,NULL);/* reset to
           initial state */
           (void) iconv_close(cd);
           if(optr == NULL) {
               free(to);
           }
           if (errno == E2BIG) {
               olen = (ilen + (4 * ileft)) * -1;
               *flags |= TMUSEIPTR;
               return (olen);/*return guesstimate of size iptr
               should be*/
           } else if (errno == EINVAL) {
               /* Incomplete char/shift sequence */
           } else if (errno == EILSEQ) {
               /* NOTE: We do not handle code set state
               dependent sequences */
           } else if (errno == EBADF) {
                /* Actually, this should happen above
               during iconv_open */
           } else {
               /* Undefined error */
           }
           return(-1);
        }
    }
}
```

Oracle Tuxedo のグローバリゼーション機能：
アジア太平洋地域でのマルチバイト・サポート
2008 年 6 月更新

Oracle Corporation
World Headquarters
500 Oracle Parkway
Redwood Shores, CA 94065
U.S.A.

海外からのお問い合わせ窓口：
電話：+1.650.506.7000
ファクシミリ：+1.650.506.7200
www oracle.com