Oracle9*i* Lite

（C および C++）オブジェクト・カーネル API リファレンス

リリース 5.0

2001 年 7 月
部品番号：J04365-01

ORACLE®

Oracle9*i* Lite（C および C++）オブジェクト・カーネル API リファレンス , リリース 5.0

部品番号：J04365-01

原本名：Oracle9*i* Lite C and C++ Object Kernel API Reference, Release5.0

原本部品番号：A90124-01

Printed in Japan.

目次

はじめに

この API リファレンスは、フットプリントの小さいデバイスでのデータベースに最適なアプリケーションを作成する開発者を対象としています。

このマニュアルでは、Oracle9*i* Lite の C および C++ の Application Programming Interface（API）である、オブジェクト・カーネル API（OKAPI）について説明します。Oracle9*i* Lite の Java 言語インタフェースの詳細は、『Oracle9*i* Lite Java 開発者ガイド』を参照してください。

説明する内容は、次のとおりです。

第 1 章「OKAPI のプログラミング」	OKAPI を使用するデータベース・アプリケーションのプログラミングの基礎を説明します。
第 2 章「OKAPI 関数」	OKAPI 関数およびデータ型を詳しく説明します。
第 3 章「言語間の相互運用性」	リレーショナルおよびオブジェクト指向のデータ記憶域モデル間の相互運用性について説明します。
第 4 章「Stateless Object Database API」	Oracle Lite Stateless Object Database API について説明します。

1

OKAPI のプログラミング

この章では、Oracle Lite オブジェクト・カーネル API（OKAPI）を使用するデータベース・アプリケーションのプログラミングの基礎を説明します。共通操作を説明するためのサンプル・コードも示します。説明する内容は、次のとおりです。

- OKAPI の使用方法
- 環境の作成
- データベースの作成
- データベースへの接続
- グループとクラスの使用方法
- グループの作成
- 属性の作成
- クラスの作成
- オブジェクトの作成
- オブジェクトの削除
- 属性の設定
- イテレータによる問合せの実行
- コミット・トランザクションおよびロールバック・トランザクション
- 索引の作成

OKAPI の使用方法

OKAPI のモジュールは C 言語のコール・インタフェースで、アプリケーションはこれを使用して Oracle Lite のオブジェクト・カーネルの機能にアクセスします。

OKAPI を使用するには、OKAPI に必要なすべての定義、宣言、および include 文が含まれている **okapi.h** ファイルをユーザーのプログラムに組み込む必要があります。

環境の作成

OKAPI 関数をコールする前に、環境を作成する必要があります。環境とは、OKAPI 関数がステート情報の格納に使用する、ヒープ内のメモリーです。OKAPI 関数 `okInit` は、次のように環境を初期化します。

```
okEnv    env;    // Variable to hold pointer to the
                 // environment
okError  err;    // Variable to store error code

err    = okInit( NULL, &env );
if ( OK_IS_ERROR(err) )
   ...           // Handle error

   ...           // Code using OKAPI

okFinal( env );  // Free environment
```

`okInit` の 1 番目の引数は、パラメータ構造へのポインタです。これが `NULL` の場合は、OKAPI はデフォルト設定で環境を初期化します。

データベースの作成

データベースを作成するには、`okCreateDatabase` 関数を使用します。デフォルト設定を使用したデータベース作成コードの例を、次に示します。

```
char     *dbsName = "My Database";
okSize   dbsNameLen = StrLen( dbsName );
err      = okCreateDatabase( env, dbsNameLen, dbsName, NULL );
```

指定された名前のデータベースがすでに存在している場合、`okCreateDatabase` は失敗します。最後の引数は、データベース作成のオプションが指定されている構造体へのポインタです。これが `NULL` に設定されている場合、OKAPI はデフォルト設定でデータベースを作成します。

データベースへの接続

ユーザーは、一度に複数のデータベースをオープンできます。既存のデータベースをオープンするには、okConnect を使用します。

```
okRef   dbRef;
err     = okConnect( env, dbsNameLen, dbsName, NULL, &dbRef );
```

4 番目の引数（サンプルでは NULL）は、オプションを設定するための構造体へのポインタです。最後の引数 dbRef は、okRef 変数へのポインタです。これには、データベース参照が返されます。この参照は、オープンされているデータベースに対するランタイム・ハンドルで、プログラムを再起動した後は無効になります。

グループとクラスの使用方法

SQL の用語では、表は同じフォーマットの行の集合です。行のフォーマットは、表の列によって定義されます。OKAPI では、グループを使用して類似のデータ・オブジェクトを収集できます。異なるフォーマットのオブジェクトも同じグループに含められるので、グループは表よりも柔軟です。これによって、構造は違っても論理的に関連するオブジェクトをグループ化できます。

オブジェクト（または行）には属性（または列）があります。オブジェクトのクラスは、オブジェクトのフォーマットを定義します。したがって、オブジェクトの集合を作成するには、次の手順が必要になります。

1. グループを作成します。
2. 属性を作成することでクラスのフォーマットを定義します。
3. クラスを作成してグループに連結します。

クラスを作成して連結すると、そのクラスからオブジェクトを作成できます。

グループの作成

okCreateGroup 関数を使用して、次の例のようにグループを作成します。

```
okRef    groupRef;
char     *groupName = "MyGroup";
okSize   groupNameLen = StrLen(groupName);
err      = okCreateGroup( env, dbRef,
           groupNameLen, groupName,
           0, &groupRef );
```

返されたハンドル groupRef は、グループ・オブジェクトへのポインタです。

名前のわかっている既存のグループを使用する場合は、okFindGroup をコールしてそのグループ名に対するハンドルを取得します。

属性の作成

クラスを作成するには、最初に属性構造体の配列を構成します。各属性構造体は、クラス内の個々の属性（または列）を表します。属性構造体 okAttrDesc は、**okapi.h** の中で宣言されています。そのフォーマットは、次のとおりです。

```
typedef struct okAttrDesc {
   okU4B     NameLen;          /* 属性の名前の長さ */
   okName    NamePtr;          /* 属性の名前のポインタ */
   okU4B     DefinedInLen;     /* クラス名で定義されたオプションの長さ */
   okName    DefinedInPtr;     /* クラス名で定義されたオプションのポインタ */
   okU4B     DomainLen;        /* ドメインクラス名の長さ */
   okName    DomainPtr;        /* ドメインクラス名のポインタ */
   okU4B     Precision;        /* 属性値のオプションの精度 */
   okU1B     SdtScale;         /* オプションの SDT スケール */
   okU1B     SdtTypeCode;      /* オプションの SDT タイプ・コード */
   okU1B     SdtSubtypeCode;   /* オプションの SDT サブタイプ・コード */
   okU1B     DefCollate;       /* オプションのデフォルト照合順番 */
   ok4B      Dimension;        /* データ・ユニット数 */
   okU4B     DefaultValSize;   /* デフォルト値のオプションのサイズ */
   okByte    *DefaultValPtr;   /* デフォルト値のオプションのポインタ */
   okU4B     ExtensionCount;   /* 拡張オブジェクトのオプション数 */
   okHandle  *ExtensionHandles;  /* 拡張ハンドルのオプション配列 */
   okU4B     Options;          /* オプションの属性オプション */
} okAttrDesc_s;
```

多くの場合、属性記述構造体で設定する必要があるフィールドはわずかです。設定する必要があるフィールドのリストを次の表に示します。

フィールド	説明
NameLen	属性の名前の長さ。
NamePtr	属性の名前。名前はクラス内で一意になる必要があり、NULL で終了する必要はありません。
DomainLen	ドメインの名前。
DomainPtr	属性のタイプ名。
Dimension	データ単位の数。1 に設定すると、要素が 1 つのみのスカラー属性になります。0 に設定すると、DomainPtr で指定したタイプの可変長配列になります。1 より大きい場合は、固定サイズの配列になります。

その他のフィールドは、すべて 0 に設定できます。

たとえば、要素が 1 個の長整数の属性を定義するコードは、次のようになります。

```
okAttrDesc_s attrDesc;
MemSet( &attrDesc, sizeof(okAttrDesc_s), 0 ); // すべてのフィールドを消去
attrDesc.NamePtr   = "My Age";          // 属性の名前
attrDesc.NameLen   = StrLen( attrDesc.NamePtr ); // 名前の長さ
attrDesc.DomainPtr   = "okU4B";         // タイプ名
attrDesc.DomainLen   = StrLen(attrDesc.DomainPtr);
attrDesc.Dimension   = 1;               // 配列なし
```

可変長文字列属性の宣言は、次のようになります。

```
okAttrDesc_s attrDesc;
MemSet( &attrDesc, sizeof(okAttrDesc_s), 0 ); // すべてのフィールドを消去
attrDesc.NamePtr   = "My String";       // 属性の名前
attrDesc.NameLen   = StrLen( attrDesc.NamePtr ); // 名前の長さ
attrDesc.DomainPtr   = "okChar";        // タイプ名
attrDesc.DomainLen   = StrLen(attrDesc.DomainPtr);
attrDesc.Dimension   = 0;               // 可変長配列
```

次の表に、サポートされている属性タイプまたはコア・オブジェクトのリストを示します。

データ型	説明
okU4B	符号なし整数（4 バイト）
ok4B	符号付き整数（4 バイト）
okU2B	符号なし整数（2 バイト）
ok2B	符号付き整数（2 バイト）
okU1B	符号なし文字（1 バイト）
ok1B	符号付き文字（1 バイト）
okChar	文字（言語依存サイズ、英語の場合は 1 バイト）
okFloat	単精度浮動小数点数（4 バイト）
okDouble	倍精度浮動小数点数（6 バイト）
okRef	他のオブジェクトへの参照（4 バイト）

ユーザーが作成したクラスの名前も使用できますが、それらのクラスのインスタンスとして生成されたオブジェクトのデータは、そのオブジェクトに埋め込まれます。他のオブジェクトへのポインタには、属性タイプとして okRef を使用します。okRef によって、他のオブジェクト、グループおよびクラスへのポインタが使用できます。

クラスの作成

属性記述を作成すると、okCreateClass を使用してクラスを作成できます。クラス作成の例を、次に示します。

```
okRef   classRef;
err     = okCreateClass( env, dbRef,
                         classNameLen, className,
                         0, 0,
                         attrCount, attrDescs,
                         0, 0,
                         0,
                         &classRef );
```

変数 dbRef は、オープンしているデータベースに対する参照（okConnect から返された値）です。attrCount には属性の数が含まれます。attrDescs は、クラスの属性を記述する okAttrDesc 構造体の配列です。

最後の引数は okRef へのポインタで、関数はこれを使用してクラスへの参照を返します。クラスの名前でクラスへの参照を検索するには、okFindClass をコールします。

オブジェクトの作成

クラスおよびグループを作成すると、そのクラスに基づいたオブジェクトを作成できます。グループへのオブジェクトの追加は、表に行を挿入するのに似ています。

オブジェクトを作成するには、グループおよびクラス・オブジェクトへの参照が必要です。それらを取得するために、okFindClass および okFindGroup をコールします。

```
err   = okCreateObj( env, groupRef, classRef, NULL, 0, &objRef );
```

最後の引数には、オブジェクト参照が返されます。この参照を使用して、オブジェクトにアクセスできます。実際、この方法を使用すると、最も速くデータ・オブジェクトにアクセスできます。

複数のオブジェクトを効率的に作成するには、okCreateObjs をコールします。okCreateObjs 関数は、ループ内では okCreateObj よりも速く実行されます（okCreateObj 自体が、okCreateObjs を使用してオブジェクトを作成しています）。

オブジェクトの削除

グループからオブジェクトを削除するには、okDeleteObj または okDeleteObjs にそのオブジェクトへの参照を渡します。オブジェクトへの参照は、反復によって、またはそのオブジェクトを作成した okCreateObj 関数から取得できます。

```
err  = okDeleteObj( env, objRef );
```

オブジェクトの配列の場合は、okDeleteObjs を使用して、引数にその配列への参照を指定します。

属性の設定

オブジェクトの属性を更新するには、関数 okSetAttrVal を使用します。属性参照を検索する方法とオブジェクトの属性値を設定する方法を、次の例に示します。

```
okRef  attrRef;
okU4B  attrPos;
okName  attrNamePtr  = "My Age";
okSize  attrNameLen  = StrLen(attrNameLen);

// Get the attributes reference
err  = okFindAttr( env, classRef, attrNameLen, attrNamePtr,
                   0, &attrRef, &attrPos );

// Set up the okAttrVal
okU4B  age  = 28;  // Buffer to hold the new value
okAttrVal  attrVal;
attrVal.BufPtr  = &age;
attrVal.BufSize  = sizeof(age);

// Set the attribute
err  = okSetAttrVal( env, objRef, attrRef, &attrVal );
```

最初に、okFindAttr 関数を使用して設定する属性への参照を取得します。次に、BufPtr フィールドの属性値インスタンス attrVal に、age に格納された値への参照を割り当てます。属性値構造体 okAttrVal のフォーマットは、次のとおりです。

```
typedef struct okAttrVal okAttrVal_s;
struct okAttrVal {
   okU4B   BufSize;    /* size of the buffer */
   okByte  *BufPtr;    /* pointer to the buffer */
   ok4B    Indicator;  /* return actual size */
};
```

次の表に、属性値構造体のフィールドの説明を示します。

フィールド	説明
BufSize	BufPtr で指定されたバッファのサイズ（バイト数）。可変長属性を設定する場合は、属性のサイズを変更してその属性にコピーするバイト数を指定します。
BufPtr	設定するデータへのポインタ。たとえば、文字列属性を設定する場合、これはその文字列を示すポインタになります。先に示した例では、属性タイプの設定に従って、BufPtr は長整数へのポインタになっています。
Indicator	okSetAttrVal へのコールで返される値（この値を設定する必要はありません）。Indicator は、属性に書き込まれた実際のバイト数を示します。

最後に、okSetAttrVal をコールして属性に新規の値を設定します。

他の関数を使用しても、同じように属性を設定できます。たとえば、okSetAttrAt は okSetAttrVal とは異なる引数を使用しますが、同様に機能します。

動的配列属性

動的配列である属性を操作するときは、BufSize に sizeof(okArray) を設定し、BufPtr を okArray 型の変数を示すポインタにする必要があります。属性を取得すると、カーネルは返されたデータの実際のサイズを Indicator フィールドに設定します。Indicator が負数のときは、属性の値は NULL です。

オプションとして、OKAPI がその動的配列を作成する必要があるかどうかを示すフラグを attrVal.BufSize 内に設定できます。そうすると、次のように、attrVal.BuffPtr 内部の文字列を渡せます。

```
attrVal.BufSize = attrVal.Indicator = OK_NEW_ARRAY | strlen((char *)
attrVal.BufPtr);
```

イテレータによる問合せの実行

実行時に複合問合せを構成するために、反復オブジェクトを使用できます。最初に、条件の配列を定義する必要があります。各条件は、事前定義された構造体 okSearchCond_s によって宣言されています。そのフォーマットを次に示します。

```
typedef struct okSearchCond {
    okU4B  AttrPos;        /* 属性の相対的な位置 */
    okU1B  Operator;       /* 比較する演算子 */
    okU1B  Connective;     /* 次の条件との結合方法 */
    okU1B  Collate;        /* オプションの照合順番 */
    okU1B  _Padding;       /* 使用しません */
    okAttrVal_s  AttrVal;  /* 比較する値 */
}  okSearchCond_s;
```

次の表に、okSearchCond_s 構造体のフィールドの説明を示します。

フィールド	説明
AttrPos	第 1 オペランドとして使用される属性の位置。0 から始まります。
Operator	式の演算子。
Connective	この条件と次の条件の結合方法。
Collate	サポートされません。現在は使用されません。
_Padding	埋込みにのみ使用されます。このフィールドを設定する必要はありません。
AttrVal	第 2 オペランドの値。

フィールド _Padding および Collate は使用されないので、設定する必要はありません。しかし、その他のフィールドはすべて設定する必要があります。

フィールド AttrPos は、左オペランドとして使用される属性の位置です。最初の属性の位置番号は 0 から始まります。すべての属性は、クラスが作成されたときに okAttrDesc 配列に渡された順序で位置づけされます。その順序がわからない場合は、okFindAttr をコールして属性の位置を取得します。

AttrVal 構造体には、属性と比較するための値があります。Operator フィールドには、属性と AttrVal に与えられたデータとの比較方法を定義します。次の表に、サポートされる演算子のリストを示します。

Operator	説明
OK_EQ	等しい（=）
OK_LT	より小さい（<）
OK_LE	等しいかより小さい（<=）
OK_GT	より大きい（>）
OK_GE	等しいかより大きい（>=）
OK_NV	NULL である
OK_NN	NULL でない

設定されたことがない属性は、NULL とみなされます。

Connective フィールドには、現在の条件が配列にある次の条件とどのように結合するかを記述します。

Connective	説明
OK_AND	論理 AND（&&）
OK_OR	論理 OR（||）

たとえば、式 "age == 5" を構成するとき、age は最初の属性（位置 0）で、次のように使用できます。

```
// Set up the data as the second operand (5)
okU4B               age  = 5;

// Set up the condition
okSearchCond_s      cond;
cond.AttrPos  = 0;  // First attribute
cond.Operator  = OK_EQ;  // Equals to
ond.attrVal.BufPtr  = (okByte*)&age;
cond.attrVal.BufSize  = sizeof(age);
```

この場合は条件が 1 つしかないため、Connective フィールドを設定する必要はありません。

次の例は、式 "(age >= 21) && (age < 65) || (name == "Tony")" を構成する方法を示します。name は 3 番目の属性（位置は 2）です。

```
// Second operands
okU4B            age1  = 21,
                 age2  = 65;
char             *name  = "Tony";

// Set up the conditions
okSearchCond_s   cond[3];

// ( age >= 21 ) &&
cond[0].AttrPos  = 0;
cond[0].Operator  = OK_GE;
cond[0].attrVal.BufPtr  = (okByte*)&age1;
cond[0].attrVal.BufSize  = sizeof(age);
cond[0].Connective    = OK_AND;

// ( age < 65 ) ||
cond[0].AttrPos  = 0;
cond[0].Operator  = OK_LT;
cond[0].attrVal.BufPtr  = (okByte*)&age2;
cond[0].attrVal.BufSize  = sizeof(age);
cond[1].Connective    = OK_OR;

// Name == "Tony"
cond[0].AttrPos  = 2;    // 3rd attribute
cond[0].Operator  = OK_EQ;
cond[0].attrVal.BufPtr  = (okByte*)name;
cond[0].attrVal.BufSize  = StrLen(name);
```

検索条件を定義すると、okCreateIterator をコールして条件をコンパイルし、索引を解決し、反復を作成できます。

```
okIterator  iterHandle;
err         = okCreateIterator( env, groupRef, classRef, true,
                                3, &(conds[0]),
                                0, 0, NULL,
                                &iterHandle );
```

グループの中には異なるクラスのインスタンスとして生成されたオブジェクトも含まれることがあるため、クラス・オブジェクトへの参照を提供する必要があります（例では classRef として渡しています）。

5 番目の引数は、6 番目の引数として渡された条件の数を示します。

最後の引数 &iterHandle は、作成されたイテレータ・オブジェクトへのハンドルとして返される値を格納するバッファへのポインタです。

イテレータを作成すると、検索したオブジェクトのリストを関数 okIterate を使用して 1 つずつ調べることができます。検索条件に合致するすべてのオブジェクトを取得するには、次のようにループを構成します。

```
okRef  foundRef;
while ( !OK_IS_ERROR( okIterate( env, iterHandle, OK_NEXT, &foundRef ) ) )
{
   ... // Do what you want with the found object
}
```

3 番目の引数は Action パラメータです。設定できる値は次のとおりです。

- OK_NEXT: 次のオブジェクトに移動します。
- OK_PRIOR: 前のオブジェクトに移動します。
- OK_FIRST: 最初のオブジェクトに移動します。
- OK_LAST: 最後のオブジェクトに移動します。

問合せを完了するときは、メモリーを解放するために、okDeleteIterator を使用してイテレータを削除する必要があります。

コミット・トランザクションおよびロールバック・トランザクション

okTransact 関数は、データベースへの変更をコミットおよびロールバックします。データベースへの変更をコミットして永続的にするときは、action に OK_COMMIT_CONTINUE を設定して okTransact をコールします。たとえば、次のようになります。

```
okTransact( env, OK_COMMIT_CONTINUE );
```

1 番目の引数 env は、現在の環境変数です。2 番目の引数は実行する action です。

変更を廃棄するには、action に OK_ROLLBACK_CONTINUE を設定して okTransact を使用します。

```
okTransact( env, OK_ROLLBACK_CONTINUE );
```

現在サポートされている action の値は、OK_COMMIT_CONTINUE および OK_ROLLBACK_CONTINUE のみです。

okFinal をコールする前に変更をコミットしていない場合は、トランザクションは自動的にロールバックされます。アプリケーションがクラッシュしたり、他の理由で異常終了したときは、トランザクションはコミットされません。これによってデータベースの整合性が維持されます。

索引の作成

反復（問合せ）を高速化するために、索引を作成できます。索引を作成するには、関数 okCreateIndex をコールします。この関数には、次のパラメータを渡します。

- 環境
- グループへの参照
- クラスへの参照（異なるクラスのインスタンスとして生成されたオブジェクトが同じグループの中に含まれることがあるため、このパラメータは必須です。）
- キーを構成するために使用する属性の数
- キーを構成するために使用するすべての属性の位置
- 索引の種類
- その他のオプションの引数

たとえば、次のようになります。

```
// Attribute positions (attribute number 1, 4 and 5)
okU4B attrPoses[] = { 1, 4, 5 };
okU4B attrCount    = 3;
okU2B retIndexNo;
okError err = okCreateIndex( env,           // 現行の環境
                    groupRef, classRef,     // グループおよびクラスへの参照
                    attrCount, attrPoses,   // キーを構成する属性
                    OK_BTREE,               // 作成する索引の種類
                    OK_PRIMARY_KEY,         // キー制約オプション
                    &retIndexNo );          // 返された索引ナンバー
```

ここでは、classRef で参照されたクラスの第 1、第 4 および第 5 属性に基づいた索引を作成しています。

groupRef は、索引を作成するグループへの参照です。

1 つのグループの中に、いくつでも索引を作成できます（クラスが同じものでもかまいません）。ただし、同じグループの中に重複索引（グループ参照、クラス参照および属性リストが同じもの）は作成できません。

最初に索引を作成したときに、OKAPI は指定されたグループのオブジェクトをスキャンします。OKAPI は、指定されたクラスのインスタンスとして生成されたオブジェクトを検出すると、それを索引に追加します。

索引は実行時に自動的にメンテナンスされます。したがって、ユーザーはデータベースを変更した後でも索引を再作成する必要はありません。しかし、このメンテナンスはパフォーマンスに影響します。オブジェクト更新（たとえば、okCreateObjs または okSetAttrVal などによる）は、索引のメンテナンスに余分な時間が必要になるため、処理時間が長くなります。

Palm プラットフォームでは、共有ライブラリとして OKAPI にリンクしている場合、索引エンジン共有ライブラリ（libidx.prc）にもリンクできます。libidx.prc にリンクしない場合、関数 okCreateIndex はエラーを返します。

okCreateIterator では可能なかぎりいつでも最も効果的な索引を選択することに注意してください。検索条件をサポートする索引がない場合、OKAPI は単にクラスに属するグループのすべてのオブジェクトに対して線形検索を実行します。

注意： 索引オプティマイザは、現在は OK_OR 結合を伴う検索条件をサポートしていません。この条件を使用する場合、索引は使用できません。

2

OKAPI 関数

この章では、Oracle Lite オブジェクト・カーネル API（OKAPI）の関数を詳しく説明します。説明する内容は、次のとおりです。

- プラットフォーム互換性
- OKAPI データ型
- オブジェクトの構造
- オブジェクト・ポインタの変換
- 環境
- データベース
- スキーマ
- グループ
- オブジェクト
- オブジェクト情報と制御
- 属性値
- メソッドの起動
- イテレータ（問合せ）
- 索引
- オブジェクトのネーミング
- 関係
- バイナリ・ラージ・オブジェクト（BLOB）
- トランザクション
- 動的配列

プラットフォーム互換性

Oracle Lite のこのリリースでは、OKAPI は次のプラットフォームをサポートします。

- Win32
- Palm
- EPOC

汎用 OKAPI インタフェースの関数は、このマニュアルで定義されているとおり、そのほとんどすべてがそれぞれのプラットフォームで実装されていますが、中には関数コードがないものもあります。各関数は、プラットフォームに応じて次のいずれかのカテゴリに割り当てられます。

互換性あり

互換性のある関数でも、特定のプラットフォームと汎用の OKAPI 実装では内部処理が異なる可能性がありますが、コール側に対する外部的な機能はすべてのプラットフォーム上で同一です。このような関数は、**C** という文字で指定されます。

ダミー・スタブとして実装され、常に正数の結果を返す

これらの関数に内部的な処理はありませんが、常に正数の結果を返します。これらのコールを使用するプログラムは正常にコンパイルされ、異常な動作は検出されません。このような関数は、**P** という文字で指定されます。

ダミー・スタブとして実装され、常に負数の結果を返す

これらの関数に内部的な処理はなく、常に負数の結果（OK_NOT_SUPPORTED）を返します。これらのコールを使用するプログラムは正常にコンパイルされますが、そのコールが実行されるとランタイム・エラーが発生します。このような関数は、**N** という文字で指定されます。

実装されない

これらの関数は、与えられたプラットフォームのライブラリに含まれていません。これらの関数を使用するプログラムはコンパイルできません。このような関数をコールする文は、コメント化する必要があります。このような関数は、**NI** という文字で指定されます。

注意： OKAPI のヘッダー・ファイル **okapi.h** には、このマニュアルで説明していないデータ構造体や関数プロトタイプが入っています。これらは下位互換性に対応するためのものですから、使用しないでください。

OKAPI データ型

OKAPI は、永続的データ型と実行時のデータ型を定義して、OKAPI データベース・アプリケーションがマシンやコンパイラに依存しないようにします。OKAPI では、次のような永続的データ型が定義されています。

データ型	説明
ok4B	4 バイトの符号付き整数
okU4B	4 バイトの符号なし整数
ok2B	2 バイトの符号付き整数
okU2B	2 バイトの符号なし整数
ok1B	1 バイトの符号付き整数
okU1B	1 バイトの符号なし整数
okChar	文字
okFloat	単精度浮動小数点数
okDouble	倍精度浮動小数点数

OKAPI では、実行時に使用する、次の基本データ型も定義されています。

データ型	定義
okBool	ブール値
okByte	バイト
okError	エラー・コード
okLock	ロック・モード
okSize	サイズ型
okPtr	汎用ポインタ
okName	名前型
okObjData	永続オブジェクトの本体を指すポインタ

オブジェクトの構造

一般的なオブジェクト・システムでは、オブジェクトは変更不可能な一意識別子と、属性の値を保持するためのオブジェクト本体を持ちます。概念的には、オブジェクト A は別のオブジェクト B の一意識別子を使用して、B を参照します。実際には、オブジェクト参照は通常、プログラム内のメモリー・ポインタにマップされ、このポインタが一意識別子のカプセル化をポイントします。この一意識別子に、オブジェクト本体の位置に関する情報が入っています。オブジェクト参照（通常は一意のオブジェクト識別子として 2 次記憶域に格納される）をメモリー・ポインタに変換するプロセスは、「ポインタ・スウィズリング」として知られています。

通常、キャッシュされたオブジェクトには、記述子、ヘッダー、本体があります。記述子にはオブジェクトの ID やオブジェクト本体へのポインタ、オブジェクトについての他のメタ情報が含まれます。ヘッダーにはオブジェクトについての追加メタ情報が含まれます。オブジェクトの本体には、オブジェクトのデータ値が含まれます。「オブジェクト参照」（`okRef` 型）は、オブジェクトの記述子を指すポインタです。キャッシュ内にオブジェクトが固定されると、ポインタ（`okObjData` 型）を通じてオブジェクトの本体に直接アクセスできます。Oracle Lite データベースは、「オブジェクト・ハンドル」（`okHandle` 型）を使用して、オブジェクト参照またはオブジェクト本体を指す直接ポインタを示します。多くの OKAPI 関数では、入力引数として、オブジェクト参照とオブジェクト本体を指すポインタのどちらも使用できます。

オブジェクト・ポインタの変換

オブジェクトのポインタをある型から別の型に変換できますが、そのような変換は慎重に行う必要があります。いくつかの型変換では、有効な結果が得られないことがあります。たとえば、オブジェクト参照はオブジェクトの本体ではなくオブジェクトの識別子を指すので、オブジェクト参照（`okRef`）を `okObjData` に変換しないでください。オブジェクト参照によってオブジェクトの本体に直接アクセスしようとすると、間違った結果が返され、メモリー保護障害を生じることがあります。

次の表に、オブジェクト・ポインタの変換の有効性を示します。凡例は次のとおりです。

- 「Auto」は変換の必要がない、または自動的に変換されることを示します。
- 「Cast」は、ポインタ型同士で安全にキャストできることを示します。
- 「?」は、慎重に変換を実行する必要があることを示します。変換しても正しい結果が得られないことがあります。
- 「X」は不適切な変換を示します。これらのポインタ型同士の変換は、実行しないでください。

	okRef	okHandle	okObjData	Foo *
okRef	Auto	Auto	X	X
okHandle	?/Auto	Auto	?	?/Cast
okObjData	X	Cast	Auto	Cast
Foo *	X	Cast	Cast	Auto

環境

すべてのデータベース操作は、環境オブジェクト（okEnv）のコンテキスト内で実行する必要があります。環境オブジェクトは、データベース操作を実行するための独立した環境を（オブジェクト・キャッシュ、メモリー・ヒープなど）作成します。環境には、1 つのカレント・トランザクションがあるか、トランザクションが 1 つもないかのどちらかです。環境の中では、複数のデータベースと同時に接続できます。

環境関数の互換性リファレンス

この表は、環境オブジェクトを管理する OKAPI 関数のリストです。

関数	用途	Win32	Palm	EPOC
okInit	アプリケーションのためにオブジェクト・キャッシュを作成し、システムを初期化します。	C	C	C
okFinal	環境を終了します。	C	C	C
okCleanup	環境を異常終了させます。	C	X	C
okGetEnvInfo	環境に関する情報を取得します。	C	X	C

okInit

アプリケーションのためにオブジェクト・キャッシュを作成し、システムを初期化します。

構文

```
okError okInit(const okEnvParams_s *EnvParams,
   okEnv *RetEnv);
```

パラメータ

EnvParams
環境ハンドル

RetEnv
データベース・ハンドル

コメント

okEnvParams_s 構造体に、環境のためのオプションを指定します。EnvParams が NULL の場合は、システムは環境のためのデフォルト・パラメータを使用します。EnvParams が NULL でない場合は、個々のパラメータにそのパラメータのデフォルトとして 0 または NULL を設定できます。

okEnvParams_s 構造体のフォーマットは、次のとおりです。

```
typedef struct okEnvParams_s {
    okU4B       StructSize;     /* 構造体のサイズ */
    okU4B       CacheSize;      /* 保持する非固定オブジェクト数 */
    okU4B       NumObjRefs;     /* 記述子のオプションの見積数 */
    okU4B       UserIDLen;      /* ユーザー ID のオプションの長さ */
    okName      UserIDPtr;      /* ユーザー ID のオプションのポインタ */
    okU4B       PasswdLen;      /* パスワードのオプションの長さ */
    okName      PasswdPtr;      /* パスワードのオプションのポインタ */
    okPtr       LocalHeap;      /* オプションのローカル・ヒープ・ハンドル */
    okPtr       CacheHeap;      /* オプションのキャッシュ・ヒープ・ハンドル */
    okU4B       Options;        /* オプションの環境オプション */
    okU1B       CharacterSet;   /* オプションのキャラクタ・セット環境 */
    okU1B       _Padding[3];    /* 使用しません */
} okEnvParams_s;
```

okFinal

環境を終了します。

構文

```
okError okFinal(okEnv Env);
```

パラメータ

Env
環境ハンドル

コメント

この関数は、環境 Env を終了します。実行中のすべてのデータベース接続をアプリケーションから切断し、Env によって割り当てられたすべてのランタイム・リソースを解放します。最後のトランザクションがまだアクティブの場合は、最後のトランザクションをコミットします。

okCleanup

環境を異常終了させます。

構文

```
void okCleanup(okEnv Env);
```

パラメータ

Env
環境ハンドル

コメント

この関数は、環境を異常終了します。実行中のすべてのデータベース接続をアプリケーションから切断し、Env によって割り当てられたすべての実行時リソースを解放します。最後のトランザクションがまだアクティブの場合は、最後のトランザクションを異常終了します。

okEnvInfo 構造体のフォーマットは、次のとおりです。

```
struct okEnvInfo {
   okU4B StructSize;     /* 構造体のサイズ */
   okU4B XactCount;      /* 現在実装されていません */
   okU4B MemoryUsed;     /* 現在実装されていません */
```

```
   okU4B MemoryFree;     /* 現在実装されていません */
   okU4B DescriptorsUsed; /* 現在実装されていません */
};
```

okGetEnvInfo

環境に関する情報を取得します。

構文

```
okError okGetEnvInfo(okEnv Env, okEnvInfo_s *RetEnvInfo);
```

パラメータ

Env
環境ハンドル

RetEnvInfo
環境情報を返すためのバッファ

環境のサンプル・プログラム

次のコードは、環境プロシージャの使用方法を示したものです。

```
#define OK_CS_ASCII 1
#include "okbasic.h"
#include "okerr.h"
#include "okapi.h"

okEnvParameters_s EnvParams;  /* Env parameters */
okEnvInfo_s EnvInfo;  /* Env information */
okEnv  Env;  /* Env handle */
okError  e; /* for returned error code */
...

   memset((char *)&EnvParams, 0, sizeof(okEnvParameters_s));
   EnvParams.StructSize = sizeof(okEnvParameters_s);
   EnvParams.CharacterSet = OK_CS_ASCII;
   e = okInit(&EnvParams, &Env);
   if (OK_IS_ERROR(e)) { /* error handling */ }
   (void)okFinal(Env);
...
```

データベース

データベースは論理的に関係し合ったオブジェクトの集合です。データベースは、オブジェクトを格納する前に初期化（またはフォーマット）する必要があります。初期化中に、データベースの基礎となる記憶域構造と記憶域割当て情報がディスク上で初期化されます。さらにカーネルにより、必要とするメタオブジェクトを保持するためのシステム管理オブジェクト・グループが作成されます。データベースが初期化されると、アプリケーションは、永続オブジェクトを格納するためのオブジェクト・グループを作成できます。データベース内のすべてのオブジェクトが不要になった場合、アプリケーションはデータベースを削除して記憶領域をリカバリする必要があります。

データベース内のオブジェクトにアクセスするには、その前にアプリケーションからデータベースに接続する必要があります。データベースへの接続には、データベースのアクティブ化やアプリケーション（クライアント）とデータベースを管理するサーバーとの間の通信リンクの設定が必要になります。

一度接続されると、アプリケーションはトランザクションのコンテキスト内にあるデータベースのオブジェクトにアクセスおよび更新できるようになります。データベース内のオブジェクトが不要になった場合は、アプリケーションをデータベースから切断できます。

データベースには、トランザクションのコンテキストの内部と外部のどちらからでも接続できます。ただし、データの整合性を確保するためには、トランザクションがアクティブなうちはデータベースを切断しないでください。

データベース関数の互換性リファレンス

この表は、OKAPI のデータベース管理関数のリストです。

関数	用途	Win32	Palm	EPOC
okCreateDatabase	データベースを作成します。	C	C	C
okDeleteDatabase	データベースを削除します。	C	C	C
okConnect	データベースへの接続を確立します。	C	C	C
okDisconnect	データベースへの接続を切断します。	C	C	C

okCreateDatabase

データベースを作成します。

構文

```
okError okCreateDatabase(okEnv Env,
   okSize DatabaseNameLen, okName DatabaseNamePtr,
   const okDatabaseParameters_s *DatabaseParams);
```

パラメータ

Env
環境ハンドル

DatabaseNameLen
データベース名の長さ

DatabaseNamePtr
新規データベースの名前

DatabaseParams
データベース作成のためのパラメータ

コメント

この関数は、名前 DatabaseNamePtr を使用してデータベースを作成します。各データベースは RAW パーティションまたはファイルにマップされるので、使用するオペレーティング・システムまたはファイル・システムによってデータベース名の長さは制限されます。

okDatabaseParameters 構造体には、新規データベースのパラメータが格納されます。そのフォーマットは、次のとおりです。

```
typedef struct okDatabaseParameters okDatabaseParameters_s;
struct okDatabaseParameters {
   okU4B  StructSize;    /* 構造体のサイズ */
   ok2B   VolumeID;      /* データベースのボリューム ID */
   okU2B  ExtentSize;    /* オプションのクラスタ・ページ数 */
   okU4B  DatabaseSize;  /* データベース・サイズ（KB）*/
   okSize LabelLen;      /* データベースのラベル長 */
   okName LabelPtr;      /* データベースのラベルのポインタ */
   okU4B  Options;       /* 現在使用されていません */
   okU1B  CharacterSet;  /* キャラクタ・セット */
   okU1B  _Padding[3];   /* 使用しません */
};
```

okDeleteDatabase

データベースを削除します。

構文

```
okError okDeleteDatabase(okEnv Env,
   okSize DatabaseNameLen, okName DatabaseNamePtr);
```

パラメータ

Env
環境ハンドル

DatabaseNameLen
データベース名の長さ

DatabaseNamePtr
削除するデータベースの名前

コメント

この関数は、名前 DatabaseNamePtr を持ったデータベースを削除し、そのデータベースに割り当てられたすべてのリソースを解放します。削除されたデータベース内のオブジェクトはリカバリされないので、データベースを削除する前に、データベース内のオブジェクトが不要だということを確認する必要があります。

Palm プラットフォームでは、データベースを削除する前にデータベース接続を切断する必要があります。詳細は、「okDisconnect」を参照してください。

okConnect

データベース接続を確立します。

構文

```
okError okConnect(okEnv Env,
   okSize DatabaseNameLen, okName DatabaseNamePtr,
   const okConnOpt_s *Options, okRef *RetDatabase);
```

パラメータ

Env
環境ハンドル

DatabaseNameLen
データベース名の長さ

DatabaseNamePtr
データベースの名前

Options
データベース接続のためのオプション。次の値が指定できます。

- OK_CONNECT_READ_ONLY: 読取り専用モードでデータベースと接続します。

RetDatabase
接続したデータベースを表す一時データベース・オブジェクトへの参照を返すためのバッファ

コメント

この関数は、名前 DatabaseNamePtr を使用してデータベースへの接続を確立します。RetDatabase には、接続したデータベースを表すデータベース・オブジェクトへの参照を返します。アプリケーションからデータベース内のオブジェクトになんらかの操作を実行する前に、データベースと接続する必要があります。1 つのアプリケーションから okConnect を複数回コールして同じデータベースに接続すると、同じデータベース・オブジェクトが返されます。

okDisconnect

データベース接続を切断します。

構文

```
okError okDisconnect(okEnv Env, okHandle Database);
```

パラメータ

Env
環境ハンドル

Database
データベース・ハンドル

コメント

この関数は、Database によって示されるデータベースとの接続を切断します。データベース内のすべてのオブジェクトをフラッシュし、接続を維持するための制御データ構造をすべて解放します。okDisconnect は、トランザクションの外側で起動する必要があります。

データベースのサンプル・プログラム

次のコードは、データベース・プロシージャの使用方法を示したものです。

```
#include "okbasic.h"
#include "okerr.h"
#include "okapi.h"

okDatabaseParameters DatabaseParams;  /* database parameters */
okRef  Database;  /* reference to database object */
okError  e;  /* for returned error code */

...

   memset((char *)&DatabaseParams, 0, sizeof(okDatabaseParameters_s));
   DatabaseParams.StructSize = sizeof(okDatabaseParameters_s);
   e = okCreateDatabase(Env, sizeof("staff.odb"), "staff.odb",  &DatabaseParam);
   if (OK_IS_ERROR(e)) { /* error handling */ }

   e = okConnect(Env, sizeof("staff.odb"), "staff.odb", 0,  &Database);
   if (OK_IS_ERROR(e)) { /* error handling */ }

   e = okDisconnect(Env, Database);
   if (OK_IS_ERROR(e)) { /* error handling */ }

   e = okDeleteDatabase(Env, sizeof("staff.odb"), "staff.odb");
   if (OK_IS_ERROR(e)) { /* error handling */ }

...
```

スキーマ

各データベースにはオブジェクト・スキーマが 1 つあります。オブジェクト・スキーマは、プログラム言語環境のクラス階層に似ています。クラスはデータベース内で一意の名前を持ち、0 個以上の属性およびメソッドを持ちます。属性は、リレーショナル・データベースにおける表の列やいくつかのプログラミング言語におけるインスタンス変数に似ています。メソッドは、クラスの個々のインスタンスに適用できるプロシージャまたは関数です。クラスの属性とメソッドは、まとめてクラスの「プロパティ」と呼ばれます。クラスは、1 つ以上のクラスからプロパティを継承できます。プロパティを継承するクラスを「サブクラス」と呼び、継承元のクラスを「スーパークラス」と呼びます。

カーネル内では、クラス・オブジェクトがクラスを表し、属性オブジェクトが属性を表し、メソッド・オブジェクトがメソッドを表します。クラス、属性およびメソッドの各オブジェクトがまとまって、クラスのスキーマ・オブジェクトを形成します。カーネルには、クラスを作成および削除する関数と、スキーマ・オブジェクトを名前で検索する関数があります。

新規クラスを定義するには、データ構造体 okAttrDesc_s で新規クラスの属性を指定します。カーネル・クラスに Java クラスをアタッチしてメソッドを定義します。

属性のデータ構造体は、次のとおりです。

```
typedef struct okAttrDesc okAttrDesc_s;
struct okAttrDesc {
    okU4B       NameLen;        /* 属性の名前の長さ */
    okName      NamePtr;        /* 属性の名前のポインタ */
    okU4B       DefinedInLen;   /* クラス名で定義されたオプションの長さ */
    okName      DefinedInPtr;   /* クラス名で定義されたオプションのポインタ */
    okU4B       DomainLen;      /* ドメインクラス名の長さ */
    okName      DomainPtr;      /* ドメインクラス名のポインタ */
    okU4B       Precision;      /* 属性値のオプションの精度 */
    okU1B       SdtScale;       /* オプションの SDT スケール */
    okU1B       SdtTypeCode;    /* オプションの SDT タイプ・コード */
    okU1B       SdtSubtypeCode; /* オプションの SDT サブタイプ・コード */
    okU1B       DefCollate;     /* オプションのデフォルト照合順番 */
    ok4B        Dimension;      /* データ・ユニット数 */
    okU4B       DefaultValSize; /* デフォルト値のオプションのサイズ */
    okByte      *DefaultValPtr; /* デフォルト値のオプションのポインタ */
    okU4B       ExtensionCount; /* 拡張オブジェクトのオプション数 */
    okHandle    *ExtensionHandles; /* 拡張ハンドルのオプション配列 */
    okU4B       Options;        /* オプションの属性オプション */
};
```

フィールド NamePtr は、クラスの属性名を指定します。通常、DefinedInPtr の値は、NULL または新規クラスの名前のどちらかです。どちらの値も、その属性がローカルに定義された属性であることを示しています。スキーマ・デザイナのような複雑なアプリケーションの場合は、コール側がデフォルトの継承メカニズムを省略して、属性の継承元になるスーパークラスを明示的に指定できます。DomainPtr は、属性値のクラス（型）名を指定します。たとえば、属性の型が okU4B であった場合、そのドメインは文字列 "okU4B" です。

Precision、SdtScale、SdtTypeCode および SdtSubtypeCode フィールドは、属性の追加の型情報を格納するために使用します。たとえば、SQL データ型の型情報を格納できます。カーネルがこれらのフィールドの意味を直接解釈することはありません。

> **注意：** 接頭辞「Sdt」は、SQL データ型の拡張である「標準」データ型を表します。これらのデータ型は、異なる言語間（たとえば、C++ と SQL 間）で共通のデータ型を共有できるように定義されています。

整合性をとるため、カーネルはすべての属性を配列として見ます。Dimension フィールドは、属性のエレメント数を指定します。たとえば、スカラー属性のディメンションは常に OK_SCALE（値は 1）で、動的属性のディメンションは OK_DYNAMIC です。Default フィールドを使用することで、コール側は属性に関するデフォルト情報を文字列として管理できます。ただし、カーネルがこの情報を解釈または利用することはありません。ExtensionHandles フィールドを使用することで、オブジェクト内の追加のメタ情報をアプリケーションに保存できます。カーネルは、アプリケーションにかわってこれらの拡張オブジェクトを管理します。ただし、カーネルが拡張オブジェクトの意味を解釈することはありません。最後に、Options フィールドを使用して、次のステータス・フラグ・ビットを追加指定できます。

フラグ	説明
OK_PRIVATE_MEMBER	プライベート・メンバー属性
OK_PUBLIC_MEMBER	パブリック・メンバー属性
OK_PROTECTED_MEMBER	プロテクト・メンバー属性
OK_INHERITED_MEMBER	スーパークラスから継承された属性
OK_COPIED_MEMBER	コピーとして継承された属性
OK_NULLABLE_ATTR	NULL 値をとることができる属性
OK_RUNTIME_ATTR	永続的記憶域を持たないランタイム属性

スキーマ関数の互換性リファレンス

この表は、スキーマを管理する OKAPI 関数のリストです。

関数	用途	Win32	Palm	EPOC
okCreateClass	データベースに新しいクラスを作成します。	C	C	C
okAttachJavaClass	Java クラスを Oracle Lite クラスにアタッチします。	C	N	N
okDeleteClass	データベースからクラスを削除します。	C	C	C
okDetachJavaClass	Java クラスを Oracle Lite クラスからデタッチします。	C	N	N
okSetCallBacks	クラスのインスタンスに対してアクティベータ関数およびデアクティベータ関数を指定します。	C	X	C
okFindClass	クラス・オブジェクトを検索します。	C	C	C
okGetAttrCount	クラスの属性の数を返します。ただし、隠された属性は含まれません。	C	X	C
okFindAttr	クラス内の属性オブジェクトを検索します。	C	C	C
okFindMethod	クラス内のメソッド・オブジェクトを検索します。	C	N	N
okGetSuClasses	クラスのスーパークラスまたはサブクラスを返します。	C	C	C

okCreateClass

データベースに新しいクラスを作成します。

構文

```
okError okCreateClass(okEnv Env, okHandle Database,
   okSize ClassNameLen, okName ClassNamePtr,
   okSize SuperclassCount, const okHandle Superclasses[],
   okSize AttrCount, const okAttrDesc_s AttrDescs[],
   okSize MethCount, const okMethDesc_s MethDescs[],
   okU4B Options, okRef *RetClass);
```

パラメータ

Env
環境ハンドル

Database
データベース・オブジェクト

ClassNameLen
クラス名の長さ

ClassNamePtr
クラスの名前

SuperclassCount
継承元になっているスーパークラスの数

Superclasses
継承元になっているスーパークラスの配列

AttrCount
属性記述の数

AttrDescs
`okAttrDesc_s` 型の属性記述の配列

MethCount
メソッド記述の数。このパラメータは、常に 0 に設定されます。現在は使用されていません。

MethDescs
`okMethDesc_s` 型の属性記述の配列。このパラメータは、常に NULL に設定されます。現在は、使用されていません。

Options
クラスを作成するための追加情報

RetClass
クラス・オブジェクトを返すためのバッファ

コメント

この関数は、Database で表されるデータベース内に、新しいクラス ClassNamePtr を作成します。AttrDescs 内の属性記述の順序によって、クラス内の対応する属性の順序が決まります。現在は、MethCount を 0 に設定し、MethDescs を NULL に設定する必要があります。コール側は、Options を使用して、クラスに関する追加情報を渡すことができます。オプション OK_APP_DEF_INHERIT は、デフォルトのクラス継承を抑制します。このオプションを使用するときは、属性記述とメソッド記述をそれぞれどのプロパティから継承するか、個別に指定する必要があります。

注意： OK_APP_DEF_INHERIT オプションは、カーネルの最上位で実行する C++ 言語バインドなどの、非常に特殊なコールで使用します。

クラスの各プロパティは、継承されたプロパティかどうかを問わず、すべてスキーマ・オブジェクトとして表現され、クラスのコンテキストにおいて名前で検索できます。

okAttachJavaClass

Java クラスを Oracle Lite クラスにアタッチします。

構文

```
okError okAttachJavaClass(okEnv Env, okHandle Database,
   okHandle ClassRef, okU4B ClassDef, okSize JavaClsOrSrcNameLen,
   okName JavaClsOrSrcName, okSize JavaClsOrSrcPthLen,
   okName JavaClsOrSrcPth, okSize ConsArgCount,
   okHandle *ConsArgs);
```

パラメータ

Env
環境ハンドル

Database
データベース・オブジェクト

ClassRef
クラス・オブジェクト

ClassDef
クラス定義。次のいずれかの値が指定できます。

- CLS_IN_POLITE: そのクラスがすでに Oracle Lite データベースに存在している（このコールの前に、別のクラスにアタッチされている）ことを示します。
- CLS_CLS_FILE: パス JavaClsOrSrcPth のファイルがクラス・ファイルであることを示します。カーネルは、クラス・ファイルを Oracle Lite クラスにアタッチし、そのクラス・ファイルをデータベースに格納します。
- CLS_SRC_FILE: パス JavaClsOrSrcPth のファイルがソース・ファイルであることを示します。カーネルは、システムに設定されたクラス・パスに指定されている Java コンパイラを使用して、Java クラスをコンパイルします。

JavaClsOrSrcNameLen
Java クラスまたはソース・ファイル名の長さ

JavaClsOrSrcName
Java クラスの名前。パッケージ名を含んだ完全修飾名の場合、各構成要素をドットで区切る必要があります。

JavaClsOrSrcPthLen
Java ソースまたはクラス・ファイルへのパスの長さ

JavaClsOrSrcPth
Java クラスまたはソース・ファイルのディレクトリ・パス

ConsArgCount
Java インスタンスの作成に使用される、コンストラクタの引数の個数

ConsArgs
クラス・コンストラクタに引数として渡す、属性ハンドルの配列

コメント

この関数は、Java クラスを Oracle Lite クラスにアタッチします。アタッチ後は、Java クラスのあらゆるメソッドが Oracle Lite クラスのメソッドになります。Java クラスの静的メソッドは Oracle Lite クラスのクラス・メソッドになり、非静的メソッドはインスタンス・メソッドになります。ClassDef 引数は、クラスの位置指定に使用します。

Java クラスをアタッチするときに、カーネルが Java インスタンスの作成に使用するコンストラクタを指定できます。引数 ConsArgCount と ConsArgs は、コンストラクタの引数の数を指定し、またコンストラクタに渡すオブジェクトの属性を指定します。インスタンス・メソッドを実行する前に（「メソッドの起動」で説明した okExecMeth メソッドによってコールされたときに）、カーネルは ConsArgs から渡された属性値を引数にとるコンストラクタを使用して、Oracle Lite オブジェクトに対応した Java オブジェクトを作成します。

okDeleteClass

データベースからクラスを削除します。

構文

```
okError okDeleteClass(okEnv Env, okHandle Class);
```

パラメータ

Env
環境ハンドル

Class
クラス・オブジェクト

コメント

この関数は、オブジェクト `Class` によって指定されるクラスを削除します。このクラスに関連付けられるスキーマ・オブジェクトもすべて削除されます。Palm プラットフォームでは、スーパー・クラスを削除しても継承しているクラス・オブジェクトは自動的に削除されません。インスタンスのダングリング（参照先がない状態）を防ぐために、クラスを削除する前にクラスの各インスタンスを削除する必要があります。

okDetachJavaClass

Java クラスを Oracle Lite クラスからデタッチします。

構文

```
okError okDetachJavaClass(okEnv Env, okHandle Database,
    okHandle ClassRef, okBool DeleteClassBody);
```

パラメータ

Env
環境ハンドル

Database
データベース・オブジェクト

ClassRef
クラス・オブジェクト

DeleteClassBody
Java クラスの本体をデータベースから削除するかどうかを示す、ブール・フラグ

コメント

この関数は、Class に指定された Oracle Lite クラスから Java クラスをデタッチします。DeleteClassBody が TRUE の場合、okDetachJavaClass はそのクラスをデータベースから削除します。

okSetCallBacks

クラスのインスタンスに対して、実行時のアクティベータ関数およびデアクティベータ関数を指定します。

構文

```
okError okSetCallBacks(okEnv Env, okHandle Class,
   okPtr ActivatorData, okPtr DeactivatorData,
   okCallBack Activator, okCallBack Deactivator);
```

パラメータ

Env
環境ハンドル

Class
クラス・オブジェクト

ActivatorData
アクティベータ関数に渡すデータを指すポインタ

DeactivatorData
デアクティベータ関数に渡すデータを指すポインタ

Activator
クラスのインスタンスに対するアクティベータ関数

Deactivator
クラスのインスタンスに対するデアクティベータ関数

コメント

このプロシージャで、クラス Class のインスタンスに対して実行時アクティベータおよびデアクティベータ関数を指定できます。Oracle Lite は、NULL でない場合、クラスのインスタンス作成時またはディスクからキャッシュへのインスタンスの移動時に関数 Activator を起動します。Oracle Lite は、NULL でない場合、インスタンスの削除またはスワップ・アウト時に関数 Deactivator を起動します。

カーネルは、これらの関数をクラス・オブジェクト内に恒久的に保存することはありません。クラス・オブジェクトがディスク内へ（またはディスク外へ）移動されたときはいつでも、実行時にこれらの関数を設定する必要があります。

関数 Activator および Deactivator は、それぞれ引数として 2 つのポインタを使用します。1 つ目の引数は、（ActivatorData または DeactivatorData のどちらかとして）okSetCallBacks がコールされたときに得られます。2 つ目は、Class オブジェクトの本体を指すポインタです。

okFindClass

データベース内のクラス・オブジェクトを検索します。

構文

```
okError okFindClass(okEnv Env, okHandle Database,
   okSize ClassNameLen, okName ClassNamePtr, okRef *RetClass);
```

パラメータ

Env
環境ハンドル

Database
データベース・オブジェクト

ClassNameLen
クラス名の長さ

ClassNamePtr
クラスの名前

RetClass
クラス・オブジェクトを返すためのバッファ

コメント

この関数は、名前 ClassNamePtr を持ったクラス・オブジェクトをデータベース Database 内で見つけて、そのクラス・オブジェクトへの参照を RetClass に返します。このデータベース内でクラスを見つけられない場合は、okFindClass に NULL 参照を返します。NULL を返すことは、エラーではありません（つまり、エラー・コードは返されません）。

okFindAttr

クラス内の属性オブジェクトを検索します。

構文

```
okError okFindAttr(okEnv Env, okHandle Class,
    okSize AttrNameLen, okName AttrNamePtr, okHandle DefinedIn,
    okRef *RetObj, okU4B *RetPos);
```

パラメータ

Env
環境ハンドル

Class
クラス・オブジェクト

AttrNameLen
属性名の長さ

AttrNamePtr
属性の名前

DefinedIn
属性またはメソッドの継承元のクラス

RetObj
属性またはメソッド・オブジェクトを返すためのオプション・バッファ

RetPos
属性またはメソッドの位置を返すためのオプション・バッファ

コメント

この関数は、属性 `AttrName` の属性オブジェクトを `Class` によって指定されたクラス内で検索します。`DefinedIn` が NULL の場合は、デフォルトで `Class` に設定されます。クラス内の最初の属性の位置は 0 です。

okGetAttrCount

クラスの属性の数を返します。ただし、隠された属性は含まれません。

構文

```
okError okGetAttrCount(okEnv Env, okHandle Class, okU4B *RetCount)
```

パラメータ

Env
環境ハンドル

Class
クラス・オブジェクト

RetCount
属性の数を返すためのオプション・バッファ

コメント

この関数は、隠された属性を除外しない DAR_COUNT(class->Attributes) のかわりに使用する必要があります。この関数は、今後のリリースで Win32 対応になります。

okFindMethod

クラス内のメソッド・オブジェクトを検索します。

構文

```
okError okFindMethod(okEnv Env, okHandle dbH,
   okHandle clsR, okU4B MethType, okSize MethNameLen,
   okName MethName, okSize MethSigLen, okName MethSig,
   okHandle *retMeth);
```

パラメータ

Env
環境ハンドル

dbH
データベース・オブジェクト

clsR
クラス・オブジェクト

MethType
検索する Java メソッドの種類。次のいずれかの値が指定できます。

- OK_CLASS_METHOD: クラス・メソッドを検索する場合
- OK_INSTANCE_METHOD: インスタンス・メソッドを検索する場合

MethNameLen
メソッド名の長さ

MethName
メソッドの名前

MethSigLen
メソッドの署名の文字列としての長さ。名前のみでメソッドを検索する場合は、この値を 0 に設定します。

MethSig
Java Native Interface(JNI) スタイルの、メソッド署名文字列

retMeth
属性またはメソッド・オブジェクトを返すためのオプション・バッファ

コメント

このプロシージャは、メソッド MethName のメソッド・オブジェクトを Class で指定されたクラス内で検索します。MethType 引数には、検索するメソッドの型を指定します。設定できる値は、クラス・メソッドの場合は OK_CLASS_METHOD、インスタンス・メソッドの場合は OK_INSTANCE_METHOD です。同名のメソッドが複数存在する場合があるので、引数 MethSigLen と MethSig にメソッドの署名を指定できます。メソッドの署名は、JNI 形式の文字列として指定する必要があります。たとえば、メソッド String M (int, Date) の場合、JNI 形式の署名は次のとおりです。

```
"(ILjava/sql/Date;)Ljava/lang/String;"
```

okGetSuClasses

クラスのスーパークラスまたはサブクラスを返します。

構文

```
okError okGetSuClasses(okEnv Env, okHandle Class,
   okBool isSuper, okBool isImmediate, okArray *RetList);
```

パラメータ

Env
環境ハンドル

Class
クラス・オブジェクト

isSuper
ブール・フラグで、TRUE にするとスーパークラス、FALSE にするとサブクラスが返されます。

isImmediate
ブール・フラグで、TRUE にするとすべてのスーパークラスまたはサブクラスが返され、FALSE にすると直接のスーパークラスまたはサブクラスのみが返されます。

RetList
検索結果を返すための配列

コメント

このプロシージャは、クラスが Class のスーパークラス（isSuper が TRUE のとき）またはサブクラス（isSuper が FALSE）を RetList に返します。

スキーマのサンプル・プログラム

次のコードは、スキーマ・プロシージャを示したものです。

```
#include "okbasic.h"
#include "okerr.h"
#include "okapi.h"

okError e;
okEnv Env;
okHandle Database;
okRef PersonRef, StudentRef, DeptRef, MethRef, NameAttr;
okArray AttrDescs;
okU4B NamePos;

...
   okAttrDesc_s DeptAttrs[] = {
      {sizeof("Name"), "Name", 0, NULL, sizeof("okChar"), "okChar",
       0, 0, 0, 0, 0, OK_DYNAMIC, 0, NULL, 0, NULL, 0},
      {sizeof("Students"), "Students", 0, NULL,
      sizeof("Student_s"),
      "Student_s", 0, 0, 0, 0, 0, OK_DYNAMIC, 0, NULL, 0, NULL, 0},
      {sizeof("Staff"), "Staff", 0, NULL, sizeof("Professor_s"),
```

```
   "Professor_s", 0, 0, 0, 0, 0, OK_SCALAR, 0, NULL, 0, NULL, 0}
};

okAttrDesc_s PersonAttrs[] = {
   {sizeof("ID"), "ID", 0, NULL, sizeof("okU4B"), "okU4B",
   0,0, 0, 0, 0, OK_SCALAR, 0, NULL, 0, NULL, 0},
   {sizeof("Name"), "Name", 0, NULL, sizeof("okChar"), "okChar",
   0, 0, 0, 0, 0, OK_DYNAMIC, 0, NULL, 0, NULL, 0},
   {sizeof("Age"), "Age", 0, NULL, sizeof("okU2B"), "okU2B",
   0, 0, 0, 0, 0, OK_SCALAR, 0, NULL, 0, NULL, 0},
   {sizeof("Gender"), "Gender", 0, NULL, sizeof("okU2B"), "okU2B",
   0, 0, 0, 0, 0, OK_SCALAR, 0, NULL, 0, NULL, 0},
   {sizeof("Phone"), "Phone", 0, NULL, sizeof("okU1B"), "okU1B",
   0, 0, 0, 0, 0, 8, 0, NULL, 0, NULL, 0},
   {sizeof("Picture"), "Picture", 0, NULL, sizeof("okLong"),
   "okLong", 0, 0, 0, 0, 0, OK_SCALAR, 0, NULL, 0, NULL, 0}
};
okAttrDesc_s StudentAttrs[] = {
   {sizeof("Dept"), "Dept", 0, NULL, sizeof("Dept_s"), "Dept_s",
   0, 0, 0, 0, 0, OK_SCALAR, 0, NULL, 0, NULL, 0},
   {sizeof("Clock"), "Clock", 0, NULL, sizeof("okU4B"), "okU4B",
   0, 0, 0, 0, 0, OK_SCALAR, 0, NULL, 0, NULL,OK_RUNTIME_ATTR}
};
okAttrDesc_s ProfessorAttrs[] = {
   {sizeof("Dept"), "Dept", 0, NULL, sizeof("Dept_s"), "Dept_s",
   0, 0, 0, 0, 0, OK_SCALAR, 0, NULL, 0, NULL, 0}
};

#include <time.h>
/* constructor function for run-time attr of "Student" */
void GetTime(okPtr NotUsed, okObjData *StudentP)
   {((Student_s *)StudentP)->Clock = (okU4B)clock(); }

e = okConnect(Env, 9, "staff.odb", FALSE, &Database);
if (OK_IS_ERROR(e)) { /* error handling */ }

/* create class "Dept_s" */
e = okCreateClass(Env, Database, sizeof("Dept_s"), "Dept_s",
    0, NULL, sizeof(DeptAttrs)/sizeof(okAttrDesc_s),
    DeptAttrs, 0, NULL, 0, NULL);
if (OK_IS_ERROR(e)) { /* error handling */ }

/* creates class "Person_s" */
e = okCreateClass(Env, Database, sizeof("Person_s"),"Person_s",
   0, NULL, sizeof(PersonAttrs)/sizeof(okAttrDesc_s),
   PersonAttrs, 0, NULL, 0, &PersonRef);
if (OK_IS_ERROR(e)) { /* error handling */ }
```

```
/* creates class "Student_s" */
e = okCreateClass(Env, Database, sizeof("Student_s"),
   "Student_s", 1, &PersonRef,
   sizeof(StudentAttrs)/sizeof(okAttrDesc_s), StudentAttrs,
   0, NULL, 0, &StudentRef);
if (OK_IS_ERROR(e)) { /* error handling */ }

/* creates class "Professor_s" */
e = okCreateClass(Env, Database, sizeof("Professor_s"),
   "Professor_s", 1, &PersonRef,
   sizeof(ProfessorAttrs)/sizeof(okAttrDesc_s),
   ProfessorAttrs, 0, NULL, 0, NULL);
if (OK_IS_ERROR(e)) { /* error handling */ }

/* set constructor for run-time attr */
e = okSetCallBacks(Env, StudentRef, NULL, NULL, GetTime, NULL);
if (OK_IS_ERROR(e)) { /* error handling */ }

e = okFindClass(Env, Database, sizeof("Dept_s"),
   "Dept_s", &DeptRef);
if (OK_IS_ERROR(e)) { /* error handling */ }

e = okFindAttr(Env, DeptRef, sizeof("Name"),
   "Name", NULL, &NameAttr, &NamePos);
if (OK_IS_ERROR(e)) { /* error handling */ }

/* attach a Java class to class Dept_s. Assume that the Java
source file is c:¥JavaSrc¥DeptMeths. Use the Name
attribute of Dept_s as a constructor argument */

e = okAttachJavaClass(Env, Database, DeptRef, CLS_SRC_FILE,
   9,"DeptMeths",10,
   "c:¥¥JavaSrc",1,&NameAttr);
if (OK_IS_ERROR(e)) { /* error handling */ }

/* find the void HireStaff(int ID, String Name) method */
e = okFindMethod(Env, Database, DeptRef, OK_INSTANCE_METHOD,
   9, "HireStaff", 22,
   "(ILjava/lang/String;)V", &MethRef);
if (OK_IS_ERROR(e)) { /* error handling */ }

{okArrayBaseClasses;
 e = okGetSuClasses(Env, StudentRef,
   TRUE, TRUE, &BaseClasses);
 if (OK_IS_ERROR(e)) { /* error handling */ }
}
```

```
    e = okDeleteClass(Env, PersonRef);
    if (OK_IS_ERROR(e)) { /* error handling */ }

...
```

グループ

オブジェクトのグループ化は、データベース内にオブジェクトの集合を配置、操作するための強力なメカニズムです。具体的には、オブジェクトのグループ化によって次の機能がサポートされます。

- ディスク上のオブジェクトのクラスタリング
- グループレベルのオブジェクトのロッキング

オブジェクトのグループは、データベースのコンテキストで作成します。グループ・オブジェクトとは各グループのことを表します。グループ・オブジェクトの名前は、データベース内で一意にしてください。新しいオブジェクトを作成するときは、そのオブジェクトが属するグループを指定できます。グループを指定しない場合、オブジェクトはシステムで管理されるデフォルトのグループ（アプリケーションで変更できます）に格納されます。

グループ関数の互換性リファレンス

この表は、グループを管理する OKAPI 関数のリストです。

関数	用途	Win32	Palm	EPOC
okCreateGroup	データベースにオブジェクトのグループを作成します。	C	C	C
okDeleteGroup	データベースからオブジェクトのグループを削除します。	C	C	C
okFindGroup	データベース内のオブジェクトのグループを検索します。	C	C	C
okSetDefaultGroup	カーネルによって管理されている現在のオブジェクトのデフォルト・グループを変更します。	C	C	C

okCreateGroup

データベースにグループ・オブジェクトを作成します。

構文

```
okError okCreateGroup(okEnv Env, okHandle Database,
   okSize GroupNameLen, okName GroupNamePtr,
   okU4B Options, okRef *RetGroup);
```

パラメータ

Env
環境ハンドル

Database
データベース・オブジェクト

GroupNameLen
グループ名の長さ

GroupNamePtr
新しいグループの名前

Options
グループ操作のためのオプション

RetGroup
グループ・オブジェクトを返すためのバッファ

コメント

この関数は、名前 GroupNamePtr を持ったオブジェクトのグループをデータベース Database 内に作成します。新しいグループ・オブジェクトに対する参照は、RetGroup に返されます。Options は、現在使用されていません。

okDeleteGroup

データベースからグループ・オブジェクトを削除します。

構文

```
okError okDeleteGroup(okEnv Env, okHandle Group);
```

パラメータ

Env
環境ハンドル

Group
削除するグループ・オブジェクト

コメント

この関数は、Group で指定されたグループを削除します。グループ・オブジェクトとそのグループ内のすべてのオブジェクトが削除されます。

okFindGroup

データベース内のグループ・オブジェクトを検索します。

構文

```
okError okFindGroup(okEnv Env, okHandle Database,
   okSize GroupNameLen, okName GroupNamePtr,
   okRef *RetGroup);
```

パラメータ

Env
環境ハンドル

Database
データベース・オブジェクト

GroupNameLen
グループ名の長さ

GroupNamePtr
新しいグループの名前

RetGroup
グループ・オブジェクトを返すためのバッファ

コメント

この関数は、名前 GroupNamePtr を持ったグループをデータベース Database 内で検索します。指定された名前のグループを見つけられない場合は、NULL 参照を返します。NULL を返すことは、エラーではありません（すなわち、エラー・コードは返されません）。

okSetDefaultGroup

カーネルによって管理されている現在のオブジェクトのデフォルト・グループを変更します。

構文

```
okError okSetDefaultGroup(okEnv Env, okHandle NewDefGroup,
      okRef *RetOldDefGroup);
```

パラメータ

Env
環境ハンドル

NewDefGroup
新しいデフォルト・グループ

RetOldDefGroup
グループを返すためのバッファ

コメント

この関数は、現在のオブジェクトのデフォルト・グループを NewDefGroup に指定されたグループに変更します。NewDefGroup が NULL の場合は、システム定義のデフォルト・グループに戻ります。RetOldRefGroup が NULL でない場合は、NewDefGroup をオブジェクトのデフォルト・グループに設定し、それまでのデフォルト・グループ・オブジェクトへの参照を RetOldDefGroup に返します。

グループのサンプル・プログラム

次のコードは、グループ・プロシージャの使用方法を示したものです。

```
#include "okbasic.h"
#include "okerr.h"
#include "okapi.h"

okRef FacultyRef;
okRef PrevDefGroup;
okError e;

...

   e = okCreateGroup(Env, Database, sizeof("Faculty"),
      "Faculty", 0, &FacultyRef);
   if (OK_IS_ERROR(e)) { /* error handling */ }

   e = okFindGroup(Env, Database, sizeof("Faculty"),
```

```
    "Faculty", &FacultyRef);
  if (OK_IS_ERROR(e)) { /* error handling */ }

  e = okSetDefaultGroup(Env, FacultyRef, &PrevDefGroup);
  if (OK_IS_ERROR(e)) { /* error handling */ }

  e = okDeleteGroup(Env, FacultyRef);
  if (OK_IS_ERROR(e)) { /* error handling */ }

...
```

オブジェクト

オブジェクトは、カーネルの基本的な単位です。すべての永続エンティティは、オブジェクトとしてモデル化されて格納されます。いくつかの重要なメタオブジェクトには、オブジェクトについての情報が含まれます。第 1 に、オブジェクトはクラスのインスタンス化によって生成されたものであるため、オブジェクトは、そのクラス・オブジェクトによって定義されます。第 2 に、すべてのオブジェクトがグループに格納されるため、グループ・オブジェクトがオブジェクトの永続的記憶域を決定します。オブジェクトに関連付けられた第 3 のメタオブジェクトはデータベース・オブジェクトで、これはグループ・オブジェクトから間接的に導出できます。

オブジェクト関数の互換性リファレンス

この表は、OKAPI のオブジェクト関数のリストです。

関数	用途	Win32	Palm	EPOC
okCreateObj	クラスに新しいインスタンスを作成します。	C	C	C
okDeleteObj	データベースからオブジェクトを削除します。	C	C	C
okFixObj	キャッシュ内のオブジェクトを固定し、オブジェクト本体へのポインタを返します。	C	C	C
okUnfixObj	オブジェクトを固定解除します。	C	C	C
okPrepForUpdate	オブジェクトの更新の準備をします。	C	C	C
okSetShortLock	オブジェクトのロックを取得します。	C	P	C
okCreateObjs	単一クラスからオブジェクトの配列を作成します。	C	C	C
okDeleteObjs	オブジェクトの配列を削除します。	C	C	C

関数	用途	Win32	Palm	EPOC
okSetShortLocks	オブジェクトの配列をロックします。	C	P	C
okPrepForUpdates	固定されたオブジェクトの配列の更新の準備をします。	C	NI	C
okFixObjs	オブジェクトの配列を固定し、各オブジェクト本体へのポインタを返します。	C	C	C
okUnfixObjs	オブジェクトの配列を固定解除します。	C	C	C

okCreateObj

クラスに新しいインスタンスを作成します。

構文

```
okError okCreateObj(okEnv Env, okHandle Group,
   okHandle Class, const void *InitialVal, okU4B Options,
   void *RetHandle);
```

パラメータ

Env
環境ハンドル

Group
オブジェクトを作成するグループ

Class
インスタンス化するクラス・オブジェクト

InitialVal
オブジェクトの値

Options
オブジェクト作成のオプション

RetHandle
オブジェクト本体を指すポインタを返すバッファ

コメント

この関数は、クラス Class の新しいインスタンスをグループ Group に作成します。Group が NULL の場合、カーネルではそのオブジェクトをシステムが管理するデフォルト・グループに割り当てます。オプションとして、オブジェクトの初期値を Val によって渡すことができます。オプション OK_FIX_OBJ（Options によって渡される）は、関数から戻るときに新規作成オブジェクトをキャッシュに固定するよう指定します。

okDeleteObj

データベースからオブジェクトを削除します。

構文

```
okError okDeleteObj(okEnv Env, okHandle Obj);
```

パラメータ

Env
環境ハンドル

Obj
削除するオブジェクト

コメント

この関数は、キャッシュからオブジェクト Obj を削除し、ディスク上の記憶領域を再生します。そのオブジェクトが削除されると、カーネルではそのオブジェクトに対するアクセスを拒否します。

okFixObj

キャッシュ内のオブジェクトを固定し、オブジェクト本体へのポインタを返します。

構文

```
okError okFixObj(okEnv Env, okHandle Obj,
   okLock LockMode, void *RetObjData);
```

パラメータ

Env
環境ハンドル

Obj
固定するオブジェクト

LockMode
そのオブジェクトに適用するロックのタイプ。OKAPI は、次のようなロック・モードを定義します。

ロック	説明
OK_NOLOCK	ロックしない
OK_ISLOCK	意図的な共有ロック
OK_IXLOCK	意図的な排他ロック
OK_SIXLOCK	共有を意図した排他ロック
OK_SLOCK	共有（読取り）ロック
OK_XLOCK	排他的（書込み）ロック
OK_ULOCK	更新ロック

RetObjData
オブジェクト本体を指すポインタを返すバッファ

コメント

この関数は、オブジェクト Obj をキャッシュに固定し、オブジェクト本体へのポインタを RetObjData に返します。カーネルは、固定されたオブジェクトがスワップ・アウトされないことを保証します。つまり、RetObjData に返されるメモリー・ポインタは、オブジェクトが固定されているかぎり有効です。LockMode は、そのオブジェクトに適用するロック・モードを指定します。シングル・ユーザー・モードでは、すべてのロック要求は無視されます。

この関数を使用すると、オブジェクトの内容を直接操作できます。返されたメモリー・ポインタを適切なポインタ型に変換することで、C の構造体の場合とまったく同様にオブジェクト本体にアクセスできます。OKAPI 関数のかわりにメモリー・ポインタを使用してデータにアクセスすることが有効な場合もありますが、データ整合性の問題の原因にもなります。メモリー・ポインタによって属性値を変更した場合、okPrepForUpdate をコールする、または okUnfixObj に適切なフラグを渡すことで、その変更をカーネルに知らせる必要があります

メモリー・リークを回避するために、okFixObj をコールした後で okUnfixObj をコールして、不要になったオブジェクトを固定解除する必要があります。しかし、トランザクション中にアプリケーションが直接、間接に取り扱うオブジェクトを保持できるキャッシュ容量があることが確実な場合は、関数コールのオーバーヘッドを削減して、それらのオブジェクトを固定したままにしてもかまいません。各トランザクションの終了時には、カーネルでは自動的にキャッシュ内のすべてのオブジェクトを固定解除します

okUnfixObj

オブジェクトを固定解除します。

構文

```
okError okUnfixObj(okEnv Env,
   okHandle Obj, okBool isUpdated);
```

パラメータ

Env
環境ハンドル

Obj
オブジェクト・ハンドル

isUpdated
キャッシュに固定されている間にオブジェクトが更新されたかどうかを示すブール・フラグ

コメント

この関数は、オブジェクト Obj を固定解除し、isUpdated が TRUE の場合は、オブジェクトをダーティ（使用済み）にマークします。オブジェクトを作成するとき、または okFixObj をコールすることで、オブジェクトを固定できます。オブジェクトが固定解除されると、カーネルではいつでもそのオブジェクトをスワップ・アウトできます。通常は、キャッシュの空きがなくなりそうになったときに、そのオブジェクトをスワップ・アウトします。

（アプリケーションの様々なモジュールがそのオブジェクトを固定するときなど）オブジェクトが数回にわたって固定されることがあるので、カーネルはキャッシュ内のオブジェクトごとに固定した回数を記録しています。カーネルでは、アプリケーションがオブジェクトを固定したときにそのカウンタを増やし、okUnfixObj を使用してオブジェクトを固定解除したときに減らします。カーネルは、固定カウントが 0 にならないかぎり、オブジェクトをスワップ・アウトしません。

okPrepForUpdate

オブジェクトの更新の準備をします。

構文

```
okError okPrepForUpdate(okEnv Env, okHandle Obj);
```

パラメータ

Env
環境ハンドル

Obj
オブジェクト・ハンドル

コメント

この関数は、オブジェクト Obj の更新の準備をします。okPrepForUpdate はオブジェクトの書込みロック（または排他ロック）を取得し、そのオブジェクトをダーティ（使用済み）にマークします。この関数は、キャッシュに固定されているオブジェクトに適用されます。キャッシュ内に存在しない場合に、オブジェクトのスワップ・インを試みることはありません。また、オブジェクトの固定カウントを増やすこともありません。

okSetShortLock

オブジェクトのロックを取得します。

構文

```
okError okSetShortLock(okEnv Env, okHandle Obj,
   okLock LockMode);
```

パラメータ

Env
環境ハンドル

Obj
オブジェクト・ハンドル

LockMode
そのオブジェクトに適用するロックのモード。OKAPI は、次のようなロック・モードを定義します。

ロック	説明
OK_NOLOCK	ロックしない
OK_ISLOCK	意図的な共有ロック
OK_IXLOCK	意図的な排他ロック
OK_SIXLOCK	共有を意図した排他ロック

ロック	説明
OK_SLOCK	共有（読取り）ロック
OK_XLOCK	排他的（書込み）ロック
OK_ULOCK	更新ロック

コメント

この関数は、オブジェクト Obj に対してモード LockMode で指定したロックを取得します。このロック要求が、同じオブジェクトで実行中の他のロックと競合する場合、コール側はロック要求が認められるまでブロックされるか、デッドロックかタイムアウトによって要求を拒否されます。シングル・ユーザー・モードでは、すべてのロック要求は無視されます。

okCreateObjs

単一クラスからオブジェクトの配列を作成します。

構文

```
okError okCreateObjs(okEnv Env, okHandle Group, okHandle Class, const okByte
   *InitialVal, okU4B Options, okU4B ObjCount, okHandle RetHandles[]);
```

パラメータ

Env
環境ハンドル

Group
複数オブジェクトを作成するグループ

Class
インスタンス化するクラス・オブジェクト

InitialVal
オブジェクトの値

Options
オブジェクト作成のオプション

ObjCount
作成するオブジェクトの数

RetHandles
各オブジェクトの本体を指すポインタを返すバッファ

コメント

この関数は、指定された数（ObjCount）の、クラス Class の新しいインスタンスをグループ Group に作成します。Group が NULL の場合、カーネルではそのオブジェクトをシステムが管理するデフォルト・グループに割り当てます。オプションとして、各オブジェクトの初期値を InitialVal によって渡すことができます。オプション OK_FIX_OBJ（Options によって渡される）は、関数から戻ったときに新規作成オブジェクトをキャッシュに固定するよう指定します。

okDeleteObjs

データベースからオブジェクトの配列を削除します。

構文

```
okError okDeleteObjs(okEnv Env, okU4B ObjCount, okHandle Objs[]);
```

パラメータ

Env
環境ハンドル

ObjCount
削除するオブジェクトの数

Objs
削除するオブジェクトのハンドル

コメント

この関数は、キャッシュからオブジェクト Objs[] を削除し、ディスク上の記憶領域を再生します。オブジェクトが削除された後、カーネルはそのオブジェクトに対するアクセスを拒否します。

okSetShortLocks

オブジェクトの配列のロックを取得します。

構文

```
okError okSetShortLocks(okEnv Env, okU4B ObjCount, okHandle Objs[],
   okLock LockMode);
```

パラメータ

Env
環境ハンドル

ObjCount
ロックするオブジェクトの数

Objs
ロックするオブジェクトのハンドル

LockMode
そのオブジェクトに適用するロックのモード。OKAPI は、次のようなロック・モードを定義します。

ロック	説明
OK_NOLOCK	ロックしない
OK_ISLOCK	意図的な共有ロック
OK_IXLOCK	意図的な排他ロック
OK_SIXLOCK	共有を意図した排他ロック
OK_SLOCK	共有（読取り）ロック
OK_XLOCK	排他的（書込み）ロック
OK_ULOCK	更新ロック

コメント

この関数は、オブジェクト Objs[] に対してモード LockMode で指定したロックを取得します。このロック要求が、同じオブジェクトで実行中の他のロックと競合する場合、コール側はロック要求が認められるまでブロックされるか、デッドロックかタイムアウトによって要求を拒否されます。シングル・ユーザー・モードでは、すべてのロック要求は無視されます。

okPrepForUpdates

オブジェクトの配列の更新の準備をします。

構文

```
okError okPrepForUpdates(okEnv Env, okU4B ObjCount, okHandle Objs[]);
```

パラメータ

Env
環境ハンドル

ObjCount
準備するオブジェクトの数

Objs
準備するオブジェクトのハンドル

コメント

この関数は、オブジェクト配列 Objs の更新の準備をします。okPrepForUpdates はオブジェクトの書込みロック（または排他ロック）を取得し、そのオブジェクトをダーティ（使用済み）にマークします。この関数は、キャッシュに固定されているオブジェクトに適用されます。キャッシュ内に存在しない場合に、オブジェクトのスワップ・インを試みることはありません。また、オブジェクトの固定カウントを増やすこともありません。

okFixObjs

キャッシュ内にオブジェクトの配列を固定し、各オブジェクト本体へのポインタを返します。

構文

```
okError okFixObjs(okEnv Env, okU4B ObjCount, okHandle Objs[],
   okLock LockMode, void *RetObjData[]);
```

パラメータ

Env
環境ハンドル

ObjCount
固定するオブジェクトの数

Objs
固定するオブジェクト

LockMode
そのオブジェクトに適用するロックのタイプ。OKAPI は、次のようなロック・モードを定義します。

ロック	説明
OK_NOLOCK	ロックしない
OK_ISLOCK	意図的な共有ロック
OK_IXLOCK	意図的な排他ロック
OK_SIXLOCK	共有を意図した排他ロック
OK_SLOCK	共有（読取り）ロック
OK_XLOCK	排他的（書込み）ロック
OK_ULOCK	更新ロック

RetObjData
各オブジェクト本体を指すポインタを返すバッファ

コメント

この関数は、オブジェクト Objs[] をキャッシュに固定し、オブジェクト本体へのポインタを RetObjData に返します。カーネルは、固定されたオブジェクトがスワップ・アウトされないことを保証します。つまり、RetObjData に返されるメモリー・ポインタは、オブジェクトが固定されているかぎり有効です。LockMode は、そのオブジェクトに適用するロック・モードを指定します。シングル・ユーザー・モードでは、すべてのロック要求は無視されます。

この関数を使用すると、各オブジェクトの内容を直接操作できます。返されたメモリー・ポインタを適切なポインタ型に変換することで、C の構造体の場合とまったく同様にオブジェクト本体にアクセスできます。OKAPI 関数のかわりにメモリー・ポインタを使用してデータにアクセスすることが有効な場合もありますが、データ整合性の問題の原因にもなります。メモリー・ポインタによって属性値を変更した場合、okPrepForUpdates をコールする、または okUnfixObjs に適切なフラグを渡すことで、その変更をカーネルに知らせる必要があります。

メモリー・リークを回避するために、okFixObjs をコールした後で okUnfixObjs をコールして、不要になったオブジェクトを固定解除する必要があります。しかし、トランザクション中にアプリケーションが直接、間接に取り扱うオブジェクトを保持できるキャッシュ容量があることが確実な場合は、関数コールのオーバーヘッドを削減して、それらのオブ

ジェクトを固定したままにしてもかまいません。各トランザクションの終了時には、カーネルでは自動的にキャッシュ内のすべてのオブジェクトを固定解除します

okUnfixObjs

オブジェクトの配列を固定解除します。

構文

```
okError okUnfixObjs(okEnv Env, okU4B ObjCount,
   okHandle Objs[], okBool isUpdated);
```

パラメータ

Env
環境ハンドル

ObjCount
固定解除するオブジェクトの数

Objs
固定解除するオブジェクト

isUpdated
キャッシュに固定されている間に各オブジェクトが更新されたかどうかを示すブール・フラグ

コメント

この関数は、オブジェクト `Objs[]` を固定解除し、`isUpdated` が `TRUE` の場合は、オブジェクトをダーティ（使用済み）にマークします。オブジェクトのグループを作成するとき、または `okFixObjs` をコールすることで、オブジェクトのグループを固定できます。すべてのオブジェクトが固定解除されると、カーネルではいつでもそれらのオブジェクトをスワップ・アウトできます。通常は、キャッシュの空きがなくなりそうになったときに、そのオブジェクトをスワップ・アウトします。

（アプリケーションの様々なモジュールがそのオブジェクトを固定するときなど）オブジェクトが数回にわたって固定されることがあるので、カーネルはキャッシュ内のオブジェクトごとに固定した回数を記録しています。カーネルでは、アプリケーションがオブジェクトを固定したときにそのカウンタを増やし、`okUnfixObj` または `OkUnfixObjs` を使用してオブジェクトを固定解除したときに減らします。カーネルは、固定カウントが 0 にならないかぎり、オブジェクトをスワップ・アウトしません。

オブジェクトのサンプル・プログラム

次のコードは、オブジェクト関数の使用方法を示したものです。

```
#include "okbasic.h"
#include "okerr.h"
#include "okapi.h"

okEnv       Env;
okHandle    Database, Object;
okRef  Group, Class, ObjRef;
okObjData  ClassObj;
okError  e;

...

   /* assume Env and Database are set up properly */
   e = okFindGroup(Env, Database, sizeof("Faculty"),
      "Faculty", &Group);
   if (OK_IS_ERROR(e)) { /* error handling */ }
   e = okFindClass(Env, Database, sizeof("Professor_s"),  "Professor_s",
      &Class);
   if (OK_IS_ERROR(e)) { /* error handling */ }

   /* fix the class object then release it */
   e = okFixObj(Env, Class, OK_NOLOCK, (okObjData *)&ClassObj);
   if (OK_IS_ERROR(e)) { /* error handling */ }
   e = okUnfixObj(Env, (okHandle)ClassObj, FALSE);
   if (OK_IS_ERROR(e)) { /* error handling */ }

   /* create a new "Professor_s" object in "Faculty" group */
   e = okCreateObj(Env, Group, Class, NULL, OK_FIX_OBJ, &Object);
   if (OK_IS_ERROR(e)) { /* error handling */ }

   /* set an exclusive lock on the "Dept_s" object */
   e = okSetShortLock(Env, Object, OK_XLOCK);
   if (OK_IS_ERROR(e)) { /* error handling */ }

   /* release the new object then delete it */
   e = okUnfixObj(Env, (okHandle)Object, FALSE);
   if (OK_IS_ERROR(e)) { /* error handling */ }
   e = okDeleteObj(Env, Object);
   if (OK_IS_ERROR(e)) { /* error handling */ }

...
```

オブジェクト情報と制御

オブジェクト情報および制御関数を使用して、オブジェクトの問合せと制御ができます。

オブジェクト情報と制御関数の互換性リファレンス

この表は、OKAPI のオブジェクト情報と制御関数のリストです。

関数	用途	Win32	Palm	EPOC
okGetObjRef	オブジェクトの参照を取得します。	C	C	C
okIsInstanceOf	オブジェクトが特定のクラスのインスタンスかどうかを判断します。	C	C	C
okGetObjInfo	オブジェクトに関する情報を返します。	C	C	C
okIsObjEqual	2 つのオブジェクト・ハンドルが等価かどうかを判断します。	C	C	C
okCtrlObjs	コール側が、オブジェクト配列に対する制御操作を適用できるようにします。	C	P	C
okGetObjRefs	オブジェクトのハンドルの配列を取得します	C	NI	C
okFindRefs	与えられたオブジェクト ID の配列で、オブジェクト参照の配列を検索します	C	C	C

okGetObjRef

オブジェクトの参照を取得します。

構文

```
okError okGetObjRef(okEnv Env, okHandle Obj,
   okRef *RetRef);
```

パラメータ

Env
環境ハンドル

Obj
オブジェクト・ハンドル

RetRef
オブジェクトへの参照を返すバッファ

コメント

この関数は、オブジェクト Obj への参照 RetRef を返します。オブジェクト・ハンドルは、オブジェクト参照、またはオブジェクト本体を指すメモリー・ポインタです。Obj がすでにオブジェクト参照である場合は、そのオブジェクト参照を RetRef に返します。Obj がオブジェクト本体を指すメモリー・ポインタである場合、okGetObjRef はそのオブジェクトへのオブジェクト参照を探すか作成して、RetRef に返します。

okIsInstanceOf

オブジェクトが特定のクラスのインスタンスかどうかを判断します。

構文

```
okError okIsInstanceOf(okEnv Env, okHandle Obj,
   okHandle Class, okBool isImmediate, okBool *RetResult);
```

パラメータ

Env
環境ハンドル

Obj
オブジェクト・ハンドル

Class
クラス・オブジェクト

isImmediate
スーパークラスを考慮するかどうかを示すブール・フラグ

RetResult
オブジェクトが、あるクラスのインスタンスかどうかを示すブール値

コメント

この関数は、オブジェクト Obj がクラス Class のインスタンスかどうかを判断します。結果のブール値（TRUE または FALSE のどちらか）を、RetResult に返します。引数 isImmediate には、継承を考慮する必要があるかどうかを指定します。isImmediate が TRUE の場合は、Class が Obj のクラス・オブジェクトの場合にかぎって TRUE を返します。isImmediate が FALSE の場合は、Class が Obj のクラス・オブジェクトまたはスーパークラス・オブジェクトの場合に TRUE を返します。

Palm プラットフォームでは、Class が obj のスーパークラス・オブジェクトである場合、isImmediate が TRUE か FALSE かにかかわらずこの関数は FALSE を返します。

okGetObjInfo

オブジェクトに関する情報を返します。

構文

```
okError okGetObjInfo(okEnv Env, okHandle Obj,
   okOID_s *RetObjID, okRef *RetDatabase,
   okRef *RetGroup, okRef *RetClass, okObjInfo_s *RetInfo);
```

パラメータ

Env
環境ハンドル

Obj
オブジェクト・ハンドル

RetObjID
オブジェクト ID を返すためのオプション・バッファ

RetDatabase
オブジェクトのデータベースを返すためのオプション・バッファ

RetGroup
オブジェクトのグループを返すためのオプション・バッファ

RetClass
オブジェクトのクラスを返すためのオプション・バッファ

RetInfo
オブジェクト情報を返すためのバッファ

コメント

この関数は、オブジェクト Obj に関する、一意 ID（RetObjID）および関連付けられたメタオブジェクトなどの情報を返します。メタオブジェクトには、データベース・オブジェクト（RefDatabase）、グループ・オブジェクト（RetGroup）およびクラス・オブジェクト（RetClass）などがあります。RetInfo には、オブジェクトのバージョンおよびロック・モードが、次のフォーマットで返されます。

```
typedef struct {
    okU4B Version;
    wsLockMode LockMode;
} okObjInfo_s;
```

okIsObjEqual

2 つのオブジェクト・ハンドルが等価かどうかを判断します。

構文

```
okError okIsObjEqual(okEnv Env,
   okHandle Obj1, okHandle Obj2, okBool *RetResult);
```

パラメータ

Env
環境ハンドル

Obj1
比較するオブジェクト・ハンドル

Obj2
比較するオブジェクト・ハンドル

RetResult
結果を示すブール値

コメント

この関数は、2 つのオブジェクト・ハンドル `Obj1` と `Obj2` が等しいかどうかを判断します。2 つのハンドルが同じオブジェクトを指している場合にのみ、等しいと判断します。

okCtrlObjs

オブジェクト配列に制御操作を適用します。

構文

```
okError okCtrlObjs(okEnv Env, okU4B ObjCount,
   const okHandle ObjHandles[], okU4B Actions);
```

パラメータ

Env
環境ハンドル

ObjCount
制御操作内のオブジェクトの数

ObjHandles
制御操作内のオブジェクト

Actions
そのオブジェクトに適用する操作。Actions には、次のあらゆる組合せを指定できます。

- OK_FLUSH_OBJECT: ダーティ（使用済み）とマークされているオブジェクトのコピーを、ディスクに書き込みます。
- OK_FREE_OBJECT: オブジェクトの本体が使用しているメモリー領域を解放します。このオプションによって、オブジェクト本体を指す直接メモリー・ポインタはすべて無効になりますが、オブジェクトへのオブジェクト参照は無効になりません。そのオブジェクトがアプリケーションにとって不要になったときに、このオプションを使用します。
- OK_FREE_DESC: オブジェクトの記述子、およびキャッシュ内に存在する場合はオブジェクトの本体を解放します。

重要事項： OK_FREE_DESC オプションを使用すると、他のキャッシュ・オブジェクトに埋め込まれているものを含めて、既存のオブジェクト参照をすべて無効化します。これは、メモリーが極端に不足し、オブジェクト記述子によって使用されている領域をリカバリする必要があるときに使用します。

コメント

この関数は、オブジェクト ObjHandles の配列に対して、Actions で指定された制御操作を適用できるようにします

okGetObjRefs

オブジェクトの配列への参照を取得します。

構文

```
okError okGetObjRef(okEnv Env, okU4B ObjCount, okHandle Objs[],
   okRef *RetRefs[]);
```

パラメータ

Env
環境ハンドル

ObjCount
オブジェクトの数

Objs
オブジェクト・ハンドル

RetRefs
オブジェクトへの参照を返すバッファ

コメント

この関数は、RetRefs[] に Objs[] 内のオブジェクトへの参照を返します。オブジェクト・ハンドルは、オブジェクト参照、またはオブジェクト本体を指すメモリー・ポインタです。Objs[] がすでにオブジェクト参照である場合は、そのオブジェクト参照を RetRefs[] に返します。Objs[] がオブジェクト本体を指すメモリー・ポインタである場合、okGetObjRefs はそのオブジェクトへのオブジェクト参照を探すか作成して、RetRefs[] に返します。

okFindRefs

与えられたオブジェクト ID の配列で、オブジェクトの配列への参照を検索します

構文

```
ok Error okFindRefs,( okEnv Env, okU4B ObjCount, okOID_s ObjIDs[],
   okRef RetObjRefs[]);
```

パラメータ

Env
環境ハンドル

ObjCount
オブジェクトの数

ObjIDs
オブジェクト ID（配列）

RetObjRefs
オブジェクトへの参照を返すバッファ

コメント

この関数は、RetObjRefs[] に ObjIDs[] 内に指定されたオブジェクトへの参照を返します。

オブジェクト情報と制御のサンプル・プログラム

次のコードは、オブジェクト情報および制御プロシージャの使用方法を示したものです。

```
#include "okbasic.h"
#include "okerr.h"
#include "okapi.h"

okRef Group, Class, ObjRef;
okHandle Object, NewObj;
Professor_s *Advisor;
okBool isa, isEqual;
okError e;

...

   e = okFindGroup(Env, Database,
      sizeof("Faculty"), "Faculty", &Group);
   if (OK_IS_ERROR(e)) { /* error handling */ }
   e = okFindClass(Env, Database,
      sizeof("Professor_s"), "Professor_s", &Class);
   if (OK_IS_ERROR(e)) { /* error handling */ }

   /* create a new "Professor_s" object in "Faculty" group */
   e = okCreateObj(Env, Group, Class, NULL, OK_FIX_OBJ, &Object);
   if (OK_IS_ERROR(e)) { /* error handling */ }

   e = okGetObjRef(Env, Object, &ObjRef);
   if (OK_IS_ERROR(e)) { /* error handling */ }

   e = okIsInstanceOf(Env, Object, Class, TRUE, &isa);
   if (OK_IS_ERROR(e)) { /* error handling */ }

   {
      okOID_sObjID;
      okRefDatabase, Group, Class;

      e = okGetObjInfo(Env, Object, &ObjID, &Database, &Group,
         &Class, NULL);
      if (OK_IS_ERROR(e)) { /* error handling */ }
   }

   e = okCtrlObjs(Env, 1, &Object, OK_FLUSH_OBJECT|OK_FREE_DESC);
   if (OK_IS_ERROR(e)) { /* error handling */ }

   /* create a new "Dept_s" object */
   e = okCreateObj(Env, Group, Class, NULL, OK_FIX_OBJ, &NewObj);
   if (OK_IS_ERROR(e)) { /* error handling */ }
```

```
   e = okIsObjEqual(Env, Object, NewObj, &isEqual);
   if (OK_IS_ERROR(e)) { /* error handling */ }

...
```

属性値

オブジェクトの属性値は、オブジェクトの内部状態をまとめて表します。「スキーマ」の項で説明しているとおり、クラスの各属性は属性オブジェクトによって表されます。属性オブジェクトは属性を定義します。

属性値のアクセスまたは更新を行う関数も、パラメータとして属性オブジェクトを必要とします。たとえば、更新操作または検索を実行するためには、更新のための値または検索述語を準備する必要があります。OKAPI には、属性値を表す次のような構造体があります。

```
typedef struct okAttrVal_s { /* for holding attribute value */
   okU4B   BufSize;    /* size of buffer in bytes */
   okByte  *BufPtr;    /* pointer to buffer */
   ok4B    Indicator;  /* data size if >= 0, NULL if < 0 */
} okAttrVal_s;
```

`BufPtr` は、属性値を代入するためのバッファを指します。`BufSize` は、バッファのサイズをバイト数で指定します。`Indicator` には、属性値の実際のバイト数が含まれます。`Indicator` が負数のときは、NULL 値（未定義または未初期化値）であることを示します。

動的配列属性

動的配列である属性を操作するときは、`BufSize` に `sizeof(okArray)` を設定し、`BufPtr` を `okArray` 型の変数を示すポインタにする必要があります。属性を取得すると、カーネルは返されたデータの実際のサイズを `Indicator` フィールドに設定します。`Indicator` が負数のときは、属性の値は NULL です。

オプションとして、OKAPI がその動的配列を作成する必要があるかどうかを示すフラグを `attrVal.BufSize` 内に設定できます。そうすると、次のように、`attrVal.BuffPtr` 内部の文字列を渡せます。

```
attrVal.BufSize = attrVal.Indicator = OK_NEW_ARRAY | strlen((char *)
attrVal.BufPtr);
```

注意： フラグ `OK_NEW_ARRAY` は、Palm プラットフォームではサポートされず、必要でもありません。

属性値関数の互換性リファレンス

この表は、属性値を管理する OKAPI 関数のリストです。

関数	用途	Win32	Palm	EPOC
okGetAttrVal	属性の値を取り出し、値のコピーか、オブジェクト本体の属性値を指す直接メモリー・ポインタのどちらかを返します。	C	C	C
okSetAttrVal	属性の値を更新します。	C	C	C
okGetAttrValAt	指定された位置の属性の値を取り出し、値のコピーか、オブジェクト本体の属性値を指す直接メモリー・ポインタのどちらかを返します。	C	C	C
okSetAttrValAt	指定された位置の属性の値を更新します。	C	C	C
okSetAttrVals	リストに名前がある複数の属性の値を更新します	C	C	C
okSetAttrValsAt	指定された位置の各属性の値を更新します。	C	C	C

okGetAttrVal

属性の値を取り出し、値のコピーか、オブジェクト本体の属性値を指す直接メモリー・ポインタのどちらかを返します。

構文

```
okError okGetAttrVal(okEnv Env, okHandle Obj,
   okHandle AttrObj, okBool toCopy, okAttrVal_s *RetVal);
```

パラメータ

Env
環境ハンドル

Obj
オブジェクト・ハンドル

AttrObj
アクセスされる属性を表すオブジェクト

toCopy
ブール・フラグ

TRUE の場合、Win32 プラットフォームおよび EPOC プラットフォームでは、属性へのポインタへのポインタを返します。Palm プラットフォームでは、属性への直接のポインタを返します。

FALSE の場合、Win32 プラットフォームおよび EPOC プラットフォームでは、属性への直接のポインタを返します。Palm プラットフォームでは、このパラメータの FALSE 値はサポートされません。

RetVal
値を返すためのバッファ

コメント

この関数は、属性 AttrObj の値を取り出し、値のコピー（toCopy が TRUE の場合）またはオブジェクト本体の属性値を指す直接メモリー・ポインタ（toCopy が FALSE の場合）を RetVal に返します。toCopy が FALSE の場合、返されるメモリー・ポインタの有効性を保証するために、キャッシュ内のオブジェクトを暗黙的に固定します。したがって、okUnfixObj をコールして、明示的にオブジェクトの固定解除をする必要があります。この関数をコールする前に、オブジェクトをあらかじめ固定しておく必要はありません。

属性のサイズが小さいときは属性値のコピーを作成することで、オブジェクトの固定解除関数を明示的にコールすることによるオーバーヘッドを回避できます。しかし、動的または埋込み配列などのように属性が大きいときは、オブジェクトへのメモリー・ポインタを取得して属性値に直接アクセスするほうが効率的なことがあります。

属性に動的配列が含まれている場合、okGetAttrVal では動的配列を指すポインタを返します。返される属性値の型は okArray、つまり汎用ポインタ型です。配列の要素は、RetVal で提供されたバッファには返されません。toCopy が TRUE のとき、okGetAttrVal は動的配列の新しいコピーを作成します。配列のコピーが不要になった場合は、okDeleteArray を使用して明示的に解放する必要があります。ただし、オブジェクト自体を固定解除する必要はありません。toCopy が FALSE のときは、関数から返される動的配列を解放する必要はありませんが、okUnfixObj を使用して、そこに含まれるオブジェクトを明示的に固定解除する必要があります。

okSetAttrVal

属性の値を更新します。

構文

```
okError okSetAttrVal(okEnv Env, okHandle Obj,
      okHandle AttrObj, const okAttrVal_s *NewVal);
```

パラメータ

Env
環境ハンドル

Obj
オブジェクト・ハンドル

AttrObj
アクセスされる属性を表すオブジェクト

NewVal
新しい値を入れるバッファ

コメント

この関数は、オブジェクト Obj の AttrObj によって識別される属性の値を、NewVal に指定された新しい値で更新します。この関数をコールする前に、オブジェクトをあらかじめ固定しておく必要はありません。属性に動的配列が含まれている場合、NewVal バッファは動的配列を指す必要があります。この関数は、属性内の元の動的配列を解放し、NewVal に新しい動的配列を作成し、その新規配列をオブジェクトに格納します。

okGetAttrValAt

指定された位置の属性の値を取り出し、値のコピーか、オブジェクト本体の属性値を指す直接メモリー・ポインタのどちらかを返します。

構文

```
okError okGetAttrValAt(okEnv Env, okHandle Obj,
      okU4B Position, okBool toCopy, okAttrVal_s *RetVal);
```

パラメータ

Env
環境ハンドル

Obj
オブジェクト・ハンドル

Position
アクセスされる属性の位置

toCopy
ブール・フラグ

TRUE の場合、Win32 プラットフォームおよび EPOC プラットフォームでは、属性へのポインタへのポインタを返します。Palm プラットフォームでは、属性への直接のポインタを返します。

FALSE の場合、Win32 プラットフォームおよび EPOC プラットフォームでは、属性への直接のポインタを返します。Palm プラットフォームでは、このパラメータの FALSE 値はサポートされません。

RetVal
値を返すためのバッファ

コメント

この関数は、指定された位置 `Position` の属性の値を取り出します。返される値（`RetVal` に）は、`toCopy` が TRUE の場合は値のコピー、`toCopy` が FALSE の場合はオブジェクト本体の属性値を指す直接メモリー・ポインタです。`toCopy` が FALSE の場合、オブジェクトを暗黙的に固定して、返されるメモリー・ポインタの有効性を保証します。したがって、コール側は `okUnfixObj` でオブジェクトを固定解除する必要があります。この関数をコールする前に、オブジェクトを固定しておく必要はありません。詳細は、前述の `okGetAttrVal` を参照してください。

okSetAttrValAt

指定された位置の属性の値を更新します。

構文

```
okError okSetAttrValAt(okEnv Env, okHandle Obj,
   okU4B Position, const okAttrVal_s *NewVal);
```

パラメータ

Env
環境ハンドル

Obj
オブジェクト・ハンドル

Position
アクセスされる属性の位置

NewVal
新しい値を入れるバッファ

コメント

この関数は、オブジェクト Obj の位置 Position に置かれた属性の値を、NewVal に指定された新しい値で更新します。この関数をコールする前に、オブジェクトを固定しておく必要はありません。詳細は、前述の okSetAttrVals を参照してください。

okSetAttrVals

リストに名前がある複数の属性の値を更新します

構文

```
okError okSetAttrVals(okEnv Env, okHandle ObjH, okSize AttrCnt,
      okHandle AttrH, const okAttrVal_s *NewVal);
```

パラメータ

Env
環境ハンドル

ObjH
オブジェクト・ハンドル

AttrCnt
更新する属性の数

AttrH
更新する属性のハンドル

NewVal
新しい値を入れるバッファ

コメント

この関数は、オブジェクト ObjH の AttrObj によって識別される属性の値を、NewVal に指定された新しい値で更新します。この関数をコールする前に、オブジェクトをあらかじめ固定しておく必要はありません。属性に動的配列が含まれている場合、NewVal バッファは動的配列を指す必要があります。この関数は、属性内の元の動的配列を解放し、NewVal に新しい動的配列を作成し、その新規配列をオブジェクトに格納します。

okSetAttrValsAt

指定された位置の各属性の値を更新します。

構文

```
okError okSetAttrValsAt(okEnv Env, okHandle ObjH, okSize AttrPosCnt,
   okU4B *Position, const okAttrVal_s *NewVal);
```

パラメータ

Env
環境ハンドル

ObjH
オブジェクト・ハンドル

AttrPosCnt
更新する属性の数

Position
アクセスされる属性の位置を入れるバッファ

NewVal
新しい値を入れるバッファ

コメント

この関数は、オブジェクト ObjH の位置 Position に置かれた属性の値を、NewVal に指定された新しい値で更新します。この関数をコールする前に、オブジェクトを固定しておく必要はありません。詳細は、前述の okSetAttrVal を参照してください。

属性値のサンプル・プログラム

次のコードは、属性値プロシージャの使用方法を示したものです。

```
#include "okbasic.h"
#include "okerr.h"
#include "okapi.h"
#define DEPT_NAME "Computer Science"

okEnv Env;
okHandle  Object, Group;
okRef AttrRef, AttrRefs[2];
okArray DeptName;
okAttrVal_s Value, AttrVals[2];
okError  e;
```

```
okU4B AttrPoss[2];

...

   /* assume class "Dept_s" is in Class, find its attribute "Name" */
   e = okFindAttr(Env, Class,
      sizeof("Name"), "Name", NULL, &AttrRef, NULL);
   if (OK_IS_ERROR(e)) { /* error handling */ }

   /* create a new "Dept_s" object */
   e = okCreateObj(Env, Group, Class, NULL, OK_FIX_OBJ, &Object);
   if (OK_IS_ERROR(e)) { /* error handling */ }

   /* prepare the string "Computer Science" in Value */
   e = okCreateArray(Env, &DeptName,
      sizeof(DEPT_NAME), DEPT_NAME);
   if (OK_IS_ERROR(e)) { /* error handling */ }
   Value.BufSize = Value.Indicator = sizeof(okArray);
   Value.BufPtr = (okByte *)&DeptName;

   /* a silly sequence just to illustrate usage ... */
   e = okSetAttrVal(Env, Object, AttrRef, &Value);
   if (OK_IS_ERROR(e)) { /* error handling */ }
   e = okGetAttrVal(Env, Object, AttrRef, FALSE, &Value);
   if (OK_IS_ERROR(e)) { /* error handling */ }
   e = okSetAttrValAt(Env, Object, 0, &Value);
   if (OK_IS_ERROR(e)) { /* error handling */ }
   e = okGetAttrValAt(Env, Object, 0, FALSE, &Value);
   if (OK_IS_ERROR(e)) { /* error handling */ }
   e = okGetAttrVals (Env, Object, Obj, 2, AttrRefs, AttrVals);
   e = okGetAttrVals At (Env, Object, 2, AttrPoss, AttrVals);
...
```

メソッドの起動

メソッド起動関数 okExecMeth によって、クラスのメソッドを起動します。

メソッド起動関数の互換性リファレンス

次の表に、okExecMeth のプラットフォーム互換性を示します。

関数	用途	Win32	Palm	EPOC
okExecMeth	メソッドを実行します。	X	X	X

okExecMeth

メソッドを実行します。

構文

```
okError okExecMeth (okEnv env, okHandle dbH, okHandle meth,
   okHandle obj,  okSize argValCount, okAttrVal_s  *argVals,
   okU1B  *outFlag, okAttrVal_s  *result );
```

パラメータ

Env
環境ハンドル

dbH
データベースへのハンドル

meth
実行するメソッド・オブジェクト

Obj
実行するメソッドを含んでいるオブジェクトへのハンドル

argValCount
メソッドに渡される引数の数

argVals
メソッドに渡される引数の配列

outFlag
パラメータの IN/OUT を示すフラグ

result
メソッドの結果

コメント

この関数は、メソッド meth を実行します。メソッドがクラス・メソッドの場合、obj は NULL にできます。そうでない場合は、obj は有効なオブジェクト・ハンドルです。ArgValCount は argVals 配列に提供されている引数値の数です。argVals は okAttrVal_s 構造体の配列で、それぞれに 1 つの引数値が含まれています。OutFlag は、ArgValCount バイトの配列です。*n* 番目のバイトが 0 でないときは、*n* 番目の引数が IN/OUT 引数であることを意味しています。メソッドが IN/OUT 引数を使用しない場合は、outFlag を NULL に設定します。IN/OUT 引数に対応する Java の型は、可変データ型にしてください。カーネルは、その結果を result が指す okAttrVal_s 構造体に返します。

メソッド起動のサンプル・プログラム

次のコードは、メソッド起動プロシージャの使用方法を示したものです。

```
#include "okbasic.h"
#include "okerr.h"
#include "okapi.h"

#define  DEPT_NAME "Computer Science"

...

   okAttrVal_s argVals[2], res;
   /* declare other variables here */
   /* find the void HireStaff(int ID, String Name) method */
   e = okFindMethod(Env, Database, DeptRef, OK_INSTANCE_METHOD,
      9, "HireStaff", 22,
      "(ILjava/lang/String;)V", &MethRef);
   if (OK_IS_ERROR(e)) { /* error handling */ }
   /* use iterator to get a department object into variable dept */
   /* initialize argVal2[0] and argVals[1] here */
   e = okExecMeth(Env, Database, MethRef, dept, 2, argVals,NULL,&res);

...
```

イテレータ（問合せ）

カーネルは、イテレータを使用してデータベース・オブジェクトに対する問合せをサポートします。イテレータは、リレーショナル・データベース・システムの検索と似ています。問合せの有効範囲は通常、オブジェクト・グループ内のクラスのすべてのインスタンスにおよびます。オプションとして、問合せの有効範囲を拡張して、すべてのサブクラスのインスタンスを含むようにできます。イテレータは、指定された検索条件を満たすインスタンスのみを返します。各イテレータに関連付けられたカーソルは、反復内のオブジェクトの現在のオブジェクトを指します。

問合せはイテレータが作成されたときに始まり、イテレータが削除されたときに終了します。イテレータの存続中は、条件を満たすオブジェクトが 1 つずつデータベースから取り出されます。反復内の現在のオブジェクトによってカーソルの位置が確立され、その位置をアプリケーションによって保存または変更できます。それぞれの反復ごとに、アプリケーションで次のいずれかをカーソルの動作プロパティとして設定できます。

- OK_FIRST: 最初のオブジェクトにカーソルを移動します。
- OK_LAST: 最後のオブジェクトにカーソルを移動します。
- OK_NEXT: 次のオブジェクトにカーソルを移動します。
- OK_PRIOR: 直前のオブジェクトにカーソルを移動します。

検索条件

カーネルは、<Connective, Attribute, Operator, Value> 形式の検索条件に基づく問合せをサポートします。

- Connective は連結演算子で、論理 AND 演算子または論理 OR 演算子のいずれかです。
- Attribute は属性で、その値が比較されます。
- Operator は比較演算子か（NULL 値をテストするための）無効述語の 1 つです。
- Value は定数値で、Attribute と比較されます。

データ構造体 okSearchCond_s を使用して、検索条件を表します。

```
typedef struct okSearchCond okSearchCond_s;
struct okSearchCond {
    okU4B       AttrPos;        /* 属性の相対的な位置 */
    okU1B       Operator;       /* 比較する演算子 */
    okU1B       Connective;     /* 次の条件との結合方法 */
    okU1B       Collate;        /* オプションの照合順番 */
    okU1B       _Padding;       /* 使用しません */
    okAttrVal_s AttrVal;        /* 比較する値 */
};
```

検索条件渡し

問合せを発行するために、okSearchCond_s の動的配列にある検索条件をカーネルに渡します。検索条件の評価時は、論理 AND 演算子が論理 OR 演算子より優先されます。つまり、AND 演算子と OR 演算子の両方が現れる場合、カーネルは検索条件をそれぞれ論理積（AND）項からなる論理和（OR）式のセットとして解釈します。

イテレータ関数の互換性リファレンス

この表は、OKAPI のイテレータ関数のリストです。

関数	用途	Win32	Palm	EPOC
okCreateIterator	あるクラスのインスタンスに対する問合せを準備し、一時イテレータ・オブジェクトを返します。	C	C	C
okDeleteIterator	イテレータに対応した問合せを終了します。	C	C	C
okIterate	イテレータに関連付けられたカーソルを前進させて、カーソルが置かれたオブジェクトの本体を指すポインタを返します。	C	C	C
okGetCursor	イテレータの現在のカーソル位置を返します。	C	C	C
okSetCursor	イテレータの現在のカーソル位置を設定します。	C	C	C

okCreateIterator

あるクラスのインスタンスに対する問合せを準備し、一時イテレータ・オブジェクトを返します。

構文

```
okError okCreateIterator(okEnv Env, okHandle Group,
   okHandle Class, okBool isImmediate, okU4B CondCount,
   const okSearchCond_s SearchConds[], Reserved,
   okU4B UpdAttrCount, const okU4B UpdAttrPos[],
   okIterator *RetIter);
```

パラメータ

Env
環境ハンドル

Group
グループ・オブジェクト

Class
クラス・オブジェクト

isImmediate
ブール・フラグで、問合せにサブクラスを含める場合は TRUE、それ以外の場合は FALSE

CondCount
検索条件の数

SearchConds
`okSearchCond_s` 型の検索条件の配列

Reserved
予約済パラメータ。値は必要ありません。

UpdAttrCount
更新する属性の数

UpdAttrPos
条件を満たすインスタンス中の、コール側が更新しようとしている属性の位置の配列

RetIter
イテレータを返すためのバッファ

コメント

この関数は、オブジェクト・グループ `Group` 内の、クラス `Class` のインスタンスに対する問合せを準備します。一時イテレータ・オブジェクトを、`RetIter` に返します。`UpdAttrPos` によって、カーネルは更新によって探索順序が影響される可能性があるアクセス・パスを選択せずにすみます。

okDeleteIterator

イテレータに対応した問合せを終了します。

構文

```
okError okDeleteIterator(okEnv Env, okIterator Iter);
```

パラメータ

Env
環境ハンドル

Iter
イテレータ

コメント

この関数は、イテレータ・オブジェクトなど、イテレータに割り当てられたすべてのシステム資源を解放します。

okIterate

イテレータに関連付けられたカーソルを前進させて、カーソルが置かれたオブジェクトの本体を指すポインタを返します。

構文

```
okError okIterate(okEnv Env, okIterator Iter,
   okU4B Action, okObjData *RetObj);
```

パラメータ

Env
環境ハンドル

Iter
イテレータ

Action
カーソルの移動方向を指定します。

- OK_FIRST: 最初のオブジェクトにカーソルを移動します。
- OK_LAST: 最後のオブジェクトにカーソルを移動します。

- OK_NEXT: 次のオブジェクトにカーソルを移動します。
- OK_PRIOR: 直前のオブジェクトにカーソルを移動します。

RetObj
オブジェクト本体を指すポインタを返すためのバッファ

コメント

この関数は、イテレータに関連付けられたカーソルを前進させて、カーソルが置かれたオブジェクトの本体を指すポインタを（RetObj に）返します。カーネルは関数から返されたオブジェクトを固定し、カーソルが移動されると自動的に固定解除します。okIterate は、次の場合に RetObj に NULL を返します。

- イテレータが探索終了位置に達しているときに OK_NEXT が指定された場合
- イテレータが探索開始位置にあるときに OK_PRIOR が指定された場合

どちらの場合も、エラー・コードは返されません。

okGetCursor

イテレータの現在のカーソル位置を返します。

構文

```
okError okGetCursor(okEnv Env, okIterator Iter,
   okCursor *RetCursor);
```

パラメータ

Env
環境ハンドル

Iter
イテレータ

RetCursor
カーソルを返すためのバッファ

コメント

この関数は、イテレータの現在のカーソル位置を RetCursor に返します。okGetCursor から返されるカーソルは、一時オブジェクトです。返されたカーソルを使用して、Iter のカーソル位置を変更できます。

okSetCursor をコールした後で、オブジェクトへの参照を取得するために okIterate をコールする必要があります。たとえば、次のようになります。

```
okSetCursor (env, iterHandle, &RetCursor);
okIterate (env, iterHandle, OK_PRIOR, &foundRef);
okIterate (env, iterHandle, OK_NEXT, &foundRef);
```

okSetCursor

イテレータの現在のカーソル位置を変更します。

構文

```
okError okSetCursor(okEnv Env, okIterator Iter,
   okCursor Cursor);
```

パラメータ

Env
環境ハンドル

Iter
イテレータ

Cursor
新しいカーソル位置

コメント

この関数は、イテレータの現在のカーソル位置を、okGetCursor によって取得した Cursor が示す位置に変更します。okDeleteIterator を使用してイテレータ・オブジェクトを削除すると、イテレータに関連付けられたすべてのカーソルが自動的に削除されます。okSetCursor をコールした後で、オブジェクトへの参照を取得するために okIterate をコールする必要があります。

イテレータのサンプル・プログラム

次のコードは、イテレータ・プロシージャの使用方法を示したものです。

```
#include "okbasic.h"
#include "okerr.h"
#include "okapi.h"

okEnv Env;
okHandle Database;
okRef  Group, Class;
okIterator  Itor;
okCursor  Cursor;
Professor_S *Advisor;
okError  e;
okU4B  MinID = 101;
okU2B  MaxAge = 50;

...

   okSearchCond_s Cond[2] = { /* ID >= 101 && Age < 50 */
      {0, OK_GE, OK_AND, 0, 0,{sizeof(okU4B),
        (unsigned char *)&MinID,sizeof(okU4B)}},
      {2, OK_LT, OK_AND, 0, 0,{sizeof(okU2B),
        (unsigned char *)&MaxAge,sizeof(okU2B)}}
   };

   e = okFindGroup(Env, Database, sizeof("Faculty"),
      "Faculty", &Group);
   if (OK_IS_ERROR(e)) { /* error handling */ }
   e = okFindClass(Env, Database, sizeof("Professor_s"),
      "Professor_s", &Class);
   if (OK_IS_ERROR(e)) { /* error handling */ }

   e = okCreateIterator(Env, Group, Class, TRUE, 2, Cond,
      0, 0, NULL, &Itor);
   if (OK_IS_ERROR(e)) { /* error handling */ }

   e = okIterate(Env, Itor, OK_NEXT, (okObjData *)&Advisor);
   if (OK_IS_ERROR(e)) { /* error handling */ }

   e = okGetCursor(Env, Itor, &Cursor);
   if (OK_IS_ERROR(e)) { /* error handling */ }
   e = okSetCursor(Env, Itor, Cursor);
   if (OK_IS_ERROR(e)) { /* error handling */ }
```

```
   e = okDeleteIterator(Env, Itor);
   if (OK_IS_ERROR(e)) { /* error handling */ }

...
```

索引

問合せ処理を高速化するために、カーネルはオブジェクト・グループ内のクラスのインスタンスに対する索引をサポートします。カーネルの問合せオプティマイザは、イテレータが作成されたときに（問合せが発行されたときに）自動的に効率的なアクセス・パスを選択します。カーネルは、オブジェクトの更新時に索引をメンテナンスします。カーネルは、現在 B ツリー索引のみをサポートしています。

既存の索引に関する情報は、okNextIndex 関数を使用して取得できます。okNextIndex は、次のデータ構造で情報を返します。

```
typedef struct {
   okU2B IdxType;
   okU2B IndexNo;
   okU2B Options;
   okArray AttrPos; /* 必要に応じて作成またはサイズ変更されます */
   /* 内部的に使用されます */
   ok2B PrevIdx;
   okU4B ClassNo;
   okPtr Conx;
} okIdxInfo_s;
```

索引関数の互換性リファレンス

この表は、OKAPI の索引管理関数のリストです。

関数	用途	Win32	Palm	EPOC
okCreateIndex	一連の属性の索引を作成します。	C	C	C
okDeleteIndex	索引を削除します。	C	C	C
okGetIndexInfo	索引情報を返します。	C	N	C
okNextIndex	指定されたグループおよびクラスに対する次の索引を返します。	C	N	C

okCreateIndex

一連の属性の索引を作成します。

構文

```
okError okCreateIndex(okEnv Env, okHandle Group,
   okHandle Class, okU4B NumAttrs, okU4B AttrPos[],
   okU2B IndexType, okU2B Options, okU2B *RetIndexNo);
```

パラメータ

Env
環境ハンドル

Group
グループ・オブジェクト

Class
クラス・オブジェクト

NumAttrs
索引が作成される属性の数

AttrPos
索引が作成される属性の位置。0 から始まります。

IndexType
作成される索引の種類。現在、このパラメータの値としては、B ツリー索引を示す OK_BTREE のみがサポートされています。

Options
索引を作成するための追加オプション

RetIndexNo
新しい索引の数を返すバッファ

コメント

この関数は、AttrPos に示される一連の属性に索引を作成します。オブジェクト・グループ Group 内の、クラス Class のインスタンスに対する索引を作成します。Options は、作成する索引ごとにキー制約を指定します。有効なキー制約には、次のようなものがあります。

- OK_UNIQUE_KEY: キー値は一意である必要がありますが、NULL 値は指定できます。
- OK_PRIMARY_KEY: キー値は一意である必要があり、NULL 値は指定できません。

一度索引を作成すると、オブジェクトが作成、更新または削除されるたびに、カーネルが自動的に索引をメンテナンスします。指定された索引がすでに存在している場合、okCreateIndex は何もしません。エラー・コードは返されません。

okDeleteIndex

索引を削除します。

構文

```
okError okDeleteIndex(okEnv Env, okHandle Group,
   okHandle Class, okU4B IndexNo, okU2B IndexType);
```

パラメータ

Env
環境ハンドル

Group
グループ・オブジェクト

Class
クラス・オブジェクト

IndexNo
索引番号または属性位置

IndexType
削除される索引の種類。現在、このパラメータの値としては、OK_BTREE（B ツリー索引を示す）のみがサポートされています。

コメント

この関数は、指定した索引を削除します。引数 Group、Class および IndexNo によって、削除する索引を識別します。その索引がマルチ属性の索引である場合は、okCreateIndex から返された索引番号を IndexNo に指定する必要があります。その索引がシングル属性の索引である場合は、属性の位置を IndexNo に指定できます。指定された索引が存在しない場合、okDeleteIndex は何もしません。エラー・コードは返されません。

okGetIndexInfo

索引に関する情報を返します。

構文

```
okError okGetIndexInfo(okEnv Env, okHandle Group,
   okHandle Class, okU4B AttrPos, okU2B IndexType,
   okU2B *RetInfo);
```

パラメータ

Env
環境ハンドル

Group
グループ・オブジェクト

Class
クラス・オブジェクト

AttrPos
索引番号または属性位置

IndexType
情報を返す索引の種類。現在、このパラメータの値としては、OK_BTREE（B ツリー索引を示す）のみがサポートされています。

RetInfo
索引情報を返すためのバッファ

コメント

この関数は、グループ Group 内の Class インスタンスについて、属性位置 AttrPos の索引に関する情報を RetInfo に返します。

RefInfo に返されたビット・マップは、次のビット・フィールドを使用します。

- OK_INDEX_EXISTS: 索引が存在することを示します。索引が存在しない場合は 0 を返します。
- OK_UNIQUE_KEY: キー値は一意である必要がありますが、NULL 値は指定できます。
- OK_PRIMARY_KEY: キー値は一意である必要があり、NULL 値は指定できません。

例として、4 つの属性に対する索引を取得する場合は、次のようになります。

```
attrpos[]={0,1,2,3}
```

属性 0 に対して okGetIndexInfo をコールして、索引情報を取得できます。その他の属性に関して okGetIndexInfo をコールすると、値 0 が返されます。

okNextIndex

指定されたグループおよびクラスに対する次の索引を返します。

構文

```
okError okNextIndex(okEnv Env, okHandle Group,
   okHandle Class, okIdxInfo_s *Info);
```

パラメータ

Env
環境ハンドル

Group
グループ・オブジェクト

Class
クラス・オブジェクト

Info
索引番号または属性位置

コメント

この関数は、指定されたグループおよびクラスの次の索引を Info に返します。最初のコールでは、適切なグループを Group 引数に設定し、Info.AttrPos 配列を NULL に設定します。それ以降のコールでは、グループを NULL に設定します。すべての索引に関する情報を取り出した後、Info.AttrPos 配列を削除してください。okError は、索引が存在する場合には TRUE、存在しない場合には FALSE を返します。

索引のサンプル・プログラム

次のコードは、索引プロシージャの使用方法を示したものです。

```
#include "okbasic.h"
#include "okerr.h"
#include "okapi.h"

okRef  Group, Class;
okU2B  IndexType;
okError  e;
```

```
...

   e = okFindGroup(Env, Database, sizeof("Faculty"),
      "Faculty", &Group);
   if (OK_IS_ERROR(e)) { /* error handling */ }
   e = okFindClass(Env, Database, sizeof("Professor_s"),  "Professor_s",
      &Class);
   if (OK_IS_ERROR(e)) { /* error handling */ }

   /* create a B-Tree type of index on attribute ID */
   e = okCreateIndex(Env, Group, Class, 0, OK_BTREE,  OK_UNIQUE_KEY);
   if (OK_IS_ERROR(e)) { /* error handling */ }

   e = okGetIndexInfo(Env, Group, Class, 0, &IndexType);
   if (!(RetInfo & OK_INDEX_EXISTS) { /* error handling */ }

   e = okDeleteIndex(Env, Group, Class, 0, OK_BTREE);
   if (OK_IS_ERROR(e)) { /* error handling */ }

...
```

オブジェクトのネーミング

オブジェクトのネーミングにより、オブジェクトに対する記号名を登録できるようになります。オブジェクトを登録すると、その名前で効率的に検索できます。すべてのオブジェクトに対して、そのオブジェクトのクラスにかかわらず名前を割り当てることができます。各データベースは階層化された独自のネームスペースを持っており、これによって名前の管理が一層簡単になっています。

UNIX のファイル・システムのように「ディレクトリ・オブジェクト」がデータベースの名前階層を実装します。ルート・ディレクトリ・オブジェクトが、名前階層の最上位のオブジェクトを記述します。階層内の各中間ノードは、サブツリーの次のレベルのオブジェクト名とオブジェクト参照を含むディレクトリ・オブジェクトです。階層のリーフ・ノードは名前付きオブジェクトです。新しいディレクトリ・オブジェクトを作成して階層内で名前を割り当てると、ネームスペースにサブ階層が作成されます。

オブジェクト名は、ディレクトリ・オブジェクトと関連しているオブジェクトを参照する文字列です。オブジェクト名を形成するときは、ルートからターゲット・オブジェクトまでのすべてのディレクトリ・オブジェクトのローカル名を、そのオブジェクトのローカル名と連結させます。オブジェクト名の要素は円記号（¥）で区切られます。システムによって作成されるルート・ディレクトリ・オブジェクトは、円記号（¥）で表されます。

オブジェクトのネーミングの基礎となるメカニズムは UNIX ファイル・システムの「ソフト・リンク」に似ています。オブジェクトのネーミングには、次のルールが適用されます。

- 1 つのオブジェクトがいくつかの名前（別名）を持つことができます。
- オブジェクト名を削除してもオブジェクトを削除したことにはなりません。
- オブジェクトを削除してもすべてのオブジェクト名が削除されることを意味しません。オブジェクトの削除後に名前ディレクトリに参照先がない参照が残される可能性があります。

オブジェクトのネーミング関数の互換性リファレンス

この表は、オブジェクトの名前を管理する OKAPI 関数のリストです。

関数	用途	Win32	Palm	EPOC
okCreateName	オブジェクトの名前を作成します。	C	C	C
okDeleteName	オブジェクトの名前を削除します。	C	C	C
okFindObj	オブジェクトの名前を検索します。	C	C	C

okCreateName

オブジェクトの名前を作成します。

構文

```
okError okCreateName(okEnv Env, okHandle Directory,
   okSize ObjNameLen, okName ObjNamePtr, okHandle Obj);
```

パラメータ

Env
環境ハンドル

Directory
ディレクトリ・オブジェクト

ObjNameLen
オブジェクト名の長さ

ObjNamePtr
オブジェクトの名前

Obj
オブジェクト

コメント

この関数は、ディレクトリ・オブジェクト Directory と関連するオブジェクト Obj の名前を作成します。ObjNamePtr は、NULL 終了文字列にしてください。空白文字列は許されません。データベースのルート・ディレクトリ・オブジェクトを指定するかわりに、データベース・オブジェクトを指定できます。すべてのオブジェクトのネーミング関数の Directory パラメータに、データベース・オブジェクトとルート・ディレクトリ・オブジェクトのどちらも同じように使用できます。

1 レベルの命名システムを使用している場合は、Directory パラメータの値としてデータベース参照のみ使用できます。

マルチレベルの命名システムを使用している場合は、各レベルにディレクトリ・オブジェクトを作成する必要があります。ディレクトリ・オブジェクトを作成するには、obj パラメータの値に NULL を使用し、Directory パラメータの値にディレクトリ・オブジェクトを使用します。

okDeleteName

オブジェクトの名前を削除します。

構文

```
okError okDeleteName(okEnv Env, okHandle Directory,
   okSize ObjNameLen, okName ObjNamePtr);
```

パラメータ

Env
環境ハンドル

Directory
ディレクトリ・オブジェクト

ObjNameLen
オブジェクト名の長さ

ObjNamePtr
オブジェクトの名前

コメント

この関数は、ディレクトリ・オブジェクト Directory に関連するオブジェクト名 ObjNamePtr を、データベースの名前階層から削除します。データベースのルート・ディレクトリ・オブジェクトを指定するかわりに、データベース・オブジェクトを指定できます。すべてのオブジェクトのネーミング関数の Directory パラメータに、データベース・オブジェクトとルート・ディレクトリ・オブジェクトのどちらも同じように使用できます。

okFindObj

オブジェクトの名前を検索します。

構文

```
okError okFindObj(okEnv Env, okHandle Directory,
   okSize ObjNameLen, okName ObjNamePtr,
   okRef *RetObj, okBool *isDirectory);
```

パラメータ

Env
環境ハンドル

Directory
ディレクトリ・オブジェクト

ObjNameLen
オブジェクト名の長さ

ObjNamePtr
オブジェクトの名前

RetObj
オブジェクト参照を返すためのバッファ

isDirectory
見つかったオブジェクトがディレクトリ・オブジェクトであれば TRUE を返すブール・フラグのバッファ

コメント

この関数は、Directory と関連する、ObjNamePtr で指定される名前のオブジェクトを検索します。オブジェクトが見つかった場合、そのオブジェクトに対する参照を RetObj に返します。オブジェクトが見つからなかった場合、NULL 参照を RetObj に返します。これはエラーとは見なされません。見つかったオブジェクトがディレクトリ・オブジェクトかどうかを示すブール値を、isDirectory に返します。データベースのルート・ディレクトリ・オブジェクトを指定するかわりに、データベース・オブジェクトを指定できます。すべてのオブジェクトのネーミング関数の Directory パラメータに、データベース・オブジェクトとルート・ディレクトリ・オブジェクトのどちらも同じように使用できます。

オブジェクトのネーミングのサンプル・プログラム

次のコードは、オブジェクトのネーミング・プロシージャの使用方法を示したものです。

```
#include "okbasic.h"
#include "okerr.h"
#include "okapi.h"

okRef  Group, Class;
Professor_s *Chairman;
okError  e;

...

   /* create a directory called "EECS" */
   e = okCreateName(Env, Database /* to use root directory */,  sizeof("EECS"),
       "EECS", NULL);
   if (OK_IS_ERROR(e)) { /* error handling */ }

   /* create a new professor object in Chairman */
   e = okFindGroup(Env, Database,
      sizeof("Faculty"), "Faculty", &Group);
   if (OK_IS_ERROR(e)) { /* error handling */ }
   e = okFindClass(Env, Database,
      sizeof("Professor_s"), "Professor_s", &Class);
   if (OK_IS_ERROR(e)) { /* error handling */ }
   e = okCreateObj(Env, Group, Class,
      NULL, OK_FIX_OBJ, (okObjData *)&Chairman);
   if (OK_IS_ERROR(e)) { /* error handling */ }

   /* assign "¥EECS¥CHAIRMAN" as name of a new "Professor_s" object */
   e = okCreateName(Env, Database,
      sizeof("¥¥EECS¥¥CHAIRMAN"), "¥¥EECS¥¥CHAIRMAN", Chairman);
   if (OK_IS_ERROR(e)) { /* error handling */ }
```

```
   e = okFindObj(Env, Database,
      sizeof("¥¥EECS¥¥CHAIRMAN"), "¥¥EECS¥¥CHAIRMAN", &Chairman, NULL);
   if (OK_IS_ERROR(e)) { /* error handling */ }

   e = okDeleteName(Env, Database,
      sizeof("¥¥EECS¥¥CHAIRMAN"), "¥¥EECS¥¥CHAIRMAN");
   if (OK_IS_ERROR(e)) { /* error handling */ }

...
```

関係

OKAPI は、異なるクラスまたは同じクラスのオブジェクト間の関係をサポートします。概念的に 2 つのオブジェクト間の関係は、それぞれ逆方向の相互従属オブジェクト参照のペアで成り立ちます。たとえば、A と B との 2 つのオブジェクト間の関係は、A から B へのオブジェクト参照と、それに対応する B から A へのオブジェクト参照で表されます。

OKAPI は、次のような関係をサポートしています。

関係	例
参照整合性	オブジェクト A がオブジェクト B を参照している場合に B が削除されると、A から B への参照は無効になります。A 内のある参照がオブジェクト C への参照に変更された場合は、B と C の中の参照も更新する必要があります。
カスケード削除	オブジェクト A が削除されると、B も削除されます。B が C を参照していてそれがカスケード削除関係の場合は、C もまた削除されます。
制限付き削除	B が存在する間は、A を削除できません。
単独所有	B は、B と単独所有関係にある 1 つのオブジェクトからのみ参照できます。

デフォルトの関係は、参照整合性です。ただし、オブジェクト参照は必ずしも関係の一部である必要はない、ということに注意してください。

2 つのクラス間の関係は各クラスから 1 つずつ、合計 2 つの属性で表されます。関係は、<FromClass, FromAttribute, ToClass, ToAttribute> の形式で表されます。

関係関数の互換性リファレンス

この表は、関係を管理する OKAPI 関数のリストです。

関数	用途	Win32	Palm	EPOC
okCreateRel	2 つのクラスの間の関係を確立します。	C	N	C
okDeleteRel	関係を削除します。	C	N	C

okCreateRel

2 つのクラスの間の関係を確立します。

構文

```
okError okCreateRel(okEnv Env, okHandle FromClass,
   okHandle FromAttr, okHandle ToClass,
   okHandle ToAttr, okU4B Options);
```

パラメータ

Env
環境ハンドル

FromClass
関係に関与する 1 番目のクラス

FromAttr
関係の 1 番目のクラスの属性

ToClass
関係に関与する 2 番目のクラス

ToAttr
関係の 2 番目のクラスの属性

Options
関係の種類。次のいずれかの値が指定できます。

- OK_REF_INTEGRITY: 参照整合性の場合
- OK_CASCADE_DEL: カスケード削除の場合

- OK_RESTRICT_DEL: 制限付き削除の場合
- OK_ONE_PARENT: 唯一の親の場合

コメント

この関数は、クラス FromClass と ToClass 間の関係を確立します。この関係は、FromClass の属性 FromAttr と ToClass の属性 ToAttr によって維持されます。カーネルが関係を適切に維持できるのは、すべてのオブジェクト更新が okCreateObj、okDeleteObj、okSetAttrVal および okSetAttrValAt の各関数によって実行された場合のみであることに注意してください。特に、関係に関与する属性を直接変更すると、関係にとって必須の制約に違反する可能性があります。この関数は、クラス属性に対してのみ機能します。

2 つのクラスの間に関係を作成し、それらのクラスからインスタンスとしてオブジェクトを生成した後、各オブジェクトのクラス属性を関係オブジェクトへの参照によって更新する必要があります。それには、okSetAttrVal または okSetAttrValAt を使用します。関係オブジェクトのクラス属性は、自動的に更新されます。

okDeleteRel

関係を削除します。

構文

```
okError okDeleteRel(okEnv Env, okHandle FromClass,
   okHandle FromAttr, okHandle ToClass, okHandle ToAttr);
```

パラメータ

Env
環境ハンドル

FromClass
関係に関与する 1 番目のクラス

FromAttr
関係の 1 番目のクラスの属性

ToClass
関係に関与する 2 番目のクラス

ToAttr
関係の 2 番目のクラスの属性

コメント

この関数は、先に確立された FromClass の属性 FromAttr と ToClass の属性 ToAttr によって維持されている関係を削除します。

関係のサンプル・プログラム

次のコードは、関係プロシージャの使用方法を示したものです。

```
#include "okbasic.h"
#include "okerr.h"
#include "okapi.h"

okRef  DeptClass, StaffAttr, ProfClass, DeptAttr;
okError  e;

...

   e = okFindClass(Env, Database, sizeof("Dept_s"), "Dept_s",  &DeptClass);
   if (OK_IS_ERROR(e)) { /* error handling */ }
   e = okFindAttr(Env, DeptClass, sizeof("Staff"), "Staff",
      NULL, &StaffAttr, NULL);
   if (OK_IS_ERROR(e)) { /* error handling */ }
   e = okFindClass(Env, Database, sizeof("Professor_s"),
      "Professor_s", &ProfClass);
   if (OK_IS_ERROR(e)) { /* error handling */ }
   e = okFindAttr(Env, ProfClass, sizeof("Dept"), "Dept",
      NULL, &DeptAttr, NULL);
   if (OK_IS_ERROR(e)) { /* error handling */ }

   e = okCreateRel(Env, DeptClass, StaffAttr, ProfClass, DeptAttr,
      OK_REF_INTEGRITY);
   if (OK_IS_ERROR(e)) { /* error handling */ }

   e = okDeleteRel(Env, DeptClass, StaffAttr, ProfClass,  DeptAttr);
   if (OK_IS_ERROR(e)) { /* error handling */ }

...
```

バイナリ・ラージ・オブジェクト（BLOB）

バイナリ・ラージ・オブジェクト（Binary Large Object: BLOB）は、構造化されていないバイナリ・データを格納します。BLOB は、キャラクタ・セットとして意味を持たないビットストリームと考えることができます。BLOB には、最大 2GB のバイナリ・データを格納できます。デジタル画像などの巨大なロー・データを格納するためのオブジェクトです。

一般的には、BLOB は他のカーネル・オブジェクトと同様の動作をします。すべての BLOB オブジェクトはクラス okLong に属します。ただし、BLOB はアプリケーションのアドレス空間に格納できない大きさの場合があるため、OKAPI には BLOB の一部を取り出したり更新するための関数があります。さらに、OKAPI には BLOB オブジェクトのサイズを問い合せたり変更するための関数もあります。BLOB のサイズが問題になるため、カーネルは BLOB をキャッシュに固定することはできません。

BLOB 関数の互換性リファレンス

この表は、BLOB を管理する OKAPI 関数のリストです。

関数	用途	Win32	Palm	EPOC
okCreateBlob	BLOB を作成します。	C	C	C
okDeleteBlob	BLOB を削除します。	C	C	C
okGetBlobVal	指定されたバイト・オフセットから始まる BLOB の一部を取り出します。	C	C	C
okSetBlobVal	BLOB の一部分を変更します。	C	C	C
okSetBlobSize	BLOB のサイズを調整します。	C	C	C
okGetBlobSize	BLOB のサイズを返します。	C	C	C

okCreateBlob

BLOB を作成します。

構文

```
okError okCreateBlob(okEnv Env, okHandle Group,
   okU4B Size, okU4B EstimatedSize, okRef *RetBlobObj);
```

パラメータ

Env
環境ハンドル

Group
グループ・オブジェクト

Size
バイト数で表した BLOB データのサイズ

EstimatedSize
バイト数で表したラージ・オブジェクト（Large Object: LOB）の予測サイズ

RetBlobObj
BLOB オブジェクトへの参照を返すバッファ

コメント

この関数は、オブジェクト・グループ `Group` に BLOB を新規作成し、その BLOB への参照を `RetBlobObj` に返します。BLOB を作成すると、実際に利用可能なディスク上の空き領域に応じて、そのオブジェクトが最低 `EstimatedSize` バイトまで拡張できることをカーネルが保証します。Palm プラットフォームでは、BLOB の最大サイズは 64KB です。引数 `Size` には、BLOB オブジェクトの初期のサイズを指定します。`EstimatedSize` に指定された値にかかわらず、すべての LOB は少なくとも 2MB まで拡張できます。LOB の最大サイズは 2GB です。

okDeleteBlob

BLOB を削除します。

構文

```
okError okDeleteBlob(okEnv Env, okHandle BlobObj);
```

パラメータ

Env
環境ハンドル

BlobObj
削除する BLOB オブジェクトへのハンドル

コメント

この関数は、BLOB オブジェクトを削除し、BLOB オブジェクトによって使用されたすべての空間を解放します。

okGetBlobVal

指定されたバイト・オフセットから始まる BLOB の一部を取り出します。

構文

```
okError okGetBlobVal(okEnv Env, okHandle BlobObj,
   okU4B Offset, okU4B Size, okAttrVal_s *RetVal);
```

パラメータ

Env
環境ハンドル

BlobObj
読み取る BLOB オブジェクトへのハンドル

Offset
BLOB オブジェクトへのバイト・オフセット

Size
バイト数で表した BLOB データのサイズ

RetVal
バイト数で表した BLOB データを返すためのバッファ

コメント

この関数は、指定されたバイト・オフセットから始まる `BlobObj` の一部を取り出します。取り出した値とその長さを、`RetVal` に指定したバッファとインジケータに返します。

okSetBlobVal

BLOB の一部分を変更します。

構文

```
okError okSetBlobVal(okEnv Env, okHandle BlobObj,
   okU4B Offset, okU4B Size, okAttrVal_s *NewVal);
```

パラメータ

Env
環境ハンドル

BlobObj
変更する BLOB オブジェクトへのハンドル

Offset
BLOB オブジェクトへのバイト・オフセット

Size
バイト数で表した BLOB データのサイズ

NewVal
新しいデータ値を入れるバッファ

コメント

この関数は、BLOB の一部分を変更します。新しい値と長さは、`NewVal` に指定されます。`okSetBlobVal` をコールする前に BLOB のサイズを調整して、変更の事前調整をする必要があります。たとえば、LOB の終わりに新しいデータを追加するには、最初に `okSetBlobSize` をコールして、LOB のサイズを新しいデータが格納できるよう調整する必要があります。

okSetBlobSize

BLOB のサイズを調整します。

構文

```
okError okSetBlobSize(okEnv Env, okHandle BlobObj,
   okU4B Size);
```

パラメータ

Env
環境ハンドル

BlobObj
変更する BLOB オブジェクトへのハンドル

Size
バイト数で表した BLOB データの新しいサイズ

コメント

この関数は、BLOB のサイズを調整します。BlobObj の現在のサイズよりも大きい値を Size に指定することで、LOB を拡張できます。同様に、BlobObj の現在のサイズよりも小さい値を Size に指定することで、LOB を切り捨てることができます。

okGetBlobSize

BLOB のサイズを返します。

構文

```
okError okGetBlobSize(okEnv Env, okHandle BlobObj,
   okU4B *RetSize);
```

パラメータ

Env
環境ハンドル

BlobObj
BLOB オブジェクトへのハンドル

RetSize
バイト数で表した BLOB のサイズ

コメント

この関数は、BlobObj の現在のサイズをバイト単位で RetSize に返します。

BLOB のサンプル・プログラム

次のコードは、BLOB を管理するプロシージャの使用方法を示したものです。

```
#include "okbasic.h"
#include "okerr.h"
#include "okapi.h"

#define  BUF_SIZE 1024
#define  PICTURE_SIZE ((okU4B)10*BUF_SIZE)

okAttrVal_s BlobVal;
okByte Buffer[BUF_SIZE];
okRef Group, Blob;
okU4B BlobSize;
okError e;
```

```
...

   /* create a blob in a group */
   e = okCreateBlob(Env, Group, PICTURE_SIZE /* a hint */, &Blob);
   if (OK_IS_ERROR(e)) { /* error handling */ }

   e = okSetBlobSize(Env, Blob, sizeof(Buffer));
   if (OK_IS_ERROR(e)) { /* error handling */ }

   memset((void *)Buffer, 1, sizeof(Buffer));
   memset((void *)&BlobVal, 0, sizeof(BlobVal));
   BlobVal.BufSize = BlobVal.Indicator = BUF_SIZE;
   BlobVal.BufPtr = Buffer;
   e = okSetBlobVal(Env, Blob, 0, &BlobVal);
   if (OK_IS_ERROR(e)) { /* error handling */ }

   e = okGetBlobVal(Env, Blob, 0, BUF_SIZE, &BlobVal);
   if (OK_IS_ERROR(e)) { /* error handling */ }

   e = okGetBlobSize(Env, Blob, BUF_SIZE, &BlobSize);
   if (OK_IS_ERROR(e)) { /* error handling */ }

   e = okDeleteBlob(Env, Blob);
   if (OK_IS_ERROR(e)) { /* error handling */ }

...
```

トランザクション

トランザクションとは、1 つ以上のデータベースに対する一連の操作から形成される不可分の作業単位です。トランザクションでは、すべての操作が正しく行われるか、すべての操作が失敗するかのどちらかです。コミットされたトランザクションの結果は、たとえシステムがクラッシュしても維持されます。カーネルはロギングを使用してトランザクション・プロパティをサポートします。

トランザクション関数の互換性リファレンス

この表は、トランザクションを管理する OKAPI 関数のリストです。

関数	用途	Win32	Palm	EPOC
okTransact	カレント・トランザクションの結果を制御し、オプションで新規トランザクションを開始します。	C	C	C
okSetIsolationLevel	現在の分離レベルを設定します。	C	X	C
okGetIsolationLevel	現在の分離レベルを取得します。	C	X	C

okTransact

カレント・トランザクションの結果を制御し、オプションで新規トランザクションを開始します。

構文

```
okError okTransact(okEnv Env, okU4B Action);
```

パラメータ

Env
環境ハンドル

Action
実行するアクション

コメント

この関数は、カレント・トランザクションの結果を制御し、ビット・マップ Action のフラグの指定によって、オプションで新規トランザクションを開始します。Action には、次のようなフラグを設定できます。

フラグ	説明
OK_BEGIN	新規トランザクションを開始します。
OK_COMMIT_END	カレント・トランザクションをコミットします。
OK_ROLLBACK_END	カレント・トランザクションをロールバックします。
OK_COMMIT_CONTINUE	コミットし、新規トランザクションを同レベルで開始します。
OK_ROLLBACK_CONTINUE	ロールバックし、新規トランザクションを同レベルで開始します。
OK_CHECKPOINT_COMMIT	カレント・トランザクションをコミットしますが、すべてのロックを保持します。

データベース内のオブジェクトに影響するほとんどの OKAPI 関数では、すでにアクティブになっているトランザクションがない場合は、トランザクションが自動的に開始されます。

okSetIsolationLevel

現在の分離レベルを設定します。

構文

```
okError okSetIsolationLevel (okEnv Env, okU2B Level);
```

パラメータ

Env
環境ハンドル

Level
分離レベル。次のいずれかの値が指定できます。

- OK_TXN_SINGLE_USER: シングル・ユーザーの場合
- OK_TXN_READ_COMMITTED: 読取りがコミットされる場合
- OK_TXN_REPEATABLE_READ: リピータブル・リードの場合
- OK_TXN_SERIALIZABLE: シリアライズ可能の場合

okGetIsolationLevel

現在の分離レベルを取得します。

構文

```
okError okGetIsolationLevel (okEnv Env, okU2B *Level);
```

パラメータ

Env
環境ハンドル

Level
分離レベルを返すためのバッファ

トランザクションのサンプル・プログラム

次のコードは、okTransact プロシージャを示したものです。

```
#include "okbasic.h"
#include "okerr.h"
#include "okapi.h"

okError  e;

...

   e = okTransact(Env, OK_BEGIN);
   if (OK_IS_ERROR(e)) { /* error handling */ }
   /* do something ... */
   e = okTransact(Env, OK_COMMIT_END);
   if (OK_IS_ERROR(e)) { /* error handling */ }

...
```

動的配列

OKAPI の動的配列ライブラリ（okArray）は、ANSI C の記憶領域割当てライブラリを拡張したものです。malloc によって割り当てられるメモリー領域のように、動的配列のメモリーは実行時に割り当てられます。配列の要素に直接アクセスするために、動的配列をポインタ型にキャストできます。動的配列が不要な場合、メモリー・リークを防ぐためにアプリケーションで明示的に解放する必要があります。動的配列ライブラリの関数では（realloc のように）、動的配列のサイズを調節できます。

malloc によって割り当てられる空間とは異なり、動的配列には配列のサイズに関する情報が格納され、コール側から配列サイズを問い合せることができます。動的配列が占める実際の空間（容量）は配列の大きさより大きくなる場合があります。動的配列ライブラリは予測割当てアルゴリズムを使用して、配列を少しずつ拡張することで起こるメモリーの断片化を減らしています。さらに、ライブラリには動的配列を追加およびコピーする関数もあります。

次の表に、ANSI C の記憶領域割当てライブラリと OKAPI 動的配列ライブラリの対応を示します。

ANSI C の関数	OKAPI の動的配列関数
malloc()	okCreateArray()
free()	okDeleteArray()
realloc()	okSetArraySize()
((UserType *)p)[i]	((UserType *)p)[i]

動的配列は、永続オブジェクトに埋め込むことによって永続化できます。埋め込まれた動的配列は、永続オブジェクトにおける可変長属性をサポートする基礎となります。永続オブジェクトの一部となった動的配列では、その記憶域の割当てと解放がカーネルによって自動的に管理されます。たとえば、可変長属性を持つオブジェクトがディスクからフェッチされると、動的配列がカーネルによって自動的に割り当てられて、メモリー内に可変長属性の値を持つようになります。同様に、属性を持つオブジェクトがスワップ・アウトまたは削除されると、可変長属性によって占められていた空間が自動的に解放されます。

動的配列関数の互換性リファレンス

この表は、動的配列を管理する OKAPI 関数のリストです。

関数	用途	Win32	Palm	EPOC
okCreateArray	動的配列を作成します。	C	C	C
okDeleteArray	動的配列を削除します。	C	C	C
okGetArraySize	動的配列のサイズを返します。	C	C	C
okSetArraySize	動的配列のサイズを設定します。	C	C	C
okAppendArray	動的配列の最後にデータを追加します。	C	C	C
okCopyArray	動的配列のコピーを作成します。	C	C	C
okCreateMDArray	新規マルチディメンション配列を作成します。	C	C	C
okCopyMDArray	マルチディメンション配列をコピーします。	C	C	C
okFreeMDArray	マルチディメンション配列を解放します。	C	C	C

okCreateArray

新しい動的配列を作成します。

構文

```
okError okCreateArray (okEnv Env, okArray *RetArray,
    okU4B Size, const okByte *Data);
```

パラメータ

Env
環境ハンドル

RetArray
新しい動的配列を指すポインタを返すバッファ

Size
動的配列のデータのサイズ

Data
動的配列のデータ値

コメント

このプロシージャは、配列を作成し、新しい配列を指すポインタを RetArray に返します。Data が NULL の場合は、動的配列に初期値が設定されません。この関数は、コールされる前には RetArray が有効な動的配列を指していないと仮定しています。メモリー・リークを防ぐために、okCreateArray に RetArray を渡す前に、コール側で RetArray が指している有効な動的配列をすべて解放しておく必要があります。RetArray が NULL の場合は、エラーになります。

一般に、配列に対してなんらかの操作を実行する場合は、その前に okCreateArray をコールしてその配列を初期化する必要があります。または、次の例のように、直接 NULL に設定して初期化することができます。

```
okArray anArray = NULL;
```

okDeleteArray

動的配列に占有されていたメモリーを解放します。

構文

```
okError okDeleteArray(okEnv Env, okArray *Array);
```

パラメータ

Env
環境ハンドル

Array
動的配列を指すポインタ

コメント

このプロシージャは、Array が指す動的配列に占有されていたメモリーを解放します。Array が指す値には、NULL が設定されます。コールされる前に Array が NULL であった場合、okDeleteArray はなにも実行しません。

okGetArraySize

動的配列のサイズを、バイト単位で取り出します。

構文

```
okError okGetArraySize(okEnv Env, const okArray *Array,
   okU4B *RetSize);
```

パラメータ

Env
環境ハンドル

Array
動的配列を指すポインタ

RetSize
動的配列のサイズを返すバッファ

コメント

このプロシージャは、Array が指す動的配列のサイズを、バイト単位で取り出します。Array が NULL であるか、Array が NULL ポインタを指している場合、0 を返します。

okSetArraySize

動的配列のサイズを、バイト単位で設定します。

構文

```
okError okSetArraySize(okEnv Env, okArray *Array,
   okU4B NewSize);
```

パラメータ

Env
環境ハンドル

Array
動的配列を指すポインタ

NewSize
動的配列の新しいサイズ

コメント

このプロシージャは、Array が指す動的配列のサイズを、バイト単位で調整します。Array が NULL ポインタを指していて、NewSize が 0 より大きい場合、okSetArraySize は新しい配列を作成します。NewSize が動的配列の元のサイズよりも大きいか等しい場合は、動的配列の元の値を保持します。新規に追加されたメモリーは、初期化されません。NewSize が動的配列の元のサイズよりも小さい場合は、NewSize に従って配列の値が切り捨てられます。動的配列に追加されるバイト数がわかっているときに、この関数を利用できます。okAppendArray を何度もコールして動的配列にデータ項目を追加するかわりに、1

回でサイズを調整し、動的配列を正規の C 配列として扱いながらデータ項目を配列内へ直接に割り当てることができます。

okAppendArray

動的配列の最後にデータを追加します。

構文

```
okError okAppendArray(okEnv Env, okArray *Array,
   okU4B Size, const okByte *Data);
```

パラメータ

Env
環境ハンドル

Array
動的配列を指すポインタ

Size
動的配列のデータのサイズ

Data
動的配列のデータ値

コメント

このプロシージャは、Array が指す動的配列の最後に、Size バイト数の（Data が指す）新規データを追加します。コール時に Array が NULL ポインタを指していて、Size が 0 より大きい場合、okAppendArray は新しい配列を作成します。Data が NULL の場合、新しく追加された領域は初期化されません。Array が NULL の場合は、エラーになります。

okCopyArray

動的配列のコピーを作成します。

構文

```
okError okCopyArray(okEnv NewEnv, okArray *RetNewArray,
   okEnv OldEnv, const okArray *OldArray);
```

パラメータ

NewEnv
ターゲット配列の環境ハンドル

RetNewArray
ターゲット配列

OldEnv
ソース配列の環境ハンドル

OldArray
ソース配列

コメント

この関数は、OldArray が指す動的配列のコピーを作成し、作成したコピーを指すポインタを RetNewArray に返します。OldArray が指す動的配列は、解放されません。メモリー・リークを防ぐために、この関数にポインタを渡す前に、RetNewArray が指している有効な動的配列をすべて解放する必要があります。OldArray が NULL であるか、NULL ポインタを指している場合、RetNewArray には NULL ポインタが返されます。RetNewArray が NULL の場合は、エラーになります。

okCreateMDArray

新規マルチディメンション配列を作成します。

構文

```
okError okCreateMDArray (okEnv Env, okSize Dim, const okSize *Size, okArray *d);
```

パラメータ

Env
環境ハンドル

Dim
ディメンション。要素の数です。

Size
各要素のデータ・サイズ（バイト数）

d
配列ハンドル

okCopyMDArray

マルチディメンション配列をコピーします。

構文

```
okError okCopyMDArray (okEnv Env, okSize Dim, okArray s, okArray *d);
```

パラメータ

Env
環境ハンドル

Dim
ディメンション。要素の数です。

s
ソース配列

d
ターゲット配列

okFreeMDArray

マルチディメンション配列を解放します。

構文

```
okError okFreeMDArray (okEnv Env, okSize Dim, okArray a);
```

パラメータ

Env
環境ハンドル

Dim
ディメンション。要素の数です。

a
解放する配列へのハンドル

動的配列のサンプル・プログラム

次のコードは、動的配列プロシージャの使用方法を示したものです。

```
#include "okbasic.h"
#include "okerr.h"
#include "okapi.h"

okChar *CityName = "San Jose";
okU4B CityPopulation = 300000;
okU4B ArraySize;      /* size of a dynamic array */
okArray City;         /* dynamic array of okChar */
okArray Population;   /* dynamic array of okU4B */
okArray NewArray;     /* a new array */
okError e;
okEnv   Env;

...

   /* see okInit to initialize the environment here */
   e = okCreateArray(Env, &City,
      strlen(CityName)+1, (okByte *)CityName);
   if (OK_IS_ERROR(e)) { /* error handling */ }
   e = okCreateArray(Env, &Population,
      sizeof(okU4B), (okByte *)&CityPopulation);
   if (OK_IS_ERROR(e)) { /* error handling */ }

   e = okGetArraySize(Env, &City, &ArraySize);
   if (OK_IS_ERROR(e)) { /* error handling */ }
   if (ArraySize != sizeof("San Jose")) { /* error handling */ }

   e = okSetArraySize(Env, &Population, 2*sizeof(okU4B));
   if (OK_IS_ERROR(e)) { /* error handling */ }

   CityPopulation = 350000;
   e = okAppendArray(Env, &Population,
      sizeof(okU4B), (okByte *)CityPopulation);
   if (OK_IS_ERROR(e)) { /* error handling */ }

   e = okCopyArray(Env, &NewArray, Env, &City);
   if (OK_IS_ERROR(e)) { /* error handling */ }

   e = okDeleteArray(Env, &City);
   if (OK_IS_ERROR(e)) { /* error handling */ }

...
```

3

言語間の相互運用性

説明する内容は、次のとおりです。

- ODBC と OKAPI の相互運用性
- データ型の対応
- 相互運用性のサンプル・プログラム

ODBC と OKAPI の相互運用性

Oracle Lite の重要な機能の 1 つに、異なる言語インタフェース間の相互運用性を扱う能力があります。つまり、Oracle Lite によって、1 つのプログラム内で SQL とオブジェクト・データベースの違いを意識しないデータの相互運用が可能になります。

データ・モデリングの点からみると、高度に統合されたこの相互運用性は「オブジェクト参照」という簡単な概念によって達成されています。Oracle Lite OKAPI インタフェースは、オブジェクト参照を `okRef` として定義します。Oracle Lite SQL には、SQL-3 向けに提案されたオブジェクト拡張機能との互換性を備えた、新しい SQL 型 `POL_REF` を取り込んでいます。

Oracle Lite のオブジェクト参照の概念により、アプリケーションにおける SQL とオブジェクト・データベース間のデータの自由な交換が可能になります。たとえば、アプリケーションで SQL 表を作成してデータを移入した後、C または C++ の構造体としてそれらの列の内容に直接アクセスできます。逆に、Oracle Lite オブジェクト・データベース内にある C または C++ の永続オブジェクトに対して、アプリケーションから問合せを発行できます。

Oracle Lite には、アプリケーションが同一データをリレーショナル・インタフェースとオブジェクト・インタフェースの両方から操作できるような、単純かつ実用的なメカニズムが提供されています。Oracle Lite では、SQL データベースの行とオブジェクト・データベース内の永続インスタンスの間に境界はありません。この柔軟性によって、アプリケーションの各タスクに最適のインタフェースを使用できます。

データ型の対応

Oracle Lite オブジェクト・カーネルは、オブジェクト・モデルとリレーショナル・モデルのデータ型参照の間で、データ型を変換します。次の表に、SQL、Java、標準 C および OKAPI のデータ型の対応を示します。

SQL データ型	Java データ型	OKAPI データ型	C データ型
char	char	okChar（ディメンションを 1 に設定）	char
char(len)	char[]	okChar（ディメンションを文字配列の長さに設定）	char[]
varchar(len)	char[]	okChar（ディメンションを 0 に設定）	char[]
すべての long 型	POLBlob	okChar（ディメンションを 0 に設定）	char[]
decimal/numeric	java.math.BigDecimal	okFloat、okDouble または okU1B	char[]、int、float または double

SQL データ型	Java データ型	OKAPI データ型	C データ型
bit	boolean	okChar（ディメンションを 1 に設定）	boolean
tinyint	byte	okChar（ディメンションを 1 に設定）	char
smallint	short	ok2B	short
integer	int	ok2B または ok4B	int
bigint	long	ok4B	long
real	float	okFloat	float
float/double	double	okDouble	double
binary/binary(len)/ varbinary	byte[]	okChar（ディメンションを 0 に設定）	unsigned char[]
long varbinary	POLBlob	okChar（ディメンションを 0 に設定）	unsigned char[]
date	java.sql.Date	okChar（ディメンションを 0 に設定）	unsigned char[]
time	java.sql.Time	okChar（ディメンションを 0 に設定）	unsigned char[]
timestamp	java.sql.Timestamp	okChar（ディメンションを 0 に設定）	unsigned char[]

相互運用性のサンプル・プログラム

この項では、ODBC と OKAPI の相互運用性の例を示します。具体的には、SQL 行を C 構造体の中にマップすることで、C プログラムから SQL 行に直接アクセスし、変更を加える方法を示します。

このプログラムでは ODBC 文の設定とクリーン・アップの他に、ODBC へのコールを介して SQL 表を作成し、データを移入します。SQL の SELECT 文を実行してオブジェクト参照を取り出します。次に、SELECT 文の結果をすべてフェッチし終わるまでループします。各反復処理の中で、ODBC のフェッチ文では、その前の ODBC コールによって ODBC 文に結合されたホスト変数 `Ref` にオブジェクト参照を返します。行へのオブジェクト参照を使用して、C 構造体として行に直接アクセスし、その行の ID 列を表示します。

各 ODBC 接続は内部的にカーネル・セッションにマップされます。Oracle Lite では、`SQL_OOT_SESSION` が指定されている場合には ODBC 接続に対応するカーネル・セッション・ハンドルが返されるように、ODBC ルーチン `SQLGetInfo` を拡張しています。`SQLGetInfo`

からセッション・ハンドルが返されると、それを使用して OKAPI とのデータのやりとりができます。

```
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <windows.h>
#include <sql.h>
#include <sqlext.h>
#include "okapi.h"
#include "interop.h"

#define DROP_PERSON "DROP TABLE PERSON"
#define CREATE_PERSON ¥
   "CREATE TABLE PERSON(ID INTEGER, AGE INTEGER)"
#define INSERT_PERSON "INSERT INTO PERSON VALUES(1, 25)"
#define SELECT_PERSON_REF "SELECT POL_REF FROM PERSON"

typedef struct Person_s { /* must match the SQL table */
  ok4B ID;
  ok4B AGE;
} Person_s;

void GetRef()
{
  HENVH env = NULL; /* environment handle */
  HDBCH dbc = NULL; /* connection handle */
  HSTMTH stmt = NULL; /* statement handle */
  RETCODE rc = 0; /* return code from ODBC functions */
  okRef Ref = NULL; /* object reference */
  okEnv Session = NULL; /* session handle */
  Person_s *aPerson; /* pointer to a person row */
  okError e = OK_NOERROR; /* return code from OKAPI functions */
  FILE *OutFile;

  OutFile = fopen("example.out", "w");
  if (OutFile == NULL) { /* error handling */ }

  /* prepare for an ODBC statement */
  rc = SQLAllocEnv(&Henv); /* allocate an environment */
  if (rc != SQL_SUCCESS) { /* error handling */ }
  rc = SQLAllocConnect(Henv, &Hdbc); /* allocate a connection */
  if (rc != SQL_SUCCESS) { /* error handling */ }
  rc = SQLConnect(Hdbc, "POL", SQL_NTS, "SYSTEM", SQL_NTS,
     "SYSTEM", SQL_NTS); /* connect to the POL data source */
  if (rc != SQL_SUCCESS) { /* error handling */ }
  rc = SQLAllocStmt(Hdbc, &Hstmt); /* allocate a statement */
```

```
  if (rc != SQL_SUCCESS) { /* error handling */ }

  /* create PERSON table and populate it */
  rc = SQLExecDirect(Hstmt, DROP_PERSON, SQL_NTS);
  if (rc != SQL_SUCCESS) { /* error handling */ }
  rc = SQLExecDirect(Hstmt, CREATE_PERSON, SQL_NTS);
  if (rc != SQL_SUCCESS) { /* error handling */ }
  rc = SQLExecDirect(Hstmt, INSERT_PERSON, SQL_NTS);
  if (rc != SQL_SUCCESS) { /* error handling */ }

  /* select POL_REF (object reference) & bind result to Ref */
  rc = SQLExecDirect(Hstmt, SELECT_PERSON_REF, SQL_NTS);
  if (rc != SQL_SUCCESS) { /* error handling */ }
  rc = SQLBindCol(Hstmt, 1, SQL_C_DEFAULT, &Ref, 0, NULL);
  if (rc != SQL_SUCCESS) { /* error handling */ }

  /* get the session handle associated with the connection */
  rc = SQLGetInfo(Hdbc, SQL_OOT_SESSION,
  &Session, sizeof(okEnv), NULL);
  if (rc != SQL_SUCCESS) { /* error handling */ }

  /* fetch the results until no data is found */
   for (; ;) {
      rc = SQLFetch(Hstmt);
      if (rc == SQL_NO_DATA_FOUND) break;
      if (rc != SQL_SUCCESS) { /* error handling */ }
      e = okFixObj(Session, Ref, OK_SLOCK,
      (okObjData *)&aPerson); /* locate the object */
      if (OK_IS_ERROR(e)) { /* error handling */ }
      fprintf(OutFile, "PERSON ID: %d¥n", aPerson->ID);
      e = okUnfixObj(Session, Ref, FALSE); /* release object */
      if (OK_IS_ERROR(e)) { /* error handling */ }
   }

  /* clean up resources used by ODBC */
  rc = SQLFreeStmt(Hstmt, SQL_DROP); /* release statement */
  if (rc != SQL_SUCCESS) { /* error handling */ }
  rc = SQLDisconnect(Hdbc); /* close connection */
  if (rc != SQL_SUCCESS) { /* error handling */ }
  rc = SQLFreeConnect(Hdbc);/* release connection */
  if (rc != SQL_SUCCESS) { /* error handling */ }
  rc = SQLFreeEnv(Henv); /* release environment */
  if (rc != SQL_SUCCESS) { /* error handling */ }
}

int PASCAL WinMain (HANDLE hInstance, HANDLE hPrevInstance,
    LPSTR lpszCmdParam, int nCmdShow)
```

```
{
  GetRef();
  return 0;
}
```

4

Stateless Object Database API

この章では、Oracle Lite Stateless Object Database API について説明します。説明する内容は次のとおりです。

概要

Stateless Object Database API は、使用しやすくエラーが発生しにくい C++ インタフェースを提供します。このインタフェースの設計目標は、次のとおりです。

- 可能なかぎりメモリー管理を自動化し（メモリー・リークを避けるため）、ユーザーにロー・データではなくオブジェクトを提供します（メモリー破損とデータ型の誤った使用を避けるため）。Stateless Object Database API オブジェクトは、使用しないリソースを解放するために参照カウントを使用します。
- 基礎となるデータベースの機能をすべて公開するかわりに、Stateless Object Database API では最も一般的に使用される操作を定義し、専門タスクの一部は従来どおりネイティブ・インタフェースによる処理を前提としています。
- パフォーマンスや機能を損なわずに Stateless Object Database API の最上位に実装する操作が、標準ではありません。最大のパフォーマンスと最小のフットプリントで、低レベルのインタフェースを定義することが目的です。

Stateless Object Database API を使用すると、一時変数を定義せずに単一の式で各操作を実行できます。次にオブジェクトを作成するためのコード例を示します。

```
DBClass c;
DBObject o = cls.create(DBSetList() << "id" << 26000 << "name" << "Jane Doe");
```

現在、Palm OS および Windows 32 用の C++ Stateless Object Database API ライブラリと、Windows 32 OKAPI および独自データベース・フォーマットを使用する単一ユーザー用 pure Java 実装に追加する Java ライブラリがあります。

サンプル・プログラム

```
void helloSODA() {
    try {
DBSession sess("HelloSODA"); // Connect to the database, creating it if necessary
// Create a new class
DBClass cls = sess.createClass("People", DBAttrList() <<
DBAttr("id", DB_INT) << DBAttr("name", DB_STRING));
// Create several objects. We can use identify columns by name or positions
DBObject o = cls.create(DBSetList() << "id" << 10 << "name" << "Alice");
cls.create(DBSetList() << 0 << 20 << 1 << "Bob");
// Note the automatic type conversion
cls.create(DBSetList() << "id" << "314"<< 1 << 3.14159265358);
// Execute a query (will return two objects)
DBCursor c = cls.createQuery(DBColumn("id") == 10 ||
DBColumn("name") == "Bob").execute();
DBObject ob;
while ((ob = c.next()) != DBNULL) {
DBString s = ob["name"];
FrmCustomAlert(1000, "Query", s, NULL);
```

```
}
// Delete an object
o.remove();
// Clean up so that create class is successful next time
sess.rollback();
    } catch(DBException e) {
DBString s = e.getMessage();
FrmCustomAlert(1000, "Error", s, NULL);
    }
}
```

このプログラムでは、Stateless Object Database API を使用して、Palm OS に常駐する Oracle Lite データベースから列の「id」と「name」を取り出します。

次の項では、Stateless Object Database API クラスの概要を簡単に説明します。サポートされる個々のメソッドについては説明しません。インタフェースの完全な定義は、**soda.h** ファイルを参照してください。

Stateless Object Database API クラス

この項では、Stateless Object Database API クラスを説明します。

DBString

これは `const char *` を含む、自動メモリー管理を備えたラッパーです。Stateless Object Database API により、文字列を返したり、受け入れるために使用されます。このクラスを使用すると、文字データを使用している間 `DBString` への参照を保持し、必要なときに `const char *` へ明示的に変換します。次に例を示します。

```
DBException e=...;

printf("%s¥n", e.getMessage());
Do this instead:
DBException e=...;

DBString s = e.getMessage();

printf("%s¥n", (const char *)s);
```

DBData

これは、Stateless Object Database API によりサポートされるすべてのデータ型を含むラッパーです。この機能により、様々な型を受け入れたり返すことができるメソッドのコール規則を簡略化します。DBData には、基本的な型を引数とするコンストラクタと、それを元の基本的な型に戻すキャスト演算子があります。Stateless Object Database API プログラムではプリミティブ型のみを扱い、C++ で自動変換することがあります。

DBData は配列で表現できます。この場合、[] 演算子を使用してコンポーネントを取得および設定できます。getSize() は、現在の配列長を取得し、setSize() は配列を拡大縮小するために使用できます。

DBData インスタンスの作成時の長さは 0 で、特定の型はありません。最初の集合演算で、オペランドと同じ型に設定されます。それ以降、DBData は新しい値を同じ型に変換します。型は DBData(DBType type) コンストラクタを使用して、即時に設定できます。

「==」または「!=」を使用して、DBData が DB_NULL と比較して初期化されているかどうかをチェックできます。この機能は、SQL スタイルの NULL 値の有無のテストに使用します。

DBData の簡単な使用例を次に示します。

```
DBData d;
d[0] = "Hello";

d[1] = 5; // Cast to string

d[3] = d[2] = d[1];

// d.getLength() == 4
```

DBData、DBArrayElement および DBObjectColumn の間の相互作用を調べて、C++ 演算子の上書き方法を理解してください。

DBObject

DBObject のインスタンスはデータベース表の行を表します。DBObject の動作は、obj["列名"] または obj[0 基準の列位置] を使用して索引付け（および変更）できる DBData の配列と似ています。DBClass の create() メソッドと同じ引数の set() メソッドを使用して、一度に複数の列を変更できます。オブジェクトは remove() コールを使用して表から削除でき、新規オブジェクトは DBClass.create() を介して作成されます。

DBClass

DBClass のインスタンスはデータベース表を表します。Stateless Object Database API プログラムは、一般に DBClass を使用して表に新規オブジェクトを作成します。これは、2 つの create メソッドのどちらかを使用して実行できます。最初のメソッドでは、列のリストと値のリストを別々に使用します。

```
DBObject obj = cls.create(DBColList() << "c1" << 1, DBDataList() << 10 << "Hello");
```

2 つ目の形式では、列とそれに対応する値が交互に使用されます。

```
DBObject obj = cls.create(DBSetList() << "c1" << 10 << 1 << "Hello");
```

クラスは、remove メソッドを使用して削除できます。

DBQuery、DBQueryTree、DBCursor

Stateless Object Database API は、問合せの作成に 3 段階モデルでサポートします。最初に、問合せの条件を記述するために DBQueryTree が作成されます。このツリーには、後に値が提給される「プレースホルダ」値を使用した条件を含めることができます。

次に、DBQuery オブジェクトを作成するために DBClass.createQuery() メソッドがコールされます。この段階で、適合した索引の検索と、類似した問合せの最適化が実行されます。最後に DBQuery.execute がコールされ、オプションとして実データのリストでプレースホルダを置き換えます。この結果、DBCursor のインスタンスが返され、first、last、prior および next メソッドを使用して該当オブジェクトが順次アクセスされます。返された DBObject は、DB_NULL と比較して結果リストの終わりに達しているかどうかをチェックできます。

演算子の多重定義を使用して、問合せツリーを通常の C++ 論理式として構成できます。次に例を示します。

```
DBQueryTree qt = (DBColumn("c1") < 5 || DBColumn(3) == "Hello") && DBColumn("name")
== "John";
```

プレースホルダが必要な場合は、値のかわりに DBMarker(位置番号) で置き換えます。同一のプレースホルダはツリー内の複数の箇所で使用できます。たとえば、次のようになります。

```
DBQueryTree qt = DBColumn(DBColumn("firstName") == DBMarker(0) ||
DBColumn("lastName") == DBMarker(0) || DBColumn("id") == DBMarker(1) ||
alwaysReturn == true;
```

このツリーで指定された問合せを次のように実行できます。

```
DBQuery q = cls.createQuery(qt);

DBCursor c = q.execute(DBDataList() << "Jane" << 1000);
```

DBSession

DBSession は、データベース接続を表します。commit() および rollback() メソッドは、期待どおりの機能を実行します。findClass メソッドは、既存のクラスの参照に使用できます。createClass は新規クラスを作成します。スキーマは、DBAttr オブジェクトのリストにより指定する必要があります。DBAttr のコンストラクタは、次のとおりです。

```
DBAttr(const DBString &name, DBType type, int dim=0, int prec=0, int scale=0);
```

型は、データベース内のオブジェクトの格納タイプを指定する次の値のいずれかになります。

DB_BYTE、DB_BOOL、DB_SHORT、DB_INT、DB_LONG、DB_FLOAT、DB_DOUBLE、DB_OBJECT、DB_BLOB、DB_STRING、DB_BINARY、DB_NUMBER、DB_DATE

dim は、配列のディメンション数を指定します。0 はスカラー、1 はディメンションが 1 の配列です。現時点で Palm がサポートしているのはスカラーの DB_STRING と DB_BINARY で、他の型の場合はスカラーまたは 1D の配列のみです。

DBException

完了できない操作はすべてロールバックされ、DBException が発生します。getErrorCode() メソッドを使用して（実装固有の）エラー・コードを取得でき、getMessage() はメッセージを返します。これらの判断を助けるメソッドが 2 つあります。getType() では次の値のいずれかが返されます。

値	説明
CONSTRAINT_TYPE	主キーなどに対する制約違反。
SCHEMA_TYPE	プログラムおよびデータベース・スキーマ間の同期が取れていない（たとえば、存在しない列を参照している場合）ために発生したエラーです。
SYSTEM_TYPE	アプリケーション・エラーではなく、システムまたは Stateless Object Database API のエラーです（たとえば、メモリー不足）。
TRANSACTION_TYPE	同時トランザクションの存在（たとえば、デッドロック）によるエラーです。
UNKNOWN_TYPE	他のどのカテゴリにも入らないエラーです。
UNSUPPORTED_TYPE	特定の Stateless Object Database API の実装でサポートされていない機能です。
USAGE_TYPE	プログラムが Stateless Object Database API を不正に使用しています（たとえば、マーカーのない問合せに対する値の指定）。

getAction() では、エラーに対して提案される応答が返されます。

アクション	説明
EXIT_ACTION	接続のクローズなどの通常のクリーンアップを実行した後にプログラムを終了します（または、少なくとも Stateless Object Database API の使用を停止します）。USAGE_TYPE および SCHEMA_TYPE のエラーでこのアクションが返されます。コール側のプログラムに問題があるか、データベースと一致しないことが原因です。
PANIC_ACTION	エラーが重大なため、クリーンアップを完了できません。即時終了、たとえば abort() などが最善の方法です。
ROLLBACK_ACTION	エラーを起こしたトランザクションをロールバックする必要があります。
UNKNOWN_ACTION	状況に応じたエラー処理が必要です。未知のエラーの場合、EXIT_ACTION と同様に処理します。
VALUE_ACTION	失敗したコールを別のデータ値で再試行します。

一般に、プログラムは、発生が予想されるエラーに対する型を使用するように作成され、予想外のエラーに対処します。

索引

B

O

い

お

か

く

け

さ

す

そ

て

と

は

ふ

め