DB2® ウェアハウス・マネージャー

情報カタログ・マネージャー プログラミングの手引きおよび解説書

バージョン 7

SC88-8549-00
(英文原典：SC26-9997-00)

DB2® ウェアハウス・マネージャー

情報カタログ・マネージャー プログラミングの手引きおよび解説書

バージョン 7

SC88-8549-00
(英文原典：SC26-9997-00)

ご注意!

本書、および本書がサポートする製品をご使用になる前に、 385ページの『特記事項』にある一般的な情報を必ずお読みください。

本書において、日本では発表されていない IBM 製品 (機械およびプログラム)、プログラミング、またはサービスについて言及または説明する場合があります。しかし、このことは、弊社がこのような IBM 製品、プログラミング、またはサービスを、日本で発表する意図があることを必ずしも示すものではありません。

本マニュアルに関するご意見やご感想は、次の URL からお送りください。今後の参考にさせていただきます。

http://www.ibm.com/jp/manuals/main/mail.html

なお、日本 IBM 発行のマニュアルはインターネット経由でもご購入いただけます。詳しくは

http://www.ibm.com/jp/manuals/ の「ご注文について」をご覧ください。

(URL は、変更になる場合があります)

原 典 :	SC26-9997-00 IBM® DB2® Warehouse Manager Information Catalog Manager Programming Guide and Reference Version 7
発 行 :	日本アイ・ビー・エム株式会社
担 当 :	ナショナル・ランゲージ・サポート

第1刷 2000.6

この文書では、平成明朝体™W3、平成明朝体™W9、平成角ゴシック体™W3、平成角ゴシック体™W5、および平成角ゴシック体™W7を使用しています。この(書体*)は、(財) 日本規格協会と使用契約を締結し使用しているものです。フォントとして無断複製することは禁止されています。

注* 平成明朝体™W3、平成明朝体™W9、平成角ゴシック体™W3、
平成角ゴシック体™W5、平成角ゴシック体™W7

目次

本書について vii
情報カタログの概要 vii
情報カタログ体系のインターフェース . . . vii
コメントの送付先 viii

第1章 情報カタログ・マネージャーの概要 . . 1
情報カタログ・マネージャーを使用する人 . . 1
ユーザー 2
管理者 2
アプリケーション・プログラマー 2
情報カタログ・マネージャーを使用するアプリケーション. 2
情報アプリケーション 3
情報カタログ・メタデータの保守と管理を行うツール 4

第2章 アプリケーションによるオブジェクトの管理 5
区分によるオブジェクトの編成 5
プログラマーにとっての情報カタログ・マネージャー・オブジェクト・タイプ 7
オブジェクト・タイプの定義. 8
登録特性の指定 9
新規のオブジェクト・タイプの区分の指定 11
必須オブジェクト・タイプ特性の定義 . . 11
新規のオブジェクト・タイプおよびオブジェクト・インスタンスの識別 14
情報カタログ・マネージャーの識別名 . . . 14

第3章 情報カタログ・マネージャー API 呼び出しを使用するプログラムの作成. 17
情報カタログ・マネージャー API 呼び出しによって行えること 17
情報カタログ・マネージャー・アプリケーション・サポートの提供. 17
オブジェクト・タイプ登録の管理 18
オブジェクト・タイプの管理 18
オブジェクト・インスタンスの管理 . . . 19
情報カタログ・マネージャー識別子の管理 19
オブジェクトの関係の定義 19
オブジェクト・インスタンスの探索 . . . 20
オブジェクトのタイプおよびインスタンスのリスト 20
情報カタログ・マネージャーとの間でのメタデータ・オブジェクトのコピー 21
外部プログラムの開始. 22
情報カタログ・マネージャー・データベースに対して行われた変更の確認または除去 . 22
企業情報カタログの管理 23
情報カタログ・マネージャー API 呼び出しの出し方 23
情報カタログ・マネージャー API 呼び出しとの間でのデータの受け渡し 24
パラメーターによる単一の入力値およびポインターの受け渡し 24
入力構造および出力構造による複数の値の受け渡し 24
ヘッダー・ファイルの組み込み. 26
C 言語プログラムの作成の概要 26
C 言語ソース・コードの作成 27
環境の設定 27
アプリケーションのコンパイルと連係 . . 27
プログラムで情報カタログ・マネージャー API 呼び出しを使用する方法 28
FLGInit によるプログラムの開始 28
FLGTerm によるプログラムの終了 . . . 28
エラーが発生した場合の情報カタログ・データベースの保護 28
プログラムを開始するための Programs オブジェクトの設定 29
API 呼び出しによるメタデータの作成 . . 30
API 呼び出しによるメタデータの削除 . . 30
情報カタログ・マネージャー・データ・タイプを使用する情報カタログ・メタデータの指定 31
各国語の考慮事項 32
翻訳された必須特性 32
英語以外の言語での値の指定 33
DG2SAMP.C の概要 33

第4章 情報カタログ・マネージャー入力および出力構造 35

情報カタログ・マネージャー API の入出力構造の一般的な特性 35
情報カタログ・マネージャー API 入力構造 36
ヘッダー域 (常に必須) 37
定義域 (常に必須) 39
オブジェクト域 (値を定義するときには必須) 43
API 呼び出しのための入力構造の作成 . . . 44
DG2API.H を使用して長さと値を定義する 44
入力構造全体のサイズを計算する 46
ヘッダー域の定義 48
定義域の定義. 50
オブジェクト域の定義. 52
ヘッダー域、定義域、およびオブジェクト域の定義例 53
情報カタログ・マネージャーの API 出力構造 56
ヘッダー域 (常に存在) 58
定義域 (常に存在) 60
オブジェクト域 (値を検索するときに存在) 63
API 呼び出しによって得られた出力構造の読み取り 64
出力構造を読むためのポインターの使い方 65
DG2API.H を使用して値を読み取る . . . 66
出力構造内の特性の数を計算する 66
戻された値の集合の数を計算する 66
定義域の特性のデータ・タイプと長さを読み取る 67
オブジェクト域から順に値を読み取る . . 68
DG2SAMP.C により出力構造内の値を見つける例 69

第5章 情報カタログ・マネージャー API 呼び出しの構文 75
API 呼び出しの構文規則 75
構文図の読み方 75
DG2API.H で定義された定数をプログラムで使用する方法 76
FLGAppendType 77
FLGCommit 82
FLGConvertID 85
FLGCreateInst. 87
FLGCreateReg 93
FLGCreateType 101
FLGDeleteInst 108
FLGDeleteReg 111
FLGDeleteTree 113
FLGDeleteType 118
FLGDeleteTypeExt 121
FLGExport 124
FLGFoundIn 131
FLGFreeMem 137
FLGGetInst 139
FLGGetReg 143
FLGGetType. 147
FLGImport 150
FLGInit 154
FLGListAnchors. 161
FLGListAssociates 164
FLGListContacts 173
FLGListObjTypes 177
FLGListOrphans. 180
FLGListPrograms 187
FLGManageCommentStatus 190
FLGManageFlags 195
FLGManageIcons 198
FLGManageTagBuf. 201
FLGManageUsers 203
FLGMdisExport 209
FLGMdisImport 212
FLGNavigate. 215
FLGOpen. 219
FLGRelation 221
FLGRollback. 225
FLGSearch 227
FLGSearchAll 237
FLGTerm. 243
FLGTrace. 245
FLGUpdateInst 248
FLGUpdateReg 254
FLGWhereUsed 259
FLGXferTagBuf. 263

付録A. サンプル・プログラム DG2SAMP.C 265
DG2SAMP.C のコンパイル. 265
DG2SAMP.C の連係 266
DG2SAMP.C の実行 266

付録B. 情報カタログ・マネージャー API ヘッダー・ファイル – DG2APIH 267
DG2API.H で定義されている定数 267
DG2API.H における構造およびデータ・タイプの定義. 276

情報カタログ・マネージャー API 呼び出しの関数原型 278

付録C. 情報カタログ・マネージャーの限界値 283

付録D. 情報カタログ・マネージャーの理由コード 285

特記事項 385
商標 388

用語集 391

参考文献 399

索引 401

本書について

本書は、情報カタログ・マネージャーを使用するアプリケーションの作成を計画しているプログラマーを対象として書かれています。これらのプログラムでは、アプリケーション・プログラム・インターフェース (API) 呼び出しを使用して情報カタログ・マネージャーの機能にアクセスすることができます。

本書は、読者が*情報カタログ・マネージャー 管理の手引き* で説明されている概念および C 言語プログラミングを熟知していることを前提としています。また、Microsoft Visual C++ コンパイラーが導入されていることも必要です。

情報カタログ・マネージャーには、情報カタログのためのアプリケーション・プログラム・インターフェース (API) とインポート / エクスポート・インターフェースが備えられています。

情報カタログの概要

情報カタログは、ある組織の情報資源に関する明細記述、すなわちメタデータを保管するためのメカニズムです。ユーザーは情報カタログを使用することにより、自分が使用することのできるデータとその意味を知ることができます。必要なデータが分かると、ユーザーは情報アプリケーションを使用してそのデータの検索と分析を行うことができます。情報カタログ・マネージャーには、ユーザーが情報カタログ・マネージャーとともに Lotus 1-2-3 などの情報アプリケーション機能を使用できるようにする機能が備えられています。

情報カタログ体系のインターフェース

本書には、情報カタログの定義が含まれています。これらのインターフェースには、以下のものがあります。

- アプリケーション・プログラム・インターフェース (API)

  情報カタログ・マネージャー API 呼び出しのための入出力の構文と仕様は、 75ページの『第5章 情報カタログ・マネージャー API 呼び出しの構文』で説明されています。

  これらの API 呼び出しで使用される入出力構造の仕様は、 35ページの『第4章 情報カタログ・マネージャー入力および出力構造』で説明されています。
- 情報カタログのためのインポート / エクスポート・インターフェース

タグ言語の構文と使い方の情報は、*情報カタログ・マネージャー 管理の手引き* に記載されています。

コメントの送付先

弊社に、品質情報のフィードバックをお寄せください。また、本書や他のデータウェアハウスセンター資料に関するコメントもお寄せください。コメントは、Web から送信することができます。 http://www.software.ibm.com/data/vw/ の Web サイトをご利用ください。

この Web サイトにはフィードバック・ページがあり、そこでコメントを入力して送信できます。

第1章 情報カタログ・マネージャーの概要

情報カタログ・マネージャーは、ビジネスの専門家が組織内のデータを迅速に、しかも簡単に探し出せるようにします。ユーザーは実際には情報アプリケーション (データの検索と分析を行うためのアプリケーション) を使用してデータにアクセスするため、データが実際にどこに保管されているのかを知っていたり配慮したりする必要はありません。

情報カタログ・マネージャーを使用することにより、以下のことを知ることができます。
- どのデータが使用可能であるのか
- そのデータがビジネスのうえでどのような意味を持つのか
- そのデータがどこにあるのか
- どのようにしてそのデータにアクセスすることができるのか
- そのデータに関して誰に尋ねることができるのか

データに関するこのような情報は、*記述データ* または*メタデータ* と呼ばれ、*情報カタログ* に保管されます。各情報カタログは、情報カタログ・マネージャーによって管理されるデータベースに保管されます。

各情報源または情報源のグループは、*オブジェクト* として表現されます。ユーザーは、多くの種類のオブジェクトを使用して、データベースの表、スプレッドシート、ディジタル化された写真など、組織で使用される各種の情報源を表現することができます。これらの多くのオブジェクトから、情報源を利用するプログラムを開始することができます。

各情報カタログ・オブジェクトは、カード・カタログにおけるカードに類似しています。それぞれのオブジェクトでは、情報源の名前、説明、情報源が最後に更新された日付などの、情報源に関する詳細が提供されます。

情報カタログ・マネージャーを使用する人

情報カタログ・マネージャーを使用する人は、次の 3 つのタイプに分かれます。
- ユーザー
- 管理者
- アプリケーション・プログラマー

情報カタログ・マネージャーを使用する人

ユーザー

組織内で、ユーザーは情報カタログ・マネージャーを使用して探し出した情報にもとづいて、業務上の意思決定を行ったり、意思決定に関与したりします。ユーザーは各種のソフトウェア・プログラムに習熟していても、データベースまたはコンピューター・プログラミングの概念を理解している必要はありません。

管理者によって権限を得ている情報カタログ・マネージャー・ユーザーは、情報カタログ・マネージャー管理者によって通常実行される追加のオブジェクト管理タスクを実行することができます。

管理者

管理者は、この情報カタログ・マネージャーを管理します。また、ユーザーが必要とするデータを見つけやすくするためにメタデータを用意します。管理者は、情報カタログ・メタデータが使用可能であって、その検出と使用が簡単にでき、内容が現行のものであり、しかも未許可のアクセスから保護されるように配慮します。

アプリケーション・プログラマー

アプリケーション・プログラマーは、情報カタログ・ユーザーをサポートするプログラムを作成します。情報カタログ・マネージャーでは、ユーザーのプログラムで情報カタログ・マネージャー機能を使用するための C 言語による API 呼び出しが提供されます。

アプリケーション・プログラマーには、情報カタログ・マネージャーによってメタデータがどのように編成され、保管されるのかを示す詳細な情報が必要です。情報カタログ・マネージャー・アプリケーションによってオブジェクトがどのように使用されるのかについては、5ページの『第2章 アプリケーションによるオブジェクトの管理』を参照してください。

情報カタログ・マネージャーを使用するアプリケーション

情報カタログ・マネージャー機能を使用するアプリケーションとして、次の 2 種類のアプリケーションを作成することができます。

- ユーザーにデータを提示するアプリケーション。
- 管理者がメタデータの追加や更新などの管理作業を実行するための援助を行う管理ツール (たとえば、抽出プログラム)。

情報アプリケーション

情報カタログ・マネージャーを使用するアプリケーションは、2 つの方法で作成することができます。これらのプログラムでは、次のことが行えます。

- 情報カタログ・マネージャーからアプリケーションを開始します。

  図1 は、ユーザーが必要なオブジェクトを見つけ出して、そのオブジェクトによって識別される情報源を使用するために、 DOS または Microsoft® Windows® で実行される、使いなれた情報アプリケーションを開始するようすを示しています。情報カタログ・マネージャーは必要なメタデータをこのアプリケーションに渡します。

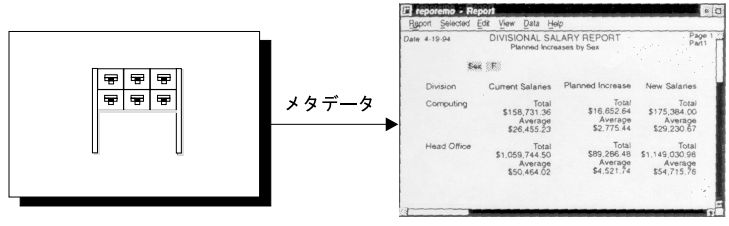


図 1. 情報カタログ・マネージャーからのアプリケーションの開始

- アプリケーションにメタデータを提供します。

  ユーザーは、DOS か Microsoft Windows で実行する使い慣れた情報アプリケーションを使用します。これらのアプリケーションは情報カタログ・マネージャー機能を使用して、ユーザーが利用しようとしている情報源を探し出します。このアプリケーションは情報カタログ・マネージャーにより見つけ出された実データの検索と分析を行い、 図2 に示すように、独自のユーザー・インターフェースを使用してユーザーに結果を提示します。

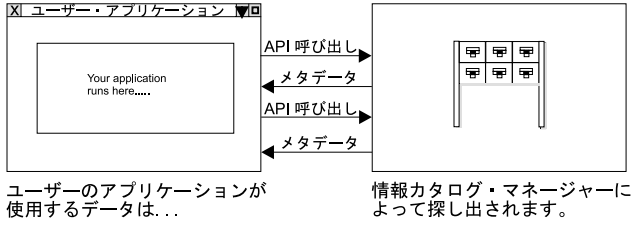


図 2. 情報カタログ・マネージャーでデータを見つけるためのアプリケーションの使用

情報カタログ・メタデータの保守と管理を行うツール

管理者のために、次のことを行うツールを作成することができます。

- 情報カタログ・メタデータの保守
- 情報カタログへのメタデータの追加

情報カタログの保守

管理者の主要な作業の 1 つとして、情報源自体が変更されたときに情報カタログ内のメタデータを更新することがあげられます。たとえば、あるファイルに関するメタデータには、最新の更新が行われた日付が含まれていることがあります。このファイルが再び更新されたときに、管理者はメタデータ内の対応する日付を更新する必要があります。

この処理は、対応する情報源が変更されたときにメタデータを更新するプログラムを書くことにより、自動化することができます。

新規オブジェクトの追加

管理者が新規の情報カタログを作成したり既存の情報カタログに新規の情報源を追加したりしたときには、新規のオブジェクト・タイプおよびオブジェクトを追加する必要があります。管理者は、情報カタログ・マネージャー・*タグ言語* を含むファイルをインポートすることにより、メタデータを追加することができます。このタグ言語は、情報カタログにインポートされるメタデータの意味を定義します。

管理者が指定した情報にもとづいてタグ言語ファイルを自動的に生成するアプリケーションを作成することができます。そして、これらのファイルを 1 つまたは複数の情報カタログにインポートして、メタデータを含む情報カタログを移植することができます。

また、既存のデータ・ソースからメタデータを抽出して、そのデータをタグ言語ファイルとして形式設定するアプリケーションを作成することもできます。このようなアプリケーションは、*抽出プログラム* と呼ばれるもので、*情報カタログ・マネージャー 管理の手引き* で説明されています。

第2章 アプリケーションによるオブジェクトの管理

情報カタログ内のメタデータの管理またはアクセスを行うアプリケーションを作成するときには、保管したメタデータを情報カタログがどのように編成するのかについて、詳しい情報が必要です。この章では、以下の事項を説明します。

- 情報カタログが区分、オブジェクト・タイプ、およびオブジェクト・インスタンスによってどのように編成されるのか
- オブジェクト・タイプ定義の 2 つの部分
- 新規のオブジェクト・タイプの定義方法
- さまざまな情報カタログ・ユーザーが使用可能な用語

区分によるオブジェクトの編成

情報カタログ・マネージャーには、メタデータを分類するための 7 つの区分が用意されています。これらの区分は、情報カタログ・データベース内のメタデータのための構造を用意するためにオブジェクトがどのように共同して作動するのかを制御します。 以下の任意の情報カタログ区分で (ただし、 Program および Attachment 区分を除く)、新しいオブジェクト・タイプを作成できます。

区分	定義
Grouping	他のオブジェクト・タイプを含めることのできるオブジェクト・タイプ。
Elemental	その他の情報カタログ・マネージャー・オブジェクト・タイプの構築ブロックである、 Grouping 以外のオブジェクト・タイプ。
Contact	特定オブジェクトの詳細の参照先を示すオブジェクト・タイプ。詳細には、オブジェクトが表す情報を作成した人や情報を管理する担当部門が含まれている。
Program	情報カタログ・マネージャー・オブジェクト・タイプで表される実際の情報を処理できるアプリケーションを識別し記述する Programs オブジェクト・タイプ。 Program 区分に属するオブジェクト・タイプは、情報カタログを作成するときに定義される Programs オブジェクト・タイプだけです。

Dictionary ユーザーの業務に特有の用語を定義するオブジェクト・タイプ。

Support ユーザーの情報カタログまたは企業についての追加情報を提供するオブジェクト・タイプ。

Attachment 他の情報カタログ・マネージャー・オブジェクトに付加された追加の情報を識別する Comments オブジェクト。 Attachment 区分に属するオブジェクト・タイプは、情報カタログを作成するときに定義される Comments オブジェクト・タイプだけです。

表1 には、情報カタログ・マネージャーのオブジェクト・タイプ区分の間の関係がまとめられています。

表 1. 情報カタログ・マネージャーの区分間の関係

区分	包含関係	リンク	関連する連絡	コメントの付加	プログラムの立ち上げ
Grouping	他の Grouping または Elemental オブジェクトを含む。	他の Grouping または Elemental オブジェクト	Yes	Yes	Yes
Elemental	なんらかの Grouping オブジェクトに含まれる。	他の Grouping または Elemental オブジェクト	Yes	Yes	Yes
Contact	なし	なし	No	Yes	Yes
Program	なし	なし	No	Yes	No
Dictionary	なし	なし	No	Yes	Yes
Support	なし	なし	No	Yes	Yes
Attachment	なし	なし	No	No	Yes

情報カタログ・マネージャーを使用すると、オブジェクト・タイプ およびオブジェクトを定義することにより、情報源に関するデータを編成することができます。

オブジェクトを分類するには、オブジェクト・タイプを使用します。たとえば、複数のデータベース表がある場合、表のオブジェクト・タイプを作成すると、それぞれの表について類似のメタデータを保管および保守できるようになります。ほとんどの区分について、ユーザーの組織にとって最も便利なメタデータが入るようにオブジェクト・タイプを定義することができます。

オブジェクトには、特定の情報単位に関するメタデータ、たとえば、表、人物、またはプログラムに関する情報などが入ります。オブジェクト・タイプはオブジェクトのテンプレートであり、類似の各情報単位について情報カタログに保管する必要のあるメタデータを定義します。したがって、オブジェクトは、オブジェクト・タイプのインスタンス と考えることができます。つまり、1 つのオブジェクト・タイプにもとづいて複数のインスタンスを定義することができます。

さまざまな区分を使用して情報カタログを設計するための詳細については、*情報カタログ・マネージャー 管理の手引き* を参照してください

プログラマーにとっての情報カタログ・マネージャー・オブジェクト・タイプ

ユーザー・インターフェースまたはタグ言語を使用してオブジェクト・タイプを管理する管理者は、オブジェクト・タイプの使用だけしか意識しません。しかし、情報カタログ・マネージャー API 呼び出しを使用してオブジェクトを管理するプログラムを作成するときには、オブジェクト・タイプには、オブジェクト・タイプ登録とオブジェクト・タイプ自体の 2 つの部分があることを考慮する必要があります。

オブジェクト・タイプ登録

オブジェクト・タイプ登録には、以下の事項を含む、オブジェクト・タイプに関する全般的な情報が含まれます。

- オブジェクト・タイプが属している区分
- オブジェクト・タイプの拡張名 (NAME) および省略名 (DPNAME)
- オブジェクト・インスタンス情報が入っている情報カタログ・データベース表の名前

オブジェクト・タイプ登録を作成または更新するときには、そのオブジェクト・タイプに関連するアイコン・ファイルの名前も情報カタログ・マネージャーに提供します。

オブジェクト・タイプ

オブジェクト・タイプは、各オブジェクトに使用される特性を定義します。これらの特性 (OWNER や DESCRIPTION など) には、そのオブジェクトによって記述されている情報源に関する情報が入っています。

上記の 2 つの部分には、別個の保守機能が必要です。これらの機能は、以下の情報カタログ・マネージャー API 呼び出しによって提供されます。

オブジェクト・タイプ登録の場合	オブジェクト・タイプの場合	目的
FLGCreateReg	FLGCreateType	新規のオブジェクト・タイプまたはオブジェクト・タイプ登録を定義する。
FLGGetReg	FLGGetType	オブジェクト・タイプまたはオブジェクト・タイプ登録に関する情報を獲得する。
FLGUpdateReg	FLGAppendType	オブジェクト・タイプまたはオブジェクト・タイプ登録の定義を変更する。
FLGDeleteReg	FLGDeleteType FLGDeleteTypeExt	オブジェクト・タイプまたはオブジェクト・タイプ登録を削除する。

オブジェクト・タイプを作成または削除するときには、FLGCreateReg 呼び出しと FLGCreateType 呼び出し、または FLGDeleteType 呼び出しと FLGDeleteReg 呼び出しを組み合わせて使用し、完全なオブジェクト・タイプが作成または削除されるようにする必要があります。特性が定義されたオブジェクト・タイプと関連付けられていないオブジェクト・タイプ登録は役に立たず、あとでそれらのオブジェクト・タイプを使用して情報カタログ内のオブジェクトを定義しようとすると、問題が発生する可能性があります。

オブジェクト・タイプを作成したあとで、オブジェクト・タイプ特性を変更または削除することはできません。FLGAppendType 呼び出し (77ページの『FLGAppendType』を参照) を使用して新規の任意選択特性を追加することができるだけです。

オブジェクト・タイプの定義

新規のオブジェクト・タイプを定義するときには、少なくとも以下の事項を指定する必要があります。

- 登録特性
- オブジェクト・タイプが属している区分
- すべてのオブジェクトに共通する必須特性

以上のステップを完了すると、オブジェクト・タイプに追加の任意選択特性を定義することができます。

登録特性の指定

オブジェクト・タイプを登録するときには、表2 に示す 6 つの特性を、このとおりの順序で指定する必要があります。

表 2. オブジェクト・タイプ登録のための必須特性

位置	特性省略名	特性名[1]	説明	コメント
1	NAME	オブジェクト・タイプの外部名	オブジェクト・タイプを表す 80 バイトの名前。	この値は FLGCreateReg 呼び出しを使用して設定する必要がある。 この値は、FLGUpdateReg 呼び出しを使用して修正することができる。
2	PTNAME	物理タイプ名	情報カタログ・データベースにおけるオブジェクト・タイプが入っている表の 30 文字の名前。	この値は FLGCreateReg 呼び出しによってのみ設定することができる。 この値は、オブジェクト・タイプの登録後に修正することはできない。
3	DPNAME	DP NAME	オブジェクト・タイプを表す 8 文字の省略名。	この値は FLGCreateReg 呼び出しを使用して設定する必要がある。 この値は、オブジェクト・タイプの登録後に修正することはできない。

表 2. オブジェクト・タイプ登録のための必須特性 (続き)

位置	特性省略名	特性名[1]	説明	コメント
4	CREATOR	CREATOR	オブジェクト・タイプを作成した管理者の 8 文字のユーザー ID。	この値は、オブジェクト・タイプに関して FLGCreateType 呼び出しが出されたときに情報カタログ・マネージャーによって設定される。 この値を設定または修正することはできない。
5	UPDATEBY	最終変更者	オブジェクト・タイプを最後に修正した管理者の 8 文字のユーザー ID。	この値は、オブジェクト・タイプに任意選択の特性を追加するために FLGAppendType 呼び出しが出されたときに、情報カタログ・マネージャーによって設定および修正される。
6	UPDATIME	最終変更日付および時刻	オブジェクト・タイプが最後に修正された日付と時刻の、26 文字のタイム・スタンプ。	この値は、オブジェクト・タイプに関する FLGCreateType または FLGAppendType 呼び出しが出されたときに、情報カタログ・マネージャーによって設定および修正される。

注:

1. この列の特性名は、情報カタログ・マネージャーの英語バージョンに適用します。情報カタログ・マネージャーの翻訳バージョンを使用している場合は、特性名も翻訳されます。

新規のオブジェクト・タイプの区分の指定

オブジェクト・タイプの区分は、FLGCreateReg を使用してオブジェクト・タイプを登録するときに設定します。

以下の区分に属するオブジェクト・タイプを作成することができます。

- Grouping
- Elemental
- Contact
- Dictionary
- Support

これらの 5 つの区分について、5ページの『区分によるオブジェクトの編成』で簡単に説明されています。詳細については、*情報カタログ・マネージャー 管理の手引き* を参照してください。

新規の情報カタログ・データベースを作成するときに、情報カタログ・マネージャー は Programs および Comments オブジェクト・タイプの両方を定義します。 Program 区分に属することができるオブジェクト・タイプは Programs だけです。その他の Program オブジェクト・タイプを作成することはできません。 Attachment 区分に属することができるオブジェクト・タイプは Comments だけです。その他の Attachment オブジェクト・タイプを作成することはできません。

必須オブジェクト・タイプ特性の定義

新規のオブジェクト・タイプを定義するときには、そのオブジェクト・タイプの最初の 5 つの特性として、12ページの表3 に示された 5 つの必須特性を指定する必要があります。情報カタログ・マネージャーは、特性省略名を使用して必須特性を識別します。

表3. 各オブジェクト・タイプの必須特性

位置	特性省略名	特性名	説明	コメント
1	OBJTYPID	オブジェクト・タイプ識別子	オブジェクト・タイプを表す 6 文字のシステム生成 ID。	情報カタログ・マネージャーが各オブジェクト・タイプに固有の識別子を生成する。 この値は FLGID の最初の部分である。いくつかの API 呼び出しでは、 FLGIDによってオブジェクト・インスタンスを識別する。 この値を修正することはできない。
2	INSTIDNT	インスタンス識別子	オブジェクト・インスタンスを表す 10 文字のシステム生成 ID。	情報カタログ・マネージャーが各オブジェクト・インスタンスに固有の識別子を生成する。 この値は FLGID の 2 番目の部分である。いくつかの API 呼び出しでは、 FLGID によってオブジェクト・インスタンスを識別する。 この値を修正することはできない。

表 3. 各オブジェクト・タイプの必須特性 (続き)

位置	特性省略名	特性名	説明	コメント
3	NAME	名前	オブジェクトを表す 80 バイトのユーザー指定名。	この値は、情報カタログ・マネージャーによって表示される。 この値は、FLGUpdateInst 呼び出しを使用して修正することができる。
4	UPDATIME	最終変更日付および時刻	オブジェクト・インスタンスが最後に修正された日付と時刻の、26 文字のタイム・スタンプ。	この値は、(FLGCreateInst または FLGUpdateInst 呼び出しを使用して) オブジェクト・インスタンスが作成または修正されるときに、情報カタログ・マネージャーによって設定される。 この値を修正することはできない。
5	UPDATEBY	最終変更者	オブジェクト・インスタンスを最後に修正した人の 8 文字のユーザー ID。	この値は、(FLGCreateInst または FLGUpdateInst 呼び出しを使用して) オブジェクト・インスタンスが作成または修正されるときに、情報カタログ・マネージャーによって設定および修正される。

これらの必須特性の特性省略名は予約されています。これらの名前は他の特性省略名に割り当てないでください。

新規のオブジェクト・インスタンスを作成するときには、NAME の値を指定しなければなりません。 OBJTYPID、INSTIDNT、UPDATIME、および UPDATEBY の値は、情報カタログ・マネージャーによって生成されます。これらのシステム生成値を修正することはできません。

新規のオブジェクト・タイプおよびオブジェクト・インスタンスの識別

システムによって OBJTYPID が生成されたあとで、ユーザーはこの値を使用して、登録および定義するオブジェクト・タイプを固有に識別します。

システムによって INSTIDNT が生成されたあとで、ユーザーはこの値と OBJTYPID を使用して、単一のオブジェクト・インスタンスを固有に識別します。

本書の 75ページの『第5章 情報カタログ・マネージャー API 呼び出しの構文』では、OBJTYPID 値と INSTIDNT 値の組み合わせを *FLGID* と呼びます。

情報カタログ・マネージャーの識別名

情報カタログ・マネージャーはいくつかの異なるレベルのユーザーが使用するように設計されているため、さまざまな製品ユーザーのオブジェクト・タイプを記述するために、異なる用語が使用されています。情報カタログ・マネージャー・インターフェース、および*情報カタログ・マネージャー 使用者の手引き*と*情報カタログ・マネージャー 管理の手引き* には、あまり技術的ではないビジネス向けの用語が使用されています。

本書では、管理者およびアプリケーション・プログラマーのために、データ処理環境向けの用語を使用します。

ユーザーまたは管理者用のアプリケーションを作成する場合には、これらの用語の相違に注意する必要があります。 表4 は、異なるレベルの用語の早見表です。

表 4. オブジェクト・タイプに関する情報カタログ・マネージャーの用語

説明	ユーザーの用語	管理者の用語	タグ言語の用語	API 呼び出しの用語
オブジェクト・タイプの長い (80 バイトの) 名前	オブジェクト・タイプ	オブジェクト・タイプ名	EXTNAME(*ext_name*)	オブジェクト・タイプの外部名 入出力構造では NAME 特性

表 4. オブジェクト・タイプに関する情報カタログ・マネージャーの用語 (続き)

説明	ユーザーの用語	管理者の用語	タグ言語の用語	API 呼び出しの用語
オブジェクト・タイプの短い (8 文字の) 名前	－	省略名	TYPE (*type*)	DP NAME 入出力構造では DPNAME 特性
オブジェクト・タイプ情報が入っている情報カタログ・データベース表の名前	－	－	PHYNAME (*table_name*) PHYNAME が指定されていない場合には、TYPE (*type*)	物理タイプ名 入出力構造では PTNAME 特性
長い (80 バイトの) 特性名	特性	特性名	EXTNAME(*ext_name*)	特性名
特性の短い (8 文字の) 名前	－	省略名	SHRTNAME (*short_name*)	特性省略名

情報カタログ・マネージャーの識別名

第3章 情報カタログ・マネージャー API 呼び出しを使用するプログラムの作成

情報カタログ・マネージャーでは、ユーザーのプログラムで情報カタログ・マネージャー機能を使用するための C 言語による API 呼び出しが提供されます。

この章では、以下の事項を説明します。
- API 呼び出しを使用することによって実行できる情報カタログ・マネージャー機能
- API 呼び出しの一般的な構造
- 情報カタログ・マネージャー API 呼び出しとの間でのデータの受け渡し
- 情報カタログ・マネージャーで提供される C 言語のヘッダー・ファイル
- 情報カタログ・マネージャーを使用して C 言語プログラムを作成する方法
- 情報カタログ・マネージャー API 呼び出しを使用するための規則
- DG2SAMP.C サンプル・プログラム

情報カタログ・マネージャー API 呼び出しによって行えること

情報カタログ・マネージャーの API 呼び出しには、一貫した構文規則があります。各 API 呼び出しの完全な構文については、75ページの『第5章 情報カタログ・マネージャー API 呼び出しの構文』を参照してください。

これらの API 呼び出しでは、自己定義の入出力構造が使用されます。これらの構造の読み取りおよび生成は、どのようなプログラミング言語でも行うことができます。入力構造と出力構造の詳細については、35ページの『第4章 情報カタログ・マネージャー入力および出力構造』を参照してください。

このセクションでは、情報カタログ・マネージャーで提供されるすべての API 呼び出しを簡単に説明し、各呼び出しの詳しい説明の記載箇所を示します。

情報カタログ・マネージャー・アプリケーション・サポートの提供

ユーザーのプログラムでこれらの API 呼び出しを使用すると、他の情報カタログ・マネージャー API 呼び出しを使用できるようになります。

API 呼び出し	目的	参照ページ
FLGInit	必要な資源を割り振り、情報カタログ・マネージャー・クライアントを初期設定する。	154
FLGFreeMem	情報カタログ・マネージャーによって定義された出力構造を解放する。	137
FLGTerm	資源を解放し、情報カタログ・マネージャー・クライアントを終了させる。	243
FLGTrace	追跡のレベルを設定する。	245

オブジェクト・タイプ登録の管理

登録を行うことにより、情報カタログ・マネージャーに対してオブジェクト・タイプが固有に識別されます。

API 呼び出し	目的	参照ページ
FLGCreateReg	新規のオブジェクト・タイプを登録する。	93
FLGDeleteReg	オブジェクト・タイプ登録を削除する。	111
FLGGetReg	オブジェクト・タイプ登録に関する情報を獲得する。	143
FLGUpdateReg	オブジェクト・タイプ登録に関する情報を更新する。	254
FLGManageIcons	オブジェクト・タイプを表すアイコンを作成、更新する。	198

オブジェクト・タイプの管理

オブジェクト・タイプは、いくつかの関連する特性を定義するものです。

API 呼び出し	目的	参照ページ
FLGAppendType	オブジェクト・タイプに新規の特性を追加する。	77
FLGCreateType	新規のオブジェクト・タイプを作成する。	101
FLGDeleteType	オブジェクト・タイプを削除する。	118

API 呼び出し	目的	参照ページ
FLGDeleteTypeExt	オブジェクト・タイプを、そのインスタンスとオブジェクト・タイプ登録とともに削除する。	121
FLGGetType	オブジェクト・タイプに関する情報を獲得する。	147

オブジェクト・インスタンスの管理

オブジェクト・インスタンスには、情報の単位を表すメタデータが含まれます。

API 呼び出し	目的	参照ページ
FLGCreateInst	新規のオブジェクト・インスタンスを作成する。	87
FLGDeleteInst	オブジェクト・インスタンスを削除する。	108
FLGDeleteTree	Grouping オブジェクト・インスタンスを削除し、基礎となるすべてのインスタンスを任意に削除する。	113
FLGUpdateInst	オブジェクト・インスタンスに関する情報を更新する。	248
FLGGetInst	オブジェクト・インスタンスに関する情報を獲得する。	139

情報カタログ・マネージャー識別子の管理

この API 呼び出しで、プログラムのパフォーマンス目的に合わせて識別子を変換することができます。

API 呼び出し	目的	参照ページ
FLGConvertID	アプリケーション・パフォーマンスに合わせてオブジェクト・タイプとインスタンス ID を変換する。	85

オブジェクトの関係の定義

関係とは、2 つのオブジェクト・インスタンスの相互関係を定義するものです。

API 呼び出し	目的	参照ページ
FLGRelation	2 つのオブジェクト・インスタンスの間の包含、連絡先、アタッチメント、またはリンク関係を作成または削除する。	221

オブジェクト・インスタンスの探索

特定の特性の値を使用して、オブジェクト・インスタンスを見つけることができます。

API 呼び出し	目的	参照ページ
FLGSearch	特定のオブジェクト・タイプの、選択基準を満たすインスタンスのリストを戻す。	227
FLGSearchAll	任意のオブジェクト・タイプの、選択基準を満たすインスタンスのリストを戻す。	237

オブジェクトのタイプおよびインスタンスのリスト

区分または関係にもとづいて、オブジェクトのタイプまたはインスタンスのリストを検索することができます。

API 呼び出し	目的	参照ページ
FLGFoundIn	以下のリストを戻す。特定のインスタンスが含まれているオブジェクト。特定のインスタンスが連絡先となるオブジェクト。特定のインスタンスがコメントとして付加されているオブジェクト。特定の Programs インスタンスが関連しているオブジェクト・タイプ。	131
FLGListAnchors	他のオブジェクトに含まれていない Grouping オブジェクトのリストを戻す。このような最上位の Grouping オブジェクトは、アンカー と呼ばれる。	161

API 呼び出し	目的	参照ページ
FLGListAssociates	以下のオブジェクトのリストを戻す。指定された Grouping オブジェクトに含まれているオブジェクト。指定されたオブジェクトが連絡先となるオブジェクト。指定されたオブジェクトに付加されたコメントであるオブジェクト。指定されたオブジェクトにリンクされているオブジェクト。または、指定されたオブジェクト・タイプに関連する Programs。	164
FLGListContacts	指定された Grouping または Elemental オブジェクトに関連するすべての Contact オブジェクトのリストを戻す。	173
FLGListObjTypes	すべてのオブジェクト・タイプのリストを戻す。	177
FLGListOrphans	現在、関連していない Attachment、Contact または Program オブジェクト・インスタンスのリストを戻す。	180
FLGListPrograms	Programs 以外のオブジェクト・タイプに関連したすべての Programs オブジェクトのリストを戻す。	187
FLGNavigate	指定された Grouping オブジェクトに含まれる Grouping または Elemental オブジェクトのリストを戻す。	215
FLGWhereUsed	指定されたオブジェクトを含む Grouping オブジェクトのリストを戻す。	259

情報カタログ・マネージャーとの間でのメタデータ・オブジェクトのコピー

情報カタログ・データベースとの間でメタデータをインポートまたはエクスポートすることができます。

API 呼び出し	目的	参照ページ
FLGExport	情報カタログ・マネージャー・メタデータ・オブジェクトをタグ言語形式のファイルにコピーし、変換する。	124
FLGImport	タグ言語形式のファイル内のメタデータ・オブジェクトを解釈し、情報カタログにコピーする。	150
FLGMdisExport	情報カタログ・マネージャー・メタデータ・オブジェクトを MDIS 適合タグ言語形式のファイルにコピーし、変換する。	124
FLGMdisImport	MDIS 適合タグ言語形式のファイル内のメタデータ・オブジェクトを解釈し、情報カタログにコピーする。	150

外部プログラムの開始

DOS または Microsoft Windows のアプリケーションを情報カタログ・マネージャーから開始することができます。

API 呼び出し	目的	参照ページ
FLGOpen	指定されたオブジェクトから得られる情報を使用して外部プログラムを開始する。	219

情報カタログ・マネージャー・データベースに対して行われた変更の確認または除去

情報カタログ・マネージャー・データベースに対して行われた変更をコミットまたはロールバックすることができます。

API 呼び出し	目的	参照ページ
FLGCommit	情報カタログ・データベースに対して行った変更を永続的なものにすることを確認する。	82
FLGRollback	情報カタログ・データベースに対して行った変更を、変更が最後にコミットされた箇所までさかのぼって除去する。	225

企業情報カタログの管理

オブジェクト管理タスクの実行を許可されたユーザーのリストを管理し、ユーザーに利用可能なコメント状況の選択項目を選択し、企業内のある情報カタログからシャドウの情報カタログへ削除を伝搬することができます。

API 呼び出し	目的	参照ページ
FLGManageUsers	管理者を更新し、オブジェクト管理権限を特定のユーザーに付与する。	203
FLGManageCommentStatus	ユーザーがコメントを割り当てるのに利用可能な状況選択項目のリストを設定、および更新する。	190
FLGManageFlags	情報カタログ削除の記録 (削除履歴) を開始、または停止する。または現在の設定値を検索する。	195
FLGManageTagBuf	現在記録されている削除履歴を照会、またはリセットする。	201
FLGXferTagBuf	他のカタログにインポートするために、削除履歴をタグ・ファイルに転送する。	263

情報カタログ・マネージャー API 呼び出しの出し方

すべての情報カタログ・マネージャー API 呼び出しの標準的な構造は、次のようになっています。

```
rc = FLGxxx (parameter,
             parameter,
             parameter,
             .
             .
             &ExtCode);
```

これらのパラメーターには、一般には、コードにおけるその API 呼び出しよりも前の部分で値またはアドレスが割り当てられています。

rc は、その API 呼び出しで戻される理由コードを表す変数です。ゼロ (0) の理由コードは、その API 呼び出しがエラーも警告もなしに完了したことを表します。 &ExtCode は、その API 呼び出しにより戻される拡張コードのアドレスです。

情報カタログ・マネージャー API 呼び出しとの間でのデータの受け渡し

情報カタログ API 呼び出しは、パラメーターと、入力構造および出力構造との 2 つのメカニズムを使用して入力を受け取り、出力を提供します。

パラメーターによる単一の入力値およびポインターの受け渡し

パラメーターを使用して、単一の入力値およびポインターを出力値およびデータ構造に提供することができます。

API 呼び出しパラメーターが文字列である場合、これらのパラメーターはすべて、 *NULL 文字で終了するストリング* として渡さなければなりません。 75ページの『第5章 情報カタログ・マネージャー API 呼び出しの構文』における各 API 呼び出しの**構文**のセクションでは、このようなパラメーターを説明する際には、NULL 文字を含まない実際のデータの最大長が指定されています。たとえば、オブジェクト・タイプ識別子 **ObjTypeID** の長さは、7 ではなく 6 と指定されています。

ただし、例の中でのこの種のパラメーターの C 宣言では、NULL 文字のための追加バイトが含まれています。たとえば、DG2API.H ファイルで #define 定数を使用する場合、**ObjTypeID** パラメーターを次のように宣言することが考えられます。

```
uchar objtypid[FLG_OBJTYPID_LEN+1]
```

(DG2API.H ファイル内の定数のリストについては、267ページの『付録B. 情報カタログ・マネージャー API ヘッダー・ファイル － DG2APIH』を参照してください。)

入力構造および出力構造による複数の値の受け渡し

情報カタログ・マネージャー API 呼び出しに複数の入力値を提供したり、情報カタログ・マネージャー API 呼び出しから複数の出力値を受け取ったりする場合には、入力構造と出力構造を使用する必要があります。

入力構造および出力構造は自己定義データ構造です。つまり、それぞれの構造で、その構造が渡すデータの形式と意味が定義されています。

それぞれの自己定義構造は、記憶域内で連続した区域になっていなければなりません。入力構造および出力構造には文字データだけが含まれ、NULL 文字を含めることはできません。

それぞれの入力構造および出力構造には、次の 2 つの区域が含まれていなければなりません。

ヘッダー域　構造のサイズを識別し、定義します。

定義域　オブジェクト区域特性を定義します。

定義域で定義されている特性の値を定義したり受け取ったりする構造には、その定義域で定義されている特性の値を指定するオブジェクト域 も含まれている必要があります。 図3 は、これらの 3 つの区域がどのような形でいっしょに収められるのかを示しています。

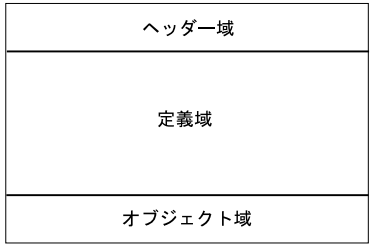


図3. 入出力構造

API 呼び出しに入力構造を渡すためには、入力構造を作成して、その入力構造の先頭を指すポインターを API 呼び出しの入力パラメーターとして渡してください。

出力構造から情報を検索するには、入力パラメーターとして NULL ポインターのアドレスを渡し、情報カタログ・マネージャーがそのポインターに出力構造の先頭のアドレスを割り当てられるようにしてください。

たとえば、API 呼び出しに ppListStruct という名前のポインターを渡すものとします。このポインターには pOutStruct という名前の NULL ポインターが含まれます。この API 呼び出しは、図4 に示すように、出力構造のアドレスを pOutStruct に割り当てます。

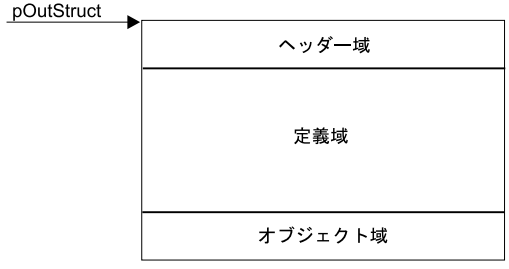


図4. 出力構造を指すポインター

複数の API 呼び出しを呼び出すことによってメモリーを使い切ることのないように、ユーザーのプログラムで情報カタログ・マネージャーの API 呼び出し FLGFreeMem を使用して、この出力構造に割り振られたメモリーを割り振り解除することができます。 FLGFreeMem の使い方の詳細については、137ページの『FLGFreeMem』を参照してください。

ヘッダー・ファイルの組み込み

情報カタログ・マネージャーには、情報カタログ・マネージャー API 呼び出しの関数原型、定数、データ・タイプ、および情報カタログ・マネージャー理由コードの定数を定義する C 言語のヘッダー・ファイルが備わっています。

情報カタログ・マネージャーを使用するには、ユーザーのプログラムに以下のヘッダー・ファイルを組み込む必要があります。

DG2API.H 使用頻度の高い値の定数、情報カタログ・マネージャーに特有のデータ・タイプ、および API 呼び出しの関数原型を定義します。

267ページの『付録B. 情報カタログ・マネージャー API ヘッダー・ファイル － DG2APIH』には、DG2API.H で定義される内容の完全なリストが記載されています。

DG2ERR.H 情報カタログ・マネージャー理由コードの定数を定義します。

情報カタログ・マネージャーを使用するためには、プログラムに以下の `#define` および `#include` ステートメントを組み込む必要があります。

```
#define DGWIN32
#include WINDOWS.H
#include DG2API.H
#include DG2ERR.H
```

WINDOWS.H は Microsoft Visual C++ コンパイラーの一部です。このファイルには、情報カタログ・マネージャーで使用される Windows データ・タイプの標準宣言を定義するヘッダー・ファイルが組み込まれています。

C 言語プログラムの作成の概要

このセクションでは、情報カタログ・マネージャー API 呼び出しを使用する C 言語プログラムの作成および実行のためのステップについて、簡単に説明します。ここに述べる情報のほとんどは、ユーザーが作成するどの C 言語プログラムにも標準的に適用されます。

C 言語ソース・コードの作成

C 言語を使用して情報カタログ・マネージャー・アプリケーションを作成するには、次のようにしてください。

1. ソース・コードを作成します。
2. C コンパイラーを使用して、そのソース・コードをコンパイルします。
3. オブジェクト・ファイルを情報カタログ・マネージャーおよび C 言語ライブラリーに連係して、実行可能プログラムを作成します。

   情報カタログ・マネージャー・ライブラリーは DGWAPI.LIB です。
4. アプリケーションを実行します。

環境の設定

以下のことを行って、C 言語で書かれた情報カタログ・マネージャー・プログラムをコンパイルおよび実行するための環境を設定してください。

1. コンパイラーを導入します。
2. LIBPATH を検査します。

   LIBPATH= 環境変数には、*x*:**¥SQLLIB** ディレクトリーを含める必要があります。この *x* は、DB2 UDB が導入されているドライブです。
3. 環境変数を設定します。 AUTOEXEC.BAT で、あるいは Microsoft Visual C++ コンパイラーのメニュー・バーから環境変数を設定します (ファイルのパスとライブラリーのパスを含めます)。

   SET INCLUDE= ステートメントには、*x*:**¥SQLLIB¥LIB** ディレクトリーを含める必要があります。 WINDOWS.H を含むこのディレクトリーについては、SET INCLUDE= でも指定するようにします。

   SET LIB= には、*x*:**¥SQLLIB¥LIB** ディレクトリーを含める必要があります。

アプリケーションのコンパイルと連係

Microsoft Visual C++ コンパイラーを使用してアプリケーションをコンパイルするには、次のようなコマンドを出す必要があります。

cl /c *filename*.**c**

使用するコンパイラーおよびプログラムの書き方によっては、別のオプションの追加が必要となったり、可能であったりすることがあります。

プログラムを連係するには、次のようなコマンドを出してください。

link /dll dgwapi.lib *filename*.**obj**

プログラムで情報カタログ・マネージャー API 呼び出しを使用する方法

情報カタログ・マネージャー API 呼び出しを含む C 言語プログラムを書く際には、特定の規則および指針に従う必要があります。このセクションでは、これらの指針について説明します。

FLGInit によるプログラムの開始

情報カタログ・マネージャー API 呼び出しを出すプログラムを作成する場合、FLGInit 呼び出しを出してからでなければ、その他の情報カタログ・マネージャー API 呼び出しを出すことができません。

FLGInit は情報カタログ・マネージャーを初期設定し、情報カタログ・マネージャー・オブジェクト・タイプおよび登録に必要な特性の名前を戻し、環境情報を戻します。

FLGInit によって戻された情報を保管してください。この情報は、他の情報カタログ・マネージャー API 呼び出しを出すために必要になることがあります。各国語版の情報カタログ・マネージャーを使用している場合、FLGInit は、情報カタログ・マネージャー・オブジェクト・タイプおよび登録に必要な特性の翻訳された名前を戻します。オブジェクト・タイプおよび登録を作成または保守するときに、入力構造の定義域でこれらの翻訳名を使用する必要があります。

FLGInit 出力構造の内容については、154ページの『FLGInit』を参照してください。

FLGTerm によるプログラムの終了

情報カタログ・マネージャー機能を使用し終わったプログラムは、FLGTerm 呼び出しをする必要があります。 FLGTerm は情報カタログ・マネージャー・セッションを終了させ、情報カタログ・マネージャーによって使用されていた資源を解放します。この API 呼び出しの詳細については、243ページの『FLGTerm』を参照してください。

エラーが発生した場合の情報カタログ・データベースの保護

いくつかの情報カタログ・マネージャー・エラーでは、情報カタログが存在するデータベース内のメタデータの一部が矛盾している可能性があることが示されます。したがって、プログラムでエラーが発生したときに情報カタログ・データベースをロールバックするようにプログラムを書く必要があります。 API 呼び出しが正常に行われたときに FLGCommit 呼び出しを出し、失敗したとき

に FLGRollback 呼び出しを出すことにより、情報カタログ・データベースに矛盾が生じないようにすることができます。

考慮事項: 情報カタログ・データベースが DB2 上で使用されている場合、エラーが発生したときには FLGRollback 呼び出しを出さなければなりません。そのようにしないと、プログラムが FLGTerm を出したときに情報カタログ・データベースが損傷を受ける可能性があります。

プログラムを開始するための Programs オブジェクトの設定

情報カタログ・マネージャー・アプリケーションから得られたデータを使用するプログラムを開始するには、その種類のデータを表すオブジェクト・タイプに関連付けられた Programs オブジェクト・インスタンスを作成してください。

表5 に示すように、プログラムを識別する 3 つの特性を Programs オブジェクト・インスタンス内で定義し、Programs オブジェクト・インスタンスとオブジェクト・タイプを関連付ける必要があります。

表 5. プログラムを開始する Programs オブジェクト・インスタンスの特性

特性名	特性省略名	値
呼び出しによる開始	STARTCMD	開始されるプログラムのパスとファイル名、および開始オプション。
このプログラムが扱うオブジェクト・タイプ	HANDLES	オブジェクト・タイプを表す 8 文字の省略名。
パラメーター・リスト	PARMLIST	関連するオブジェクト・タイプにおける、プログラムにコマンド行パラメーターとして渡したい特性のリスト。各特性は 2 つのパーセント記号 (%%) によって区切られる。

呼び出しによる開始 (STARTCMD) 特性の値に推奨される形式は、そのプログラムのインターフェース・タイプによって異なります。プログラムのファイル名と、推奨される開始パラメーターを入力します。 Windows NT®、Windows 95、および Windows 98 の場合、推奨される開始パラメーターは `START filename.exe` です。 PATH ステートメントには、プログラムが存在するディレクトリーを指定する必要があります。

プログラムのファイル名がハイ・パフォーマンス・ファイル・システム (HPFS) 形式になっていて、ブランクが含まれている場合には、次のように、そのプログラムのパスとファイル名を二重引用符で囲む必要があります。

```
""D:¥PROGPATH¥My Program.EXE""
```

プログラム名にブランクが含まれている場合には、STARTCMD 特性値で別の開始オプションを指定することはできません。

プログラムを開始するには、Programs オブジェクト FLGID とオブジェクト・インスタンス FLGID をパラメーターとして指定した FLGOpen 呼び出しを出してください。 FLGOpen 呼び出しの詳細については、219ページの『FLGOpen』を参照してください。

API 呼び出しによるメタデータの作成

登録、オブジェクト・タイプ、オブジェクト・インスタンス、および関係は、相互に依存し合って作成されます。したがって、これらのエンティティーの集合は、特定の順序でなければ作成することができません。新規のオブジェクト・タイプ、オブジェクト・インスタンス、および関係を作成するときには、次の順序で情報カタログ・マネージャー API 呼び出しを出す必要があります。

1. FLGCreateReg
2. FLGCreateType
3. FLGCreateInst
4. FLGRelation

API 呼び出しによるメタデータの削除

一方で、登録、オブジェクト・タイプ、オブジェクト・インスタンスおよび関係の削除には、2 通りあります。保守的な方法 (時間がかかります) と、破壊されてしまう可能性のある方法 (しかし速くできます) です。

オブジェクト・タイプおよびオブジェクト・インスタンスを保守的な方法で削除するときには、次の順序でそのオブジェクト・インスタンスおよびオブジェクト・タイプに関する情報カタログ・マネージャー API 呼び出しを出します。

1. FLGRelation

   特定のオブジェクト・インスタンスが他のオブジェクトを含む という関係をすべて削除しないと、これらのオブジェクト・インスタンスを削除することはできません。 FLGDeleteInst は、オブジェクト・インスタンスが含まれている、あるいは Contact、Attachment、またはリンク・オブジェクトに関連していたという関係を自動的に削除します。

2. FLGDeleteInst

   FLGDeleteType を使ったオブジェクト・タイプを削除する前に、特定のオブジェクト・タイプのオブジェクト・インスタンスをすべて削除しなくてはなりません。

3. FLGDeleteType
4. FLGDeleteReg

次の API を使うと、オブジェクト・インスタンスおよびオブジェクト・タイプの削除をより速く行うことができますが、情報カタログの内容を完全に理解していないと、破壊される恐れがあります。

1. FLGDeleteTree

   Grouping オブジェクト・インスタンスと、任意で、それに含まれるオブジェクト・インスタンスに関連するすべての関係に加え、それが含むすべてのオブジェクト・インスタンスを同時に削除します。

2. FLGDeleteTypeExt

   オブジェクト・タイプ、オブジェクト・タイプ登録、およびオブジェクト・タイプのすべてのインスタンスを同時に削除します。 FLGDeleteTypeExt を使ってオブジェクト・タイプを削除する前に、他のオブジェクト・タイプのオブジェクトを含む個々のブランチを削除しなくてはなりません。

情報カタログ・マネージャー・データ・タイプを使用する情報カタログ・メタデータの指定

情報カタログ・マネージャーは、表6 で定義されている 4 つのデータ・タイプを使用して、オブジェクトの特性に関するメタデータを保管します。

ユーザーのプログラムでは、メタデータが有効な形式になるように、なんらかのデータ変換を行わなければならないことがあります。

表 6. 情報カタログ・メタデータのための有効なデータ・タイプ

データ・タイプ	表現方法	入力および出力構造における省略された値の表現方法
CHAR	定義された長さを占める。定義された長さよりも値が短い場合には、値の右側に後続ブランクが埋め込まれる。	その値に定義された長さだけブランクが入る。
TIMESTAMP	yyyy-mm-dd-hh.mm.ss.nnnnnn 形式で全部の長さ (26) を占める。	26 個のブランクで表現。

表 6. 情報カタログ・メタデータのための有効なデータ・タイプ (続き)

データ・タイプ	表現方法	入力および出力構造における省略された値の表現方法
LONG VARCHAR	後続の値の実際の長さを表す 8 文字長フィールドが前に付く。	長さフィールドは、ゼロにセットされて、そのあとに値がないことを表す。 例: `00000000`
VARCHAR	後続の値の実際の長さを表す 8 文字長フィールドが前に付く。	長さフィールドはゼロにセットされて、そのあとに値がないことを表す。 例: `00000000`

入力構造については、要求の検査と受け入れを行う前に、情報カタログ・マネージャーは変数値から自動的に後続ブランクを除去し、その結果に合わせて長さを調整します。したがって、必須変数値にブランクだけが指定されている場合には、その要求は拒否され、必須値が指定されなかったことを示す理由コードが戻されます。値が必須であっても利用不能なときは、エラーを防ぐために非適用記号を使うことができます。

各国語の考慮事項

特に明記されていない限り、情報カタログ・マネージャーのコマンド、パラメーター、必須特性省略名、データ・タイプ名、インディケーター値、およびオプション値は各国語版用には翻訳されないため、英語で入力しなければなりません。

翻訳された必須特性

必須の登録特性およびオブジェクト・タイプ特性の 80 バイトの名前は、各国語に翻訳されます。

必須登録特性の日本語名は以下のとおりです。
- オブジェクト・タイプの外部名
- 物理タイプ名
- DP 名
- 作成者
- 最終変更者
- 最終変更日付および時刻

必須オブジェクト・タイプ特性の日本語名は以下のとおりです。
- オブジェクト・タイプ識別子

- インスタンス識別子
- 名前
- 最終変更日付および時刻
- 最終変更者

翻訳された名前は、FLGInit 呼び出しで作成される出力構造に入って戻されます。

英語以外の言語での値の指定

情報カタログに保管されるメタデータ値の多くは、どの言語でも保管することができます。このセクションでは、情報カタログ・マネージャーの値に SBCS 文字と DBCS 文字を使用する場合の指針を説明します。

SBCS 文字だけが使用される値

– DP 名 (オブジェクト・タイプ省略名) 値
– 特性省略名
– PT 名 (物理タイプ名) 値

SBCS と DBCS の両方が使用できる値

– 名前 (オブジェクト・タイプの外部名) 値
– オブジェクト・タイプおよび登録に関する必須特性以外の特性名
– ユーザー定義特性の特性値
– 以下の API 呼び出しパラメーターの値

FLGCreateReg	pszIconFileID
FLGGetReg	pszIconFileID
FLGExport	pszTagFileID、pszLogFileID、pszIcoPath
FLGImport	pszTagFileID、pszLogFileID、pszIcoPath
FLGInit	pszUserID、pszPassword、pszDatabaseName
FLGManageIcons	pszIconFileID
FLGMdisExport	pszTagFileID、pszLogFileID、pszObjTypeName、pszObjectName
FLGMdisImport	pszTagFileID、pszLogFileID
FLGUpdateReg	pszIconFileID
FLGXferTagBuf	pszTagFileID

DG2SAMP.C の概要

情報カタログ・マネージャーには、コンパイル、連係、および実行することのできるサンプル・プログラム DG2SAMP.C が用意されています。DG2SAMP.C は、情報カタログ・マネージャーが導入されたドライブの DG2LIB¥LIB ディレクトリーに入っています。

ヘッダー・ファイルの組み込み

本書では、情報カタログ・マネージャー API 呼び出しを使用するアプリケーションの書き方を示すために、DG2SAMP.C の一部を使用しています。DG2SAMP.C は、以下の呼び出しを出します。

- FLGCommit
- FLGFreeMem
- FLGGetInst
- FLGInit
- FLGListObjTypes
- FLGRollback
- FLGSearch
- FLGTerm
- FLGTrace
- FLGUpdateInst

DG2SAMP.C をコンパイルおよび連係する手順、およびプログラムの実行例については 265ページの『付録A. サンプル・プログラム DG2SAMP.C』を参照してください。

第4章 情報カタログ・マネージャー入力および出力構造

情報カタログ・マネージャー API 呼び出しは、パラメーター、入力構造、および出力構造を使用して入力を受け取り、出力を提供します。ユーザーは、入力構造と出力構造を使用して、情報カタログ・マネージャー API 呼び出しに複数の入力値を提供したり、情報カタログ・マネージャー API 呼び出しから複数の出力値を受け取ったりすることができます。

入力構造および出力構造は自己定義データ構造です。つまり、それぞれの構造で、その構造が渡すデータの形式と意味が定義されています。

API 呼び出しに入力構造を渡すためには、入力構造を作成して、その入力構造の先頭を指すポインターを API 呼び出しの入力パラメーターとして渡す必要があります。このプロセスについては 44ページの『API 呼び出しのための入力構造の作成』で説明します。

出力構造から情報を検索するには、1 つまたは複数のポインターを使用して出力構造を順に処理する必要があります。このプロセスについては 64ページの『API 呼び出しによって得られた出力構造の読み取り』で説明します。

本書の例は C 言語で書かれていますが、任意のプログラミング言語で入力および出力構造を作成することができます。

情報カタログ・マネージャー API の入出力構造の一般的な特性

情報カタログ・マネージャーの入力構造および出力構造は、図5 に示すように、 3 つの*区域* と呼ばれる部分からなります。

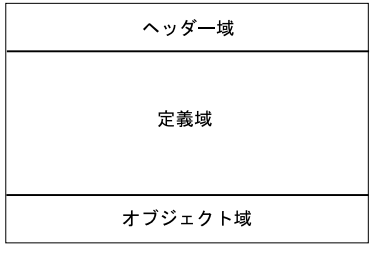


図5. 入出力構造

ヘッダー　　構造のサイズを識別し、定義します。

API の入出力構造の一般的な特性

定義　　オブジェクト区域特性を定義します。

オブジェクト　特性値を指定します。

この自己定義構造全体が、記憶域の連続区域になっていなければなりません。

入力構造および出力構造には文字データだけが含まれ、NULL 文字を含めることはできません。

入出力構造の値を省略する場合には、本書でブランク と呼んでいる間隔文字をその値の代りに使用して、値のバイト・オフセットと入力構造および出力構造の定義との整合性が維持されるようにしてください。

情報カタログ・マネージャー API 入力構造

図6 は、情報カタログ・マネージャーの API 入力構造の一般的な形式を示しています。この構造は、ヘッダー域、定義域、およびオブジェクト域という 3 つの連続する区域からなります。情報カタログ・マネージャー API 呼び出しによっては、最初の 2 つの区域だけしか必要としないものもあります。

各区域のフィールドについては、以下のセクションで説明します。

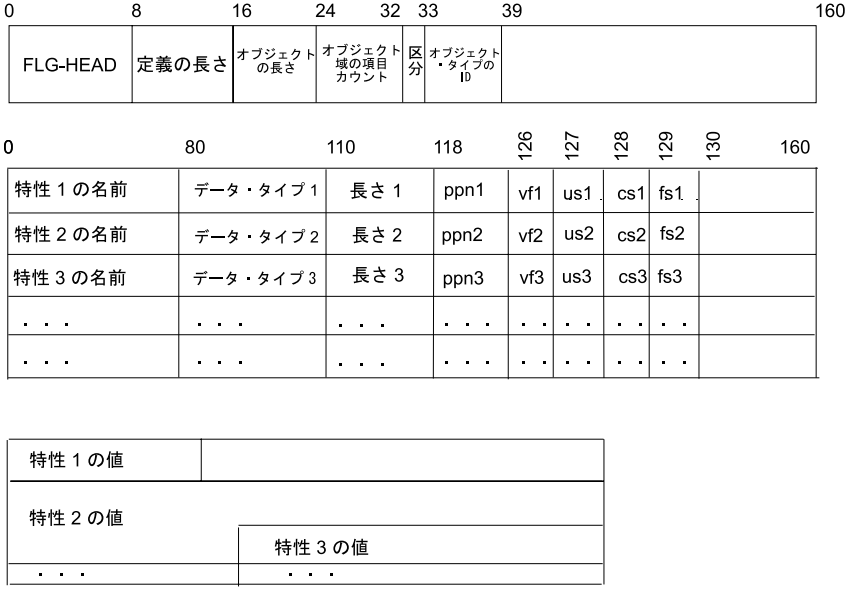


図 6. API 入力構造

以下の API 呼び出しは入力構造から入力を受け取ります。

FLGAppendType
オブジェクト・タイプに新規の特性を追加します。

FLGCreateInst
新規のオブジェクト・インスタンスを作成します。

FLGCreateReg
新規のオブジェクト・タイプを登録します。

FLGCreateType
新規のオブジェクト・タイプを作成します。

FLGExport 情報カタログ・マネージャー・メタデータ・オブジェクトをファイルにコピーし、タグ言語形式に変換します。

FLGManageCommentStatus
コメント用に利用可能な選択項目のリストを更新します。

FLGManageUsers
情報カタログの管理者とユーザーを更新し、各ユーザーの権限のエクステントを識別します。

FLGSearch 特定のオブジェクト・タイプの、選択基準を満たすインスタンスのリストを戻します。

FLGSearchAll 任意のオブジェクト・タイプの、選択基準を満たすインスタンスのリストを戻します。

FLGUpdateInst
オブジェクト・インスタンスに関する情報を更新します。

FLGUpdateReg
オブジェクト・タイプ登録に関する情報を更新します。

FLGSearch および FLGSearchAll は、入力構造を受け取らなかった場合には、すべてのオブジェクトの検索を試みます。

ヘッダー域 (常に必須)

ヘッダー域では、定義域とオブジェクト域の情報を記述します。必須でないフィールドで値を指定しないものは、ブランクにセットしなければなりません。

図 7. 入力構造のヘッダー域

表7 は、図7 で示したヘッダー域の各バイト・オフセット位置の意味を示しています。

表 7. 入力構造のヘッダー域とそのフィールド

図7 で示したセクション	バイト・オフセット	必須かどうか	説明
FLG-HEAD	0-7	常に必須	構造の識別子。
定義の長さ	8-15	常に必須	定義域の長さ。 この値は、160 の倍数 (160 に定義レコードの数を掛けた値) でなければならない。
オブジェクトの長さ	16-23	常に必須	オブジェクト域の長さ。 FLGAppendType および FLGCreateType の場合、この値はゼロ (00000000) である。
オブジェクト域の項目カウント	24-31	常に必須	オブジェクト域内の項目 (特性値) の数。 この値は、定義域内の特性の数にオブジェクト域で記述された値のセットの数を掛けた値になる。 FLGAppendType および FLGCreateType の場合、この値はゼロ (00000000) である。

表 7. 入力構造のヘッダー域とそのフィールド (続き)

38ページの図 7 で示したセクション	バイト・オフセット	必須かどうか	説明
区分	32	以下の場合には必須 • FLGAppendType • FLGCreateInst • FLGCreateReg • FLGCreateType • FLGUpdateInst • FLGUpdateReg	オブジェクト・タイプまたはオブジェクトの区分 有効な値は以下のとおり。 **G** Grouping **E** Elemental **C** Contact **P** Program **D** Dictionary **S** Support **A** Attachment
オブジェクト・タイプの ID	33-38	以下の場合には必須 • FLGAppendType • FLGCreateInst • FLGCreateType • FLGUpdateInst • FLGUpdateReg	オブジェクト・タイプに関するシステム生成された識別子。
	39-159	常に必須	ブランクのままにする必要がある。

定義域 (常に必須)

定義域には、特定の情報カタログ・マネージャー API 機能の入力パラメーターとして必要な特性定義の集合が入ります。

40ページの表8 は、入力構造を使用する各種の API 呼び出しにとって、定義域に入っている情報がどのような意味を持つのかを示しています。

表 8. 各種の API 呼び出しにおける定義域の意味

API 呼び出し	定義域で示される情報
FLGAppendType FLGCreateInst FLGCreateReg FLGCreateType FLGUpdateInst FLGUpdateReg	オブジェクト登録、オブジェクト・タイプ、またはオブジェクト・インスタンスを定義する特性の集合の定義。
FLGSearch FLGSearchAll	選択基準を記述する特性の集合の定義。
FLGExport	エクスポートされるメタデータを指定する特性の定義。
FLGManageCommentStatus	Comments 状況選択項目を指定する特性の集合の定義。
FLGManageUsers	情報カタログ・マネージャー・ユーザーを記述する特性の集合の定義。

定義域内の各特性は、形式設定された仕様の集合によって定義されます。表9 は、図8 で示されたバイト・オフセット位置を説明しています。

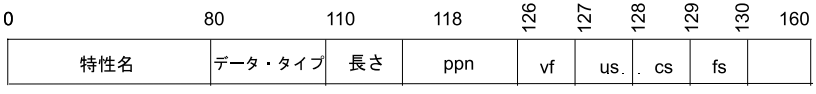


図 8. 入力構造の定義レコード

表 9. 入力構造の定義域とそのフィールド

図8 で示したセクション	バイト・オフセット	必須かどうか	説明
特性名	0-79	常に必須	特性の外部名。

表 9. 入力構造の定義域とそのフィールド (続き)

40ページの図 8 で示したセクション	バイト・オフセット	必須かどうか	説明
データ・タイプ	80-109	常に必須	特性のデータ・タイプ。 有効な値は以下のとおり。 **CHAR** 固定長文字データ。最大長は 254。 **VARCHAR** 可変長文字データ。最大長は 4000。 **LONG VARCHAR** 可変長文字データ。最大長は 32700。 **TIMESTAMP** `yyyy-mm-dd-hh.mm.ss.nnnnnn` 形式のタイム・スタンプ。タイム・スタンプの長さは 26。
長さ	110-117	常に必須	特性値の最大長。
ppn	118-125	以下の場合には必須 • FLGAppendType • FLGCreateInst • FLGCreateReg • FLGCreateType • FLGManage-CommentStatus • FLGSearch • FLGSearchAll • FLGUpdateInst • FLGUpdateReg その他の API 呼び出しの場合、このフィールドは未使用のためブランクのまま。	特性省略名

表 9. 入力構造の定義域とそのフィールド (続き)

40ページの図 8 で示したセクション	バイト・オフセット	必須かどうか	説明
vf	126	以下の場合には必須 • FLGAppendType • FLGCreateInst • FLGCreateReg • FLGCreateType • FLGUpdateInst • FLGUpdateReg その他の API 呼び出しの場合、このフィールドは未使用のためブランクのまま。	特性が必須であるのか、任意選択であるのか、あるいはシステム生成されるのかを指定する値フラグ。 有効な値は以下のとおり。 **R** 必須 **O** 任意選択 **S** システム生成
us	127	以下の API 呼び出しでは必須 • FLGCreateInst • FLGCreateType • FLGUpdateInst その他の API 呼び出しの場合、このフィールドは未使用のためブランクのまま。	汎用固有識別コード (UUI) の順序番号を指定。これにより、ある特性がその UUI のパートであることが示される。 有効な値は以下のとおり。 **1** UUI のパート 1 **2** UUI のパート 2 **3** UUI のパート 3 **4** UUI のパート 4 **5** UUI のパート 5 **(ブランク)** UUI の一部ではない どのオブジェクト・タイプについても、UUI のパート 1 として少なくとも 1 つは特性を指定しなければならない。 UUI のパートの定義の詳細については、*情報カタログ・マネージャー 管理の手引き* を参照。

表 9. 入力構造の定義域とそのフィールド (続き)

40ページの図 8 で示したセクション	バイト・オフセット	必須かどうか	説明
cs	128	以下の API 呼び出しでは必須 • FLGSearch • FLGSearchAll その他の API 呼び出しの場合、このフィールドは未使用のためブランクのまま。	大文字小文字の区別フラグ。 有効な値は以下のとおり。 **Y** 大文字小文字を区別する **N** 大文字小文字を区別しない 大文字小文字の区別フラグについては、227ページの『FLGSearch』および 237ページの『FLGSearchAll』を参照。
fs	129	以下の API 呼び出しでは必須 • FLGSearch • FLGSearchAll その他の API 呼び出しの場合、このフィールドは未使用のためブランクのまま。	ファジー探索フラグ。 有効な値は以下のとおり。 **Y** ファジー探索を行う **N** ファジー探索を行わない ファジー探索フラグについては、 227ページの『FLGSearch』および 237ページの『FLGSearchAll』を参照。
	130-159	常に必須	予約セクション。 ブランクのままにする必要がある。

オブジェクト域 (値を定義するときには必須)

オブジェクト域には、定義域で定義された特性の値が入ります。これらの値は、定義域で定義したとおりの順序で指定する必要があります。

入力構造のオブジェクト域には、FLGExport および FLGManageUsers 以外のすべての API の定義域で定義された各定義について 1 つずつしか値が入りません。 FLGExport および FLGManageUsers の場合のオブジェクト域には、定義域で定義された各特性について 1 つ以上の値を含めることができます。

オブジェクト域は、以下の API 呼び出しでは必須です。
- FLGCreateInst
- FLGCreateReg
- FLGExport

- FLGManageCommentStatus
- FLGManageUsers
- FLGSearch
- FLGSearchAll
- FLGUpdateInst
- FLGUpdateReg

それぞれの値を表現する方法は、以下の規則によって判別することができます。

データ・タイプ	オブジェクト域で値を表現する方法
VARCHAR	値の前に、その値の実際の長さを示す 8 文字長のフィールドが付きます。これらの値における後続ブランクは自動的に除去され、情報カタログ・マネージャーはそれに応じて長さフィールドを調整します。
LONG VARCHAR	値の前に、その値の実際の長さを示す 8 文字長のフィールドが付きます。これらの値における後続ブランクは自動的に除去され、情報カタログ・マネージャーはそれに応じて長さフィールドを調整します。
CHAR	値は定義域の特性の長さフィールドで定義されたバイト数を占め、定義された長さになるように右側に埋め込みが行われます。
TIMESTAMP	26 バイトです。

API 呼び出しのための入力構造の作成

入力構造を作成するには、以下のステップに従ってください。

1. DG2API.H を使用して長さと値を定義します。
2. 出力構造全体のサイズを計算します。
3. ヘッダー域を定義します。
4. 定義域を定義します。
5. オブジェクト域を定義します。

DG2API.H を使用して長さと値を定義する

情報カタログ・マネージャーでは、DG2API.H という名前の C 言語のヘッダー・ファイルが提供されます。このファイルは、ユーザーが入力構造を作成し

たり出力構造を読んだりするために必要な多くの値の長さおよび有効な値を定義しています。このファイルを (#include ステートメントを使用して) ユーザーのプログラムに組み込むと、特定のデータ・タイプ、構造、および関数原型を自分で書く必要がなくなります。

DG2API.H には、図9 に示すように、ヘッダー域と定義域の作成に必要な構造のタイプ定義 (typedef) 宣言が含まれます。 (図9 に示す WINDOWS とは、Microsoft Windows 3.1 のみを意味します。)

```
#pragma pack(1)
/* Structure definition for the FLG header area */
  typedef   struct _FLG_HEADER_AREA {
     UCHAR        pchHIdent          [ FLG_H_IDENT_LEN      ];
     UCHAR        pchHDefLength      [ FLG_H_DEFAREA_LEN    ];
     UCHAR        pchHObjLength      [ FLG_H_OBJAREA_LEN    ];
     UCHAR        pchHObjEntryCount  [ FLG_H_OBJAREAENT_LEN ];
     UCHAR        pchHCategory       [ FLG_H_CATEGORY_LEN   ];
     UCHAR        pchHObjTypeId      [ FLG_H_OBJTYPID_LEN   ];
     UCHAR        pchHReserved       [ FLG_H_RESERVED_LEN   ];
  } FLGHEADERAREA;
#ifdef WINDOWS
  typedef FLGHEADERAREA __huge *PFLGHEADERAREA;
#else
  typedef FLGHEADERAREA *PFLGHEADERAREA;
#endif
/* Structure definition for the FLG definition area */
  typedef   struct _FLG_DEFINITION_AREA {
     UCHAR        pchDPropName       [ FLG_D_PROPNM_LEN   ];
     UCHAR        pchDDataType       [ FLG_D_DATATYP_LEN  ];
     UCHAR        pchDDataLength     [ FLG_D_DATA_LEN     ];
     UCHAR        pchDTagName        [ FLG_D_PPN_LEN      ];
     UCHAR        pchDVF             [ FLG_D_VF_LEN       ];
     UCHAR        pchDUS             [ FLG_D_US_LEN       ];
     UCHAR        pchDCS             [ FLG_D_CS_LEN       ];
     UCHAR        pchDFS             [ FLG_D_FS_LEN       ];
     UCHAR        pchDReserved       [ FLG_D_RESERVED_LEN ];
  } FLGDEFINITIONAREA;
#ifdef WINDOWS
  typedef FLGDEFINITIONAREA __huge *PFLGDEFINITIONAREA;
#else
  typedef FLGDEFINITIONAREA *PFLGDEFINITIONAREA;
#endif
```

図 9. DG2API.H: ヘッダー域と定義域の構造定義

FLG_D または FLG_H で始まる変数は、DG2API.H で定義された構造部分の長さです。

DG2API.H ファイルで定義されているすべての定数のリストについては、267ページの『付録B. 情報カタログ・マネージャー API ヘッダー・ファイル − DG2APIH』を参照してください。

定義されたこれらの構造を使用して、入力構造のヘッダー域と定義域に必要な記憶域を定義することができます。図10 は、DG2API.H ヘッダー・ファイルで定義されたデータ・タイプを使用して、入力構造のヘッダー域と定義域を保管するためにあとで使用される構造を定義する、DG2SAMP.C の一部を示しています。

```
//  This structure defines the input structure for FLGSearch.
typedef _Packed struct SEARCH_STRUCT  {
   FLGHEADERAREA        srchHdr;
   FLGDEFINITIONAREA    srchDef;
   OBJECTAREA           Item;
} SEARCHSTRUCT;
typedef SEARCHSTRUCT  *PSEARCHSTRUCT;
```

図 10. DG2SAMP.C: ヘッダー域と定義域の定義

入力構造が連続する記憶域として定義されるようにするために、45ページの図9で `#pragma pack (1)` 命令が使用され、図10 で `typedef _Packed struct` 定義が使用されています。別のプログラミング言語を使用して入力構造を作成する場合には、入力構造を連続記憶域として定義するために類似のコマンドを出すことが必要な可能性もありますので、ご注意ください。

入力構造全体のサイズを計算する

入力構造用の記憶域を割り振ることができるように、入力構造全体のサイズを計算する必要があります。この計算を行うためには、以下の値が分かっている必要があります。

- 定義域内で定義されている特性の数

  この値は、API 呼び出しで必要となる特性の数によって異なります。この値を使用して、定義域の長さを計算します。
- オブジェクト域内の値の長さ。これらの値を合計して、オブジェクト域の長さを求めます。

DG2API.H では、ヘッダー域の長さ (FLG_HEADER_SIZE) および 1 つの定義レコードの長さ (FLG_DEFINITION_SIZE) を定義する変数が提供されます。

定義域の長さの計算

定義域の長さを計算するためには、図11 に示すように、各定義レコードの長さ (160) に、データを定義するために必要なレコードの数を掛けてください。

```
Definition_area_length = number_of_properties × FLG_DEFINITION_SIZE
```

図 11. 定義域の長さの計算

DG2API.H では、ユーザーのコードでこの計算を定義しやすくするために、160 として定義された変数 FLG_DEFINITION_SIZE が提供されています。

48ページの『ヘッダー域の定義』で示されているように、ヘッダー域の定義域長フィールドを定義するためには、この値が必要です。

オブジェクト域の長さの計算

オブジェクト域の長さは、オブジェクト域に入るすべての値の長さの合計です。

48ページの『ヘッダー域の定義』で示されているように、ヘッダー域のオブジェクト域長フィールドを定義するためには、この値が必要です。

オブジェクト域を必要としたり予期したりしない API 呼び出し用の入力構造を作成する場合には、オブジェクト域の値はゼロ (00000000) になります。

オブジェクト域の長さを正確に計算するには、オブジェクト域のすべての値の長さが分かっている必要があります。 CHAR 値と TIMESTAMP 値の場合には、定義域で定義された長さを使用してください。ただし、LONG VARCHAR 値と VARCHAR 値の場合には、各値の長さを調べたうえで、長さ値の一部として 8 バイトの長さフィールドを含める必要があります。この計算のための式を図12 に示します。

```
オブジェクト域の長さ = 特性 1 の長さ +
                       特性 2 の長さ +
                       特性 3 の長さ +
                     .
                     .
                     .
```

図 12. オブジェクト域の正確な長さの計算

VARCHAR 値と LONG VARCHAR 値を含め、特性の値すべてに可能な最長の値を含めることができるように、オブジェクト域を定義することもできます。

この方法を使用してすべての特性について最大データ長を合計すると、オブジェクト域に関して定義した値が割り振られた記憶域に必ず収まるようになります。 VARCHAR 特性と LONG VARCHAR 特性については、最大長さ値の一部として 8 バイト長フィールドを含めてください。この計算のための式は次のとおりです。

```
オブジェクト域の長さ = 特性 1 の最大長 +
                       特性 2 の最大長 +
                       特性 3 の最大長 +
                        .
                        .
                        .
```

図 13. オブジェクト域の可能な最大長の計算

ただしこの方法を使用すると、特にいくつかの特性が最大長 32700 バイトの LONG VARCHAR フィールドである場合には、多くの記憶域を消費することになりますのでご注意ください。

すべての部分の合計

割り振る必要のある記憶域を判別するための式を図14 に示します。

```
構造のサイズ = FLG_HEADER_SIZE +
               定義域の長さ +
               オブジェクト域の長さ
```

図 14. 入力構造に必要な記憶域の計算

ヘッダー域の定義

入力構造は自己定義構造であるため、ヘッダー域にはその構造のサイズと形式を定義するいくつかの値が含まれます。これらの値を正しく定義するには、情報の集合全体と、作成する必要のある構造を考慮する必要があります。

ヘッダー域は 160 バイトです。各バイト位置には値を割り当てる必要があります。値を指定しない場合には、その位置をブランクとして定義しなければなりません。 1 バイトまたは複数バイトの位置をブランクとして定義するための方法として、 C 言語の `memset` 関数を使用して構造全体を FLG_BLANK またはゼロ文字だけにセットしたうえで、C 言語の `memcpy` 関数を使用して別の値にセットする必要のある情報だけをコピーするやり方があります。この方法を使用すると、DG2API.H で定義された定数が使用しやすくなります。これは、ブ

ランクまたはゼロの上書きだけを配慮する必要があり、値に埋め込みを行ってデータ長を合わせる必要がないためです。

ヘッダー域の各バイトの完全な仕様については、37ページの『ヘッダー域 (常に必須)』で説明されています。

各 API 呼び出しのためのヘッダー域の構文については、75ページの『第5章 情報カタログ・マネージャー API 呼び出しの構文』で説明します。

ヘッダー域の値のなかには特定の API 呼び出しでは必要のないものもありますが、図15 で示したバイト・オフセット位置が入るようにヘッダー域を定義する必要があります。

図 15. ヘッダー域

これらのバイト・オフセット位置については、38ページの表7 で説明されています。表10 には、ヘッダー域の定義に役立つ DG2API.H 内の定数がリストされています。

表 10. ヘッダー域のバイト・オフセット位置および DG2API.H で定義された役に立つ定数

バイト	内容	DG2API.H で定義された役に立つ定数	値
0-7	FLG-HEAD	FLG_H_IDENT	FLG-HEAD
8-15	定義域の長さ	FLG_DEFINITION_SIZE	160。1 つの定義域レコードの長さ。
16-23	オブジェクト域の長さ		
24-31	オブジェクト域の項目カウント		
32	区分	FLG_GROUPING_OBJ	G
		FLG_ELEMENTAL_OBJ	E
		FLG_CONTACT_OBJ	C
		FLG_DICTIONARY_OBJ	D
		FLG_PROGRAM_OBJ	P
		FLG_SUPPORT_OBJ	S
		FLG_ATTACHMENT_OBJ	A

表 10. ヘッダー域のバイト・オフセット位置および DG2API.H で定義された役に立つ定数 (続き)

バイト	内容	DG2API.H で定義された役に立つ定数	値
33-38	オブジェクト・タイプの ID		
39-159	予約域 (常にブランク)		

ヘッダー域を定義する場合、次の 3 つの値は定義域とオブジェクト域の内容によって異なります。

- 定義域の長さ (バイト 8-15)

  この値はおそらく、入力構造用の記憶域を割り振るためにすでに計算されていることと思われます。 この計算に関する説明を見直す場合には、47ページの『定義域の長さの計算』を参照してください。

- オブジェクト域の長さ (バイト 16-23)

  この値はおそらく、入力構造用の記憶域を割り振るためにすでに計算されていることと思われます。この計算に関する説明を見直す場合には、47ページの『オブジェクト域の長さの計算』を参照してください。

- オブジェクト域の項目カウント (バイト 23-31)

  FLGExport および FLGManageUsers を除き、入力構造を必要とするすべての API 呼び出しの場合、オブジェクト域の項目カウントは定義域内の特性の数に等しくなります。 FLGExport の場合には、オブジェクト域の項目カウントは、エクスポートするように指定されたオブジェクトの数の 5 倍になります。 FLGManageUsers の場合には、オブジェクト域の項目カウントは、追加または更新される各ユーザーについて 2 になります。

定義域の定義

定義域を定義するためには、入力構造内で API 呼び出しがどのような情報を必要とするのかが分かっている必要があります。

定義域の各レコードの長さは 160 バイトです。各バイト位置には値を割り当てる必要があります。値を指定しない場合には、その位置をブランクとして定義しなければなりません。 1 バイトまたは複数バイトの位置をブランクとして定義するための方法として、C 言語の `memset` 関数を使用して構造全体を FLG_BLANK にセットしたうえで、C 言語の `memcpy` 関数を使用して、別の値にセットする必要のある情報だけをコピーするやり方があります。この方法を使用すると、DG2API.H で定義された定数が使用しやすくなります。これは、

ブランクの上書きだけを配慮する必要があり、値に埋め込みを行ってデータ長を合わせる必要がないためです。 値のなかには特定の API 呼び出しでは必要のないものもありますが、定義域は図16 で示した 160 バイト全部を常に含まなければなりません。

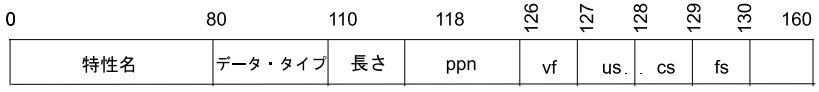


図 16. 定義域内のレコード

これらのバイト・オフセット位置については、40ページの表9 で説明されています。表11 には、定義域の定義に役立つ DG2API.H 内の定数がリストされています。

表 11. 定義域のバイト・オフセット位置および DG2API.H で定義された役に立つ定数

バイト	内容	DG2API.H 内の役に立つ変数	値
0-79	特性名		
80-109	データ・タイプ	FLG_DTYPE_CHAR	CHAR
		FLG_DTYPE_VARCHAR	VARCHAR
		FLG_DTYPE_LONGVARCHAR	LONG VARCHAR
		FLG_DTYPE_TIMESTAMP	TIMESTAMP
110-117	データ長		
118-125	特性省略名	FLG_PPN_OBJTYPID	OBJTYPID
		FLG_PPN_INSTIDNT	INSTIDNT
		FLG_PPN_INST_NAME	NAME
		FLG_PPN_UPDATIME	UPDATIME
		FLG_PPN_UPDATEBY	UPDATEBY
		FLG_PPN_EXTERNAL_NAME	NAME
		FLG_PPN_PTNAME	PTNAME
		FLG_PPN_DPNAME	DPNAME
		FLG_PPN_CREATOR	CREATOR
126	値フラグ	FLG_REQUIRED	R
		FLG_OPTIONAL	O
		FLG_SYSTEM	S
127	UUI 順序番号	FLG_UUI_1	1
		FLG_UUI_2	2
		FLG_UUI_3	3
		FLG_UUI_4	4
		FLG_UUI_5	5
		FLG_BLANK	

表 11. 定義域のバイト・オフセット位置および DG2API.H で定義された役に立つ定数 (続き)

バイト	内容	DG2API.H 内の役に立つ変数	値
128	大文字小文字の区別フラグ	FLG_YES FLG_NO	Y N
129	ファジー探索フラグ	FLG_YES FLG_NO	Y N
130-159	予約域 (常にブランク)		

定義域内のすべてのバイト位置の特定の意味に関する詳細については、39ページの『定義域 (常に必須)』を参照してください。使用している API 呼び出しの定義に関する詳細については、75ページの『第5章 情報カタログ・マネージャー API 呼び出しの構文』を参照してください。

オブジェクト域の定義

オブジェクト域における値の定義方法は、定義する各特性のデータ・タイプによって異なります。 CHAR 値と TIMESTAMP 値は固定長であるため、比較的簡単に定義することができますが、変数値 (VARCHAR および LONG VARCHAR) の定義はそれよりも複雑になります。

TIMESTAMP 値の長さと形式は固定されています。

CHAR 値は左寄せされ、後続ブランクが埋め込まれて、定義された長さに合わされます。たとえば、次の例のようになります。

```
'My example                    '
```

すべての値は文字データでなければなりません。値が数値の場合には、文字データに変換する必要があります。

いかなる値にも NULL 文字を使用することはできません。指定した値によって固定長すべてが埋まらない場合には、未記入の位置をブランクまたはゼロとして定義しなければなりません。未使用のバイト位置をブランクまたはゼロとして定義するための方法として、C 言語の `memset` 関数を使用して構造全体を FLG_BLANK またはゼロ文字 ('0' または 0x30) にセットしたうえで、C 言語の `memcpy` 関数を使用して、別の値にセットする必要のある情報だけをコピーすることができます。この方法を使用すると、DG2API.H で定義された定数が使用しやすくなります。これは、ブランクの上書きだけを配慮する必要があり、値に埋め込みを行ってデータ長を合わせる必要がないためです。

VARCHAR 値および LONG VARCHAR 値を指定する場合には、値の前に、その値の長さを指定するための 8 バイトを加えてください。たとえば、『Employee records -- Southwest Region』 という VARCHAR 値を指定するために必要な値は、次のようになります。

```
00000036Employee records -- Southwest Region
```

この値は VARCHAR 値であるため、後続ブランクで値を埋め込む必要はありません。

ヘッダー域、定義域、およびオブジェクト域の定義例

このセクションでは、DG2SAMP.C のうちの入力構造を定義する部分について説明します。

オブジェクト域の長さの計算

図17 に示すコードは、入力構造のオブジェクト域の長さを計算するものです。

```
//-------------------------------------------
// Build input structure for FLGSearch
//-------------------------------------------
printf ("Enter object instance name:¥n");
gets(pszObjInstName);                          1
ulInstValLen = strlen(pszObjInstName);         2
ulInstLen = (FLG_VARIABLE_DATA_LENGTH_LEN + ulInstValLen);   3
convertultoa(ulInstLen, pszLength);            4
```

図 17. DG2SAMP.C: オブジェクト域の長さの判別

図17 のコードは以下のステップによってオブジェクト域を判別します。

1. pszObjInstName をユーザーが指定したオブジェクト・インスタンス名にセットします。
2. オブジェクト・インスタンス名の長さを判別します。
3. 可変データ長フィールドの長さ (8) をオブジェクト・インスタンス名の長さに加算します。
4. オブジェクト域の長さの値を文字データに変換します。

ヘッダー域の定義

54ページの図18 のコードは、DG2SAMP.C が FLGSearch のための入力構造のヘッダー域をどのように定義するのかを示しています。このヘッダー域には、54ページの図19 と同じ値が入ります。

```
//-------------------------------------------
//  Header
//-------------------------------------------
memset(&(SearchStruct.srchHdr), FLG_BLANK, FLG_HEADER_SIZE);  1
memcpy(&SearchStruct.srchHdr.pchHIdent, FLG_H_IDENT, FLG_H_IDENT_LEN);              2
memcpy(&SearchStruct.srchHdr.pchHDefLength, "00000160", FLG_H_DEFAREA_LEN);         3
memcpy(&SearchStruct.srchHdr.pchHObjLength, pszLength , FLG_H_OBJAREA_LEN);         4
memcpy(&SearchStruct.srchHdr.pchHObjEntryCount, "00000001", FLG_H_OBJAREAENT_LEN); 5
```

図 18. DG2SAMP.C: ヘッダー域の定義

図18 に示すコードは、以下のステップに従って、C 言語を使用してヘッダー域を定義します。

1 ヘッダー域全体をブランクにセットします。

2 バイト 0-7 を識別子 (FLG_HEAD) にセットします。

3 定義の長さを 160 にセットします。

4 オブジェクト域の長さをセットします。この長さはこのプログラムの前の部分で計算されています。

5 オブジェクト域の項目カウントを 1 にセットします。

図19 は、図18 の C 言語コードによって定義された記憶域を示しています。

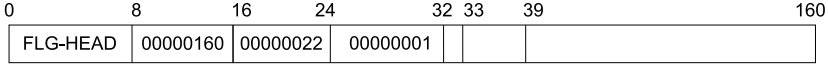


図 19. 定義されたヘッダー域 - SearchStruct.srchHdr

定義域の定義

図20 のコードは、DG2SAMP.C が FLGSearch のための入力構造の定義域をどのように定義するのかを示しています。この定義域には、55ページの図21 で示した値が入ります。

```
//-------------------------------------------
// Definition area
//-------------------------------------------
memset(&(SearchStruct.srchDef), FLG_BLANK, FLG_DEFINITION_SIZE);     1
memcpy(&SearchStruct.srchDef.pchDPropName,
"Name                                                    ",FLG_D_PROPNM_LEN); 2
memcpy(&SearchStruct.srchDef.pchDDataType, "VARCHAR                ", FLG_D_DATATYP_LEN);      3
memcpy(&SearchStruct.srchDef.pchDDataLength, "00000080", FLG_D_DATA_LEN);  4
memcpy(&SearchStruct.srchDef.pchDTagName, "NAME    ", FLG_D_PPN_LEN);       5
memset(SearchStruct.srchDef.pchDCS, 'N', FLG_D_CS_LEN);  6
memset(SearchStruct.srchDef.pchDFS, 'N', FLG_D_FS_LEN);  7
```

図 20. DG2SAMP.C: 定義域の定義

54ページの図20 に示すコードは、以下のステップに従って、C 言語を使用してレコード定義域を定義します。

1. 定義レコード全体をブランクにセットします。
2. 特性名を名前にセットします。
3. データ・タイプを VARCHAR にセットします。
4. データ長を 80 にセットします。
5. 特性省略名を NAME にセットします。
6. 大文字小文字の区別フラグを N にセットします。
7. ファジー探索フラグを N にセットします。

図21 は、54ページの図20 の C 言語コードによって定義された記憶域を示しています。

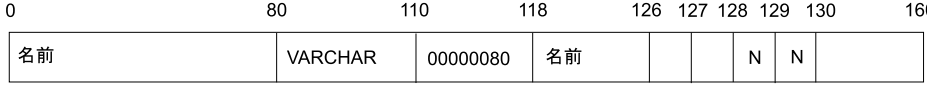


図 21. 定義された定義域 - SearchStruct.srchDef

オブジェクト域の定義

図22 は、DG2SAMP.C が FLGSearch のための入力構造のオブジェクト域をどのように定義するのかを示しています。このオブジェクト域には、56ページの図23 に示す値が入ります。

```
//-------------------------------------------
// Object area
//-------------------------------------------
memset(&(SearchStruct.Item), FLG_BLANK, FLG_INST_NAME_LEN + FLG_VARIABLE_DATA_LENGTH_LEN); 1
convertultoa(ulInstValLen, pszNameLength);                                                 2
pszInstanceName=strncat(pszNameLength,pszObjInstName,ulInstValLen);                        3
memcpy(&SearchStruct.Item.Name, pszInstanceName, ulInstLen);                               4
```

図 22. DG2SAMP.C: オブジェクト域の定義

図22 に示すコードは、以下のステップに従って、C 言語を使用してオブジェクト域を定義します。

1. オブジェクト域をブランクにセットします。
2. 名前値の長さを文字データに変換します。
3. VARCHAR 値の長さをその値に連結します。

4 オブジェクト域に値の長さと値を入れます。

図23 は、55ページの図22 の C 言語コードによって定義された記憶域を示しています。

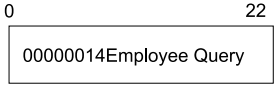


図 23. 定義されたオブジェクト域 - SearchStruct.Item

情報カタログ・マネージャーの API 出力構造

57ページの図24 は、情報カタログ・マネージャーの API 出力構造の一般的な形式を示しています。この出力構造は、ヘッダー域、定義域、およびオブジェクト域の 3 つの連続した区域からなります。情報カタログ・マネージャー API 呼び出しによっては (FLGGetType のように)、最初の 2 つの区域だけしか作成されないこともあります。

ユーザーのプログラムが出力構造を作成する API 呼び出しを呼び出すと、NULL ポインターを指すポインターがパラメーターとして渡されます。そして、その API 呼び出しが出力構造のアドレスを NULL ポインターに割り当てます。

複数の API 呼び出しを呼び出すことによってメモリーを使い切ることのないように、ユーザーのプログラムで情報カタログ・マネージャーの API 呼び出し FLGFreeMem を使用して、この出力構造に割り振られたメモリーを割り振り解除することができます。 FLGFreeMem の詳細については、137ページの『FLGFreeMem』を参照してください。

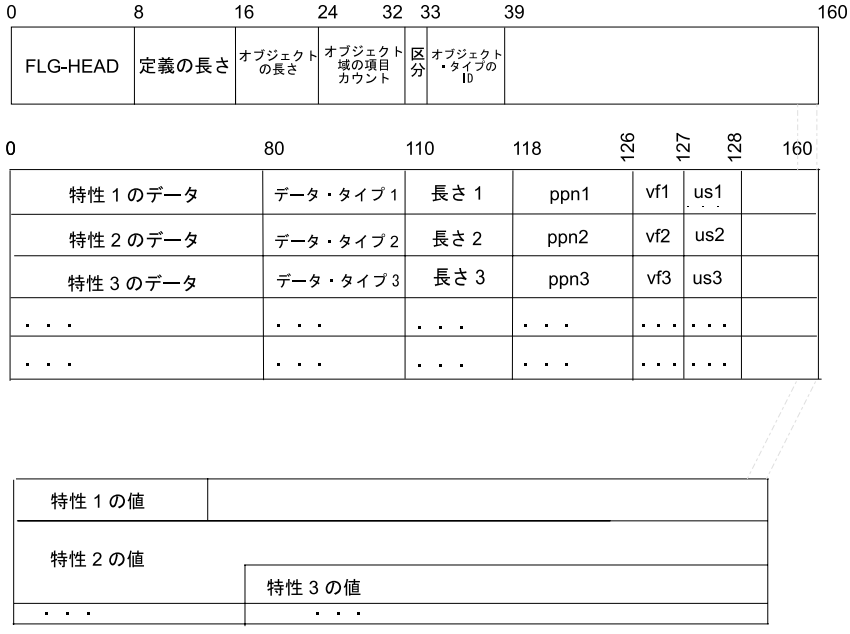


図 24. API 出力構造

以下の API 呼び出しでは、データを戻すために出力構造が作成されます。

FLGDeleteTree
削除されたオブジェクト・インスタンスのリストを戻します。

FLGFoundIn 特定のインスタンスが検索されたインスタンスまたはオブジェクト・タイプのリストを戻します。

FLGGetInst オブジェクト・インスタンスに関する情報を獲得します。

FLGGetReg オブジェクト・タイプ登録に関する情報を獲得します。

FLGGetType オブジェクト・タイプに関する情報を獲得します。

FLGInit 必要な資源を割り振り、情報カタログ・マネージャー・クライアントを初期設定します。

FLGListAnchors
他のオブジェクトに含まれていない Grouping オブジェクトのインスタンスのリストを戻します。このような最上位の Grouping オブジェクトは、アンカー と呼ばれます。

FLGListAssociates
特定のインスタンスまたはオブジェクト・タイプと関連するインスタンスのリストを戻します。

FLGListContacts
指定したインスタンスのためにすべての Contact オブジェクト・インスタンスのリストを戻します。

FLGListObjTypes
すべてのオブジェクト・タイプのリストを戻します。

FLGListOrphans
現在他のインスタンスとは関連していない、特定のオブジェクト・タイプのインスタンスのリストを戻します。

FLGListPrograms
すべての Program オブジェクトのリストを戻します。

FLGManageCommentStatus
コメント用に利用可能な選択項目のリストを更新します。

FLGManageUsers
情報カタログの管理者とユーザーを更新し、各ユーザーの権限のエクステントを識別します。

FLGNavigate 指定された Grouping オブジェクトに含まれる Grouping または Elemental オブジェクトのリストを戻します。

FLGSearch 特定のオブジェクト・タイプの、選択基準を満たすインスタンスのリストを戻します。

FLGSearchAll 任意のオブジェクト・タイプの、選択基準を満たすインスタンスのリストを戻します。

FLGWhereUsed
指定されたオブジェクトを含む Grouping オブジェクトのリストを戻します。

ヘッダー域 (常に存在)

ヘッダー域では、定義域とオブジェクト域の情報を記述します。ヘッダー域のバイト・オフセット位置は、図25 および 59ページの表12 で表示、説明しています。

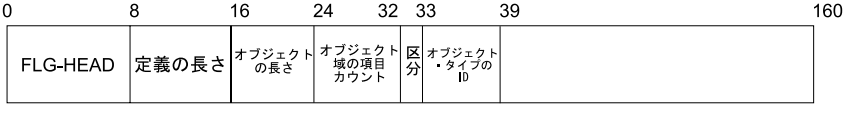


図 25. 出力構造のヘッダー域

表 12. 出力構造のヘッダー域とそのフィールド

58ページの図 25 で示したセクション	バイト・オフセット	存在するかどうか	説明
FLG-HEAD	0-7	常に存在	構造の識別子。
定義の長さ	8-15	常に存在	定義域の長さ。 値は 160 の倍数 (特性の数に各定義レコードの長さを掛けた値)。
オブジェクトの長さ	16-23	常に存在	オブジェクト域の長さ。 データが戻されない場合には、このオブジェクト域の長さはゼロ (00000000) になる。
オブジェクト域の項目カウント	24-31	常に存在	オブジェクト域に入力された各特性値の数。 この値は、定義域内の特性の数にオブジェクト域で記述された値のセットの数を掛けた値になる。 データが戻されない場合には、このオブジェクト域の長さはゼロ (00000000) になる。
区分	32	以下の呼び出しで存在 • FLGGetInst • FLGGetReg • FLGGetType	オブジェクト・タイプまたはオブジェクトの区分 有効な値は以下のとおり。 **G** Grouping **E** Elemental **C** Contact **P** Program **D** Dictionary **S** Support **A** Attachment

表 12. 出力構造のヘッダー域とそのフィールド (続き)

58ページの図 25 で示したセクション	バイト・オフセット	存在するかどうか	説明
オブジェクト・タイプの ID	33-38	以下の呼び出しで存在 • FLGGetInst • FLGGetReg • FLGGetType	オブジェクト・タイプに関するシステム生成された識別子。
	39-159	常に存在	ブランクのままにする必要がある。

定義域 (常に存在)

定義域には、特定の情報カタログ・マネージャー API 機能によって出力値として作成された特性定義の集合が入ります。

表13 は、出力構造を作成する API 呼び出しのための定義域の意味を示しています。

表 13. 各種の API 呼び出しにおける定義域の意味

API 呼び出し	定義域で示される情報
FLGGetInst FLGGetReg FLGGetType FLGDeleteTree	オブジェクト登録、オブジェクト・タイプ、またはオブジェクト・インスタンスを定義する特性の集合の定義。
FLGInit	情報カタログ・マネージャー環境に関する情報
FLGFoundIn FLGListAnchors FLGListAssociates FLGListContacts FLGListObjTypes FLGListOrphans FLGListPrograms FLGManage-CommentStatus FLGManageUsers FLGNavigate FLGSearch FLGSearchAll FLGWhereUsed	これらのいずれかの API 呼び出しによって戻された各項目を記述する特性の集合の定義。

図26 は、この定義域内のレコードのバイト・オフセット位置を示しています。

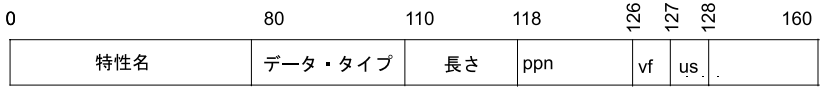


図 26. 定義域内のレコード

特性の集合における各特性は、表14 に示すように、形式設定された仕様の集合によって定義されます。

表 14. 出力構造の定義域とそのフィールド

図26 で示したセクション	バイト・オフセット	存在するかどうか	説明
特性名	0-79	常に存在	外部名。
データ・タイプ	80-109	常に存在	特性のデータ・タイプ。 有効な値は以下のとおり。 **CHAR** 固定長文字データ。最大長は 254。 **VARCHAR** 可変長文字データ。最大長は 4000。 **LONG VARCHAR** 可変長文字データ。最大長は 32700。 **TIMESTAMP** yyyy-mm-dd-hh.mm.ss.nnnnnn 形式のタイム・スタンプ。タイム・スタンプの長さは 26。
長さ	110-117	常に存在	オブジェクト域内の特性値の最大長。

表 14. 出力構造の定義域とそのフィールド (続き)

61ページの図 26 で示したセクション	バイト・オフセット	存在するかどうか	説明
ppn	118-125	以下の呼び出しで存在 • FLGGetInst • FLGGetReg • FLGGetType • FLGManage-CommentStatus その他の API 呼び出しの場合、このフィールドは未使用のためブランクのまま。	特性省略名
vf	126	以下の呼び出しで存在 • FLGGetInst • FLGGetReg • FLGGetType その他の API 呼び出しの場合、このフィールドは未使用のためブランクのまま。	特性が必須であるのか、任意選択であるのか、あるいはシステム生成されるのかを指定する値フラグ。 有効な値は以下のとおり。 **R** 必須 **O** 任意選択 **S** システム生成

表 14. 出力構造の定義域とそのフィールド (続き)

61ページの図 26 で示したセクション	バイト・オフセット	存在するかどうか	説明
us	127	以下の API 呼び出しで存在 • FLGGetInst • FLGGetType その他の API 呼び出しの場合、このフィールドは未使用のためブランクのまま。	ある特性が汎用固有識別コード (UUI) のパートであることを示す、UUI の順序番号。 有効な値は以下のとおり。 **1** UUI のパート 1 **2** UUI のパート 2 **3** UUI のパート 3 **4** UUI のパート 4 **5** UUI のパート 5 **(ブランク)** UUI の一部ではない UUI のパートの詳細については、*情報カタログ・マネージャー 管理の手引き* を参照。
	128-159	常に存在	予約セクション。 ブランクのまま。

オブジェクト域 (値を検索するときに存在)

オブジェクト域には、定義域で定義された特性の値が入ります。これらの値は、定義域で定義したとおりの順序で表示されます。

オブジェクト域は、以下の API 呼び出しの出力構造に含まれます。

FLGDeleteTree

FLGFoundIn

FLGGetInst

FLGGetReg

FLGInit

FLGListAnchors

FLGListAssociates

FLGListContacts

FLGListObjTypes

FLGListOrphans

FLGListPrograms

FLGManageCommentStatus

FLGManageUsers

FLGNavigate

FLGSearch

FLGSearchAll

FLGWhereUsed

それぞれの値のサイズは、以下の規則に従って判別することができます。

データ・タイプ	値のサイズに関する規則
VARCHAR	値の前に、その値の実際の長さを示す 8 文字長のフィールドが付きます。
LONG VARCHAR	値の前に、その値の実際の長さを示す 8 文字長のフィールドが付きます。
CHAR	値は定義域の特性の長さフィールドで定義されたバイト数を占め、定義された長さになるように右側に埋め込みが行われます。
TIMESTAMP	26 バイトです。

API 呼び出しによって得られた出力構造の読み取り

情報を戻す情報カタログ・マネージャー API 呼び出しは、その情報を出力構造に収めます。

出力構造を読むためには、構造の各部分が他の部分の意味を定義しているため、その構造全体を 1 つのものとしてとらえてください。

オブジェクト・インスタンスのリストを戻す API 呼び出しの場合、オブジェクト域には、各特性ごとに複数の値が含まれている可能性があります。このオブジェクト域には、定義域で定義された特性をマップするための値の集合が 2 つ以上含まれることがあります。

出力構造を読むためのポインターの使い方

出力構造内の値を読むには、API 呼び出しで戻されたポインターを使用して、その構造を指す 2 つ以上のポインターを定義してください。

ユーザーのプログラムは、出力構造を作成する API 呼び出しを出す場合、NULL ポインターのアドレスを含むポインターを定義して、この定義済みポインターをパラメーターとして API 呼び出しに渡す必要があります。 API 機能はこの NULL ポインターにその出力構造のアドレスを割り当てます。

ユーザーは、この構造のヘッダー域と定義域を順に処理する 2 番目のポインターと、オブジェクト域を順に処理する 3 番目のポインターも定義する必要があります。

図27 で示す pOutStruct は、出力構造の先頭を指すポインターです。そのあとで、ヘッダー域と定義域を順に読み取るための pReadStruct を定義し、オブジェクト域を順に読み取る pObjArea を定義することができます。

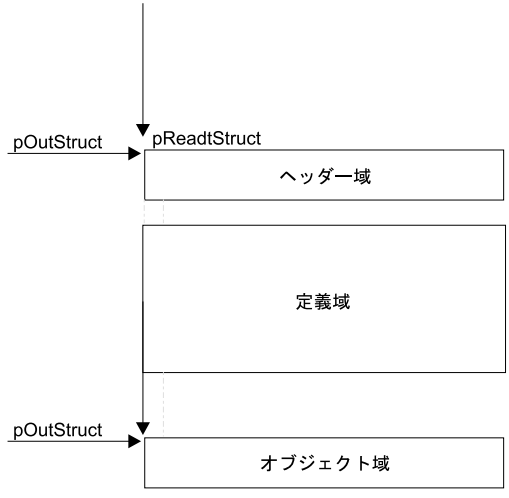


図 27. 出力構造を順に読み取るポインターの定義

ユーザーの必要に応じて、構造の値を戻されたとおりの順で読み取ることも、特定の値を探索することもできます。いずれの場合にも、次のことを行う必要があります。

1. 戻される特性の数とオブジェクトの数を計算する。
2. 各特性のデータ・タイプとデータ長を検出する。
3. オブジェクト域を順にたどって、値の読み取りまたは検出を行う。

DG2API.H を使用して値を読み取る

情報カタログ・マネージャーでは、出力構造を読むために必要な多くの値の長さと有効な値を定義する DG2API.H というヘッダー・ファイルが提供されています。これらの長さを使用して、ヘッダー域、定義域、およびオブジェクト域を順に読むために必要な C 言語コードを作成することができます。

DG2API.H ファイルで定義されている内容の完全なリストについては、 267ページの『付録B. 情報カタログ・マネージャー API ヘッダー・ファイル － DG2APIH』を参照してください。

出力構造内の特性の数を計算する

API 呼び出しのなかには不特定数の特性を戻すものがあるため、その数を計算することが必要になります。

API 呼び出しによって戻されたポインター・アドレスを使用して、出力構造の先頭を指すようにポインターをセットしてください。

定義域内の特性の数を計算するために、ヘッダー域の定義の長さ域 (バイト 8-15) の数値を定義域の個別レコードの長さ (160) で割ってください。この計算を行うためには、定義の長さの文字列を整数値に変換する必要があります。

DG2API.H では、この計算を行いやすくするために、変数 FLG_DEFINITION_SIZE が提供されています。

```
特性の数 = 定義の長さの整数値 / FLG_DEFINITION_SIZE
```

図 28. 特性の数の計算

戻された値の集合の数を計算する

出力構造に戻された値の集合の数を計算するためには、67ページの図30 に示すように、図29 に示すオブジェクト域の項目カウントを構造内の特性の数で割ってください。

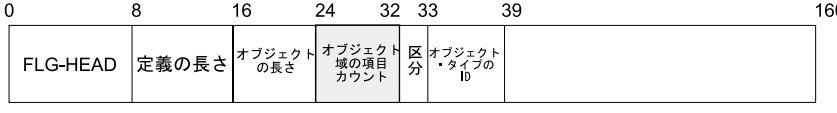


図 29. ヘッダー域におけるオブジェクト域の項目カウント

```
値の集合の数 = オブジェクト域の項目カウント / 特性の数
```

図 30. 値の集合の数の計算

ヘッダー域のフィールドは文字形式なので、図30 の計算で使用されるときには、数値形式に変換されなければなりません。図30 の計算にたどりつくためには、DG2API.H で定義された構造を使用することができます。

定義域の特性のデータ・タイプと長さを読み取る

特性のデータ・タイプと長さを読み取るためには、ポインターを定義して、定義域の正しい値を読むためのポインター算術計算を行ってください。 図31 では最初の特性のデータ・タイプと長さの位置が強調表示されています。

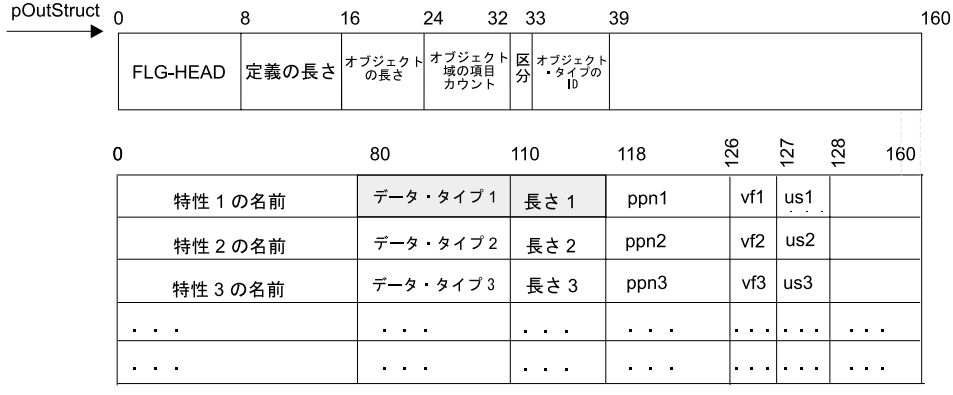


図 31. 最初の特性のデータ・タイプとデータ長

定義域の最初の特性のデータ・タイプを読むには、図32 に示すように、最初の定義レコードのヘッダー域および特性名のフィールドの長さを、出力構造を指すフィールドの位置に加算してください。

```
pLocationOfDataType = pOutStruct +
                      FLG_HEADER_SIZE +
                      FLG_D_PROPNM_LEN
```

図 32. データ・タイプ値の位置の計算

pOutStruct は出力構造を指すポインターを表し、FLG_HEADER_SIZE はヘッダー域の長さを表し、FLG_D_PROPNM_LEN は特性名フィールドの長さを表します。これで、この位置にある値を別の変数に保管できるようになります。

定義域の最初の特性のデータ長を読むには、図33 に示すように、計算によって得られたポインターにデータ・タイプ・フィールドの長さを加算して、データ・タイプの位置を求めてください。

```
pLocationOfDataLen = pLocationOfDataType +
                     FLG_D_DATATYP_LEN
```

図 33. データ長値の位置の計算

pLocationOfDataType は定義レコード内のデータ・タイプ・フィールドを指すポインターであり、FLG_D_DATATYP_LEN はデータ・タイプ・フィールドの長さです。

その他の特性のデータ・タイプと長さを読むには、さらにオフセット値を加算します。次の特性のデータ・タイプ・フィールドを求めるには、図34 に示すように、現在の特性のデータ・タイプを指すポインターに、1 つのデータ・レコード全体の長さ (160) を加えてください。

```
pLocationOfDataType = pLocationOfDataType + FLG_DEFINITION_SIZE
```

図 34. 次のデータ長値の位置の計算

FLG_DEFINITION_SIZE は 160 バイトです。

オブジェクト域から順に値を読み取る

オブジェクト域の値を読むには、ポインター算術計算によってその位置を求める必要があります。位置を正しく計算するためには、特性のデータ・タイプと長さが分からなければなりません。

1. 図35 に示すように、そのオブジェクト域の先頭を指すようにポインターに増分を加えて、オブジェクト域の最初の値を読みます。

```
pObjArea = pOutStructure + FLG_HEADER_SIZE +
           (FLG_DEFINITION_SIZE × number_of_properties)
```

図 35. オブジェクト域の先頭へのポインターの移動

   FLG_HEADER_SIZE はヘッダー域の長さであり、FLG_DEFINITION_SIZE は定義域内の 1 つのレコードの長さです。

2. 定義域内でこの値が属している特性のデータ・タイプとデータ長を調べます。

CHAR または TIMESTAMP の場合

定義域で指定された長さ値を読みます。

VARCHAR または LONG VARCHAR の場合

a. 値の長さを判別するために、この値に関する最初の 8 文字を読みます。

b. ポインターを 8 バイト移動させて、値自体を読みます。

図36 に示すように、現行値の実際の長さをポインターに加算して、オブジェクト域内の次の値に移動します。

```
pObjValue = pObjArea + 値の実際の長さ
```

図 36. 次の値へのポインターの移動

図37 は、オブジェクト域の先頭から開始して、VARCHAR 値の長さを読み、ポインターをその値自体の先頭に移動させ、その値を読み、さらに次の値に進む方法を示しています。

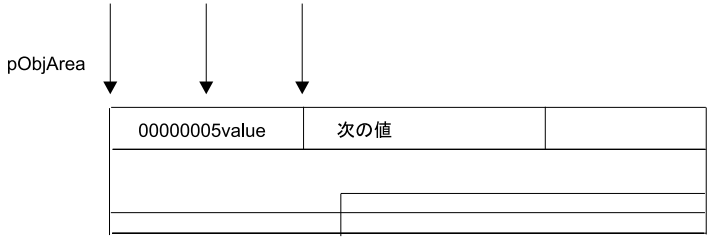


図 37. オブジェクト域の VARCHAR 値の読み取り

DG2SAMP.C により出力構造内の値を見つける例

DG2SAMP.C プログラムは、ユーザーからオブジェクト・タイプ名を入手してから FLGListObjTypes 呼び出しを出して、情報カタログ・データベースで使用可能なオブジェクト・タイプのリストを検索します。このプログラムは、ユーザーが指定したオブジェクト・タイプの外部名と、FLGListObjTypes が出力構造に入れて戻した名前を突き合わせます。

70ページの図38 は、FLGListObjTypes の API 呼び出しによって作成される出力構造の形式を示しています。

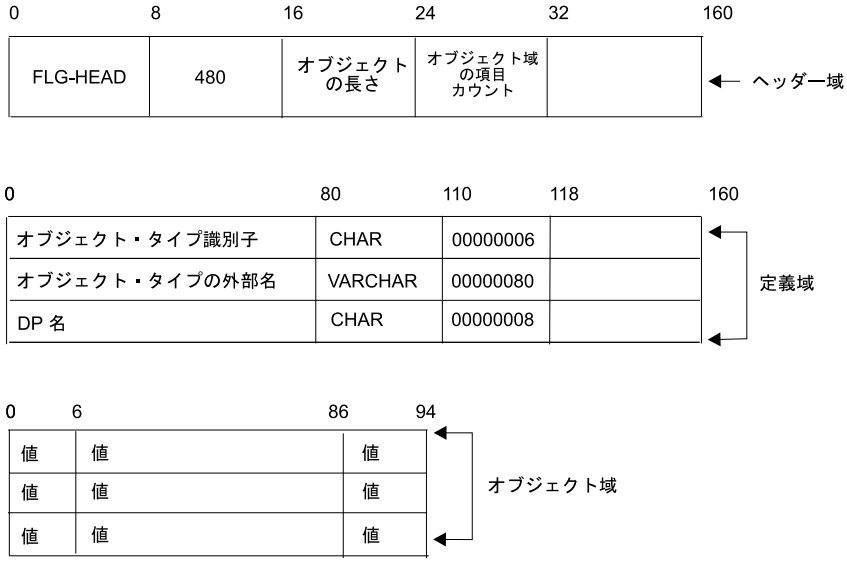


図 38. FLGListObjTypes 出力構造

ユーザーおよび出力構造からの値の獲得

図39 は、このプログラムがユーザーによって指定された値を読み取ってその長さを計算するようすを示しています。この図は、プログラムが出力構造内の値を NULL 文字で終了するストリングにコピーし、オブジェクト域内の値の集合の数を計算する方法も示しています。

```
gets(pszObjName);      1
ulTypeLen = strlen(pszObjName);      2
memcpy(&pszObjEntryCount, pListStruct->pchHObjEntryCount, FLG_H_OBJAREAENT_LEN);  3
memcpy(&pszDefLength, pListStruct->pchHDefLength, FLG_H_DEFAREA_LEN);              4
ulCount = (atoi(pszObjEntryCount) / (atoi(pszDefLength) / FLG_DEFINITION_SIZE));   5
```

図 39. DG2SAMP.C: ユーザーからの値の獲得

図39 に示したコードは、以下のステップを実行します。

1 ユーザーからオブジェクト・タイプ名を入力として獲得します。

2 オブジェクト・タイプ名の長さを判別します。

3 オブジェクト項目カウントを NULL 文字で終了するストリングにコピーします。

4 定義の長さを NULL 文字で終了するストリングにコピーします。

5 オブジェクト域内の値の集合の数を計算します。

オブジェクト域の先頭へのポインターの割り当て

図40 に示すコードは、オブジェクト域の先頭にポインターを割り当てます。

この例では、FLGListObjTypes の出力は常に 3 つの同じ特性を持つため、プログラムは特性の数、データ・タイプ、およびデータ長を判別する必要がありません。

```
ulPosition = 0;                                                    1
pCurrPos = ((UCHAR *)pListStruct + FLG_HEADER_SIZE + ulDefLen);    2
```

図 40. DG2SAMP.C: オブジェクト域の先頭へのポインターの割り当て

図40 に示したコードは、以下のステップを実行します。

1 位置カウンターを 0 にセットします。

2 出力構造 (pListStruct) の先を指すポインターに、ヘッダー域と定義域の長さを加算して、オブジェクト域の先頭にポインターを位置付けます。

オブジェクト域内での移動

72ページの図41 に示すコードは、オブジェクト域内でポインターを移動させ、ユーザーが指定した名前に一致するオブジェクト・タイプ名の検出を試みます。

```
while (fNotFound && (ulPosition < ulCount))     1
{
  ulPosition = (ulPosition + 1);
  memcpy(&pszObjTypeId, (void *) pCurrPos, FLG_H_OBJTYPID_LEN);        2
  pCurrPos = pCurrPos + FLG_H_OBJTYPID_LEN;                            3
  memcpy(&pszLength, (void *)pCurrPos, FLG_VARIABLE_DATA_LENGTH_LEN); 4
  ulLength = atoi(pszLength);                                          5
  pCurrPos = pCurrPos + FLG_VARIABLE_DATA_LENGTH_LEN;                  6
  strncpy (pszObjectName, (void *)pCurrPos, ulLength);                 7
  pszObjectName[ulLength] = NULLCHAR;                          8
  if (!(strcmp(pszObjName, pszObjectName)))    9
    {
      fNotFound = FALSE;
      printf ("The object type ID for %s is %s.¥n¥n", pszObjName, pszObjTypeId);
    }
  else
    { // Move temporary pointer to the next object
      pCurrPos = pCurrPos + ulLength + FLG_DPNAME_LEN;   10
    }
}
```

図 41. DG2SAMP.C: オブジェクト・タイプ名とオブジェクト域内のオブジェクト・タイプ名の突き合わせ

図41 に示したコードは、以下のステップを実行します。

1 プログラムが一致するオブジェクト名をまだ検出していないこと、およびポインターがオブジェクト域の最後の値の集合に達していないことを確認します。

2 最初のオブジェクト・タイプのオブジェクト・タイプ ID を pszObjTypeId にコピーします。

3 次の値 (オブジェクト・タイプ名の値) にポインターを移動させます。

4 オブジェクト・タイプ名値の最初の 8 バイト (この VARCHAR 値の長さが入っています) をコピーします。

5 この長さを整数データに変換します。

6 変数データ長を超えてオブジェクト・タイプ名の先頭までポインターを移動させます。

7 そのポインター位置のオブジェクト・タイプ名を pszObjectName にコピーします。

8 オブジェクト・タイプ名の終わりに NULL 文字を追加して、この値を NULL 文字で終了するストリングにします。

9 pszObjectName とユーザーが指定したオブジェクト・タイプ名を比較します。

10 pszObjectName の値とユーザーが指定したオブジェクト・タイプ名が一致しない場合には、次の値の集合の先頭にカーソルを移動させます。

第5章 情報カタログ・マネージャー API 呼び出しの構文

情報カタログ・マネージャーで提供される API 呼び出しを使用すると、ユーザー独自のアプリケーションで情報カタログ・マネージャー機能を使用することができます。

各 API 呼び出しは英字順に並んでいます。説明には、各 API 呼び出しに関する入力用のパラメーターおよび構造と、出力用のパラメーターおよび構造が含まれています。

各 API 呼び出しの説明には、以下の情報のうちの該当するものが含まれています。

- 入力パラメーター
- 入力構造
- 出力パラメーター
- 出力構造

API 呼び出しの構文規則

情報カタログ・マネージャー API 呼び出しを使用する際には、特定の構文規則に従う必要があります。

構文図の読み方

このセクションで示す構文図は、C 言語の関数原型の形式で書かれています。

これらの関数原型は DG2API.H ヘッダー・ファイルで定義されているため、(`#include` ステートメントを使用して) このファイルをプログラムに組み込むことにより、関数原型を指定するために独自のコードを書く必要がなくなります。 267ページの『付録B. 情報カタログ・マネージャー API ヘッダー・ファイル − DG2APIH』には、DG2API.H ファイルで定義されているデータ・タイプ、関数原型、および定数が列挙されています。

理由コードは、APIRET データ・タイプとして戻されます。 APIRET は、DG2API.H ヘッダー・ファイルで符号なし長整数データ・タイプとして定義されます。

理由コードおよび拡張コードは、285ページの『付録D. 情報カタログ・マネージャーの理由コード』に列挙されています。

DG2API.H で定義された定数をプログラムで使用する方法

DG2API.H ヘッダー・ファイルには、情報カタログ・マネージャー API 呼び出し用の構造体、型定義、および一般的に使用される値が含まれています。このファイルには、情報カタログ・マネージャー API 呼び出し用の関数原型も含まれています。これらの定数を使用することにより、C 言語プログラムが作成しやすくなります。情報カタログ・マネージャー API ヘッダー・ファイルで定義されている定数のリストについては、 267ページの『付録B. 情報カタログ・マネージャー API ヘッダー・ファイル － DG2APIH』を参照してください。

FLGAppendType

既存のオブジェクト・タイプに任意選択の特性を追加するために使用します。

Comments オブジェクト・タイプは拡張できませんが、 Comments オブジェクト・タイプ以外のオブジェクト・タイプはどれでも追加することができます。

許可

管理者

構文

```
APIRET  APIENTRY    FLGAppendType( PFLGHEADERAREA   pObjTypeStruct,
                                   PFLGEXTCODE      pExtCode );
```

パラメーター

pObjTypeStruct (PFLGHEADERAREA) － 入力
このオブジェクト・タイプに追加する 1 つまたは複数の特性に関する仕様を含む入力構造を指します。

pExtCode (PFLGEXTCODE) － 出力
理由コードに関連する拡張コードを指します。戻された理由コードに関連した意味のある拡張コードがあるかどうかについては、285ページの『付録D. 情報カタログ・マネージャーの理由コード』を参照してください。

理由コード (APIRET)
この API 呼び出しの実行結果を表します。

戻された理由コードの説明については、285ページの『付録D. 情報カタログ・マネージャーの理由コード』を参照してください。

入力構造

FLGAppendType を使用するには、78ページの図42 に示す入力構造を定義しなければなりません。この構造はヘッダー域と定義域だけからなります。

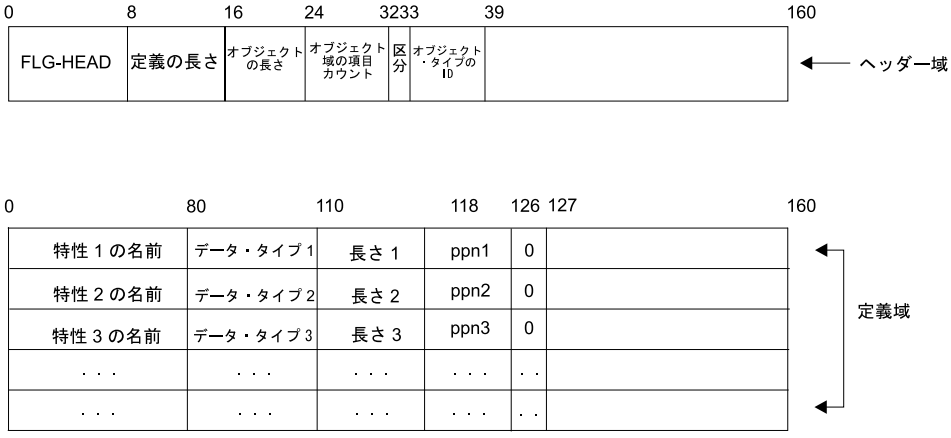


図 42. FLGAppendType 入力構造

バイト・オフセットの意味については、36ページの『情報カタログ・マネージャー API 入力構造』を参照してください。

使用方法

制約事項

Comments オブジェクト・タイプは拡張できませんが、 Comments オブジェクト・タイプ以外のオブジェクト・タイプはどれでも追加することができます。

オブジェクト・タイプ内に既に存在する新規の特性を追加すると、『新規』特性は、複写として扱われ、FLGAppendType は警告 (FLG_WRN_PROPDUP) をもって正常に完了します。以下のすべてが既存の特性と一致する場合、特性はコピーです。

データ・タイプ

データ長

特性省略名

値フラグ

UUI 番号

入力要件

ヘッダー域

- バイト 33 から 38 までに指定するオブジェクト・タイプ識別子は、カタログに存在しているものでなければなりません。
- バイト 32 に指定する区分は、バイト 33 から 38 までに指定した既存のオブジェクト・タイプと一致していなければなりません。

定義域

- 入力構造には、このオブジェクト・タイプに関してすでに定義されている特性を含めることはできず、追加される新規の特性だけを含めることができます。
- 追加される特性は、任意選択のものでなければなりません。バイト 126 の値フラグ・フィールドに文字 O を指定してください。
- 追加される特性は、汎用固有識別コード (UUI) のパートとして定義することはできません 。したがって、バイト 127 のフィールドはブランクとして定義してください。
- 新規の特性名は、そのオブジェクト・タイプの中では固有のものでなければなりません。
- 新規の特性省略名は、そのオブジェクト・タイプの中では固有のものでなければなりません。特性省略名は以下の規則に従っている必要があります。
  - 1 バイト文字セット (SBCS) の文字だけしか使用することができません。
  - 先頭文字は、A から Z または a から z まで、@ または #、あるいは $ でなければなりません。
  - 先頭以外の文字には、A から Z または a から z まで、0 から 9 まで、 @ または #、$ または _ (下線) を使用することができます。
  - 先行または組み込みブランクを使用することはできません。
  - 現行データベースに関する SQL 予約語を名前として使用することはできません。予約語のリストについては、使用するデータベースに関する資料を参照してください。
- 1 つのオブジェクト・タイプに関するすべての特性の合計長が環境限界を超えてはなりません。この制限は、それぞれのデータベース・システムの行 (オーバーヘッドを含む) の最大制限値によって異なります。詳細は、それぞれのデータベース・システムの DB2 UDB SQL 解説書を参照してください。

一般的な規則

- 1 つのオブジェクト・タイプの特性の最大長は、 255 (FLG_MAX_PROPERTIES) です。
- 1 つのオブジェクト・タイプに関する特性のうちで LONG VARCHAR データ・タイプにすることができるものの最大数は、 14 (FLG_MAX_NUM_LONG_VARCHARS) です。

情報カタログ更新の制御

プログラムを可能な限り、情報カタログと同期化させるために、FLGAppendType が正常に完了したあとで、呼び出しを FLGCommit (82ページの『FLGCommit』参照) に組み込む必要があります。FLGAppendType が正常に完了しない場合は、呼び出しを FLGRollback (225ページの『FLGRollback』参照) に組み込む必要があります。

例

図43 は、FLGAppendType 呼び出しを出すために必要な C 言語コードを示しています。このコードは、オブジェクト・タイプ識別子 000044 のオブジェクト・タイプに Density という名前特性を追加するものです。

```
APIRET           rc;              // Declare reason code
PFLGHEADERAREA   pObjTypeStruct;  // Pointer to the input structure
FLGEXTCODE       ExtCode = 0;          // Declare extended code
  .
  .  /* Appending pObjTypeStruct object Type */
  .          /*  by providing object properties     */
  .
rc = FLGAppendType (pObjTypeStruct,
                    &ExtCode);             // Pass pointer to extended code
```

図 43. C 言語による FLGAppendType の呼び出しのサンプル

図44 は、この FLGAppendType 呼び出しのための入力構造を示しています。pObjTypeStruct パラメーターはこの入力構造を指しています。

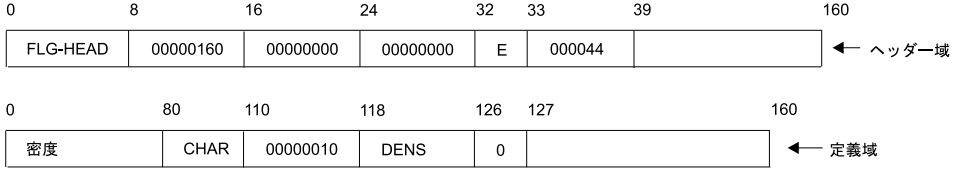


図 44. FLGAppendType のための入力構造のサンプル

特別なエラー処理

FLGAppendType でデータベース・エラーが発生すると、情報カタログ・マネージャーは、プログラム内で最後に行われたコミット点にデータベースをロールバックします。

このロールバックが正常に行われた場合、FLGAppendType は理由コード FLG_SEVERR_DB_AUTO_ROLLBACK_COMPLETE を戻します。拡張コードには、情報カタログ・マネージャーがデータベースのロールバックを行う原因となったデータベース・エラーに関する SQL コードが含まれています。

考慮事項: このロールバックが失敗した場合には、FLGAppendType は理由コード FLG_SEVERR_DB_AUTO_ROLLBACK_FAIL を戻します。拡張コードには、情報カタログ・マネージャーがデータベースのロールバックを行う原因となったデータベース・エラーに関する SQL コードが含まれています。この場合、データベースの保全性に関する重大な問題が発生している可能性があるため、プログラムで FLGTerm を呼び出して情報カタログ・マネージャーを終了させる必要があります。

データベースの状態によっては、バックアップ用データベース・ファイルを使用してデータベースを回復させなければならないことがあります。 情報カタログ・データベースの回復に関する詳細については、*情報カタログ・マネージャー 管理の手引き* を参照してください。

FLGAppendType エラーとは無関係なコミットされていない変更内容が情報カタログ・マネージャーによって除去されないようにするために、プログラム内のこの呼び出しの直前に FLGCommit 呼び出しを組み込んでください。

FLGCommit

作業単位の開始以降または最後のコミット点以降に情報カタログに対して行われたすべての変更内容をコミットするために使用します。

許可

管理者またはユーザー

構文

```
APIRET APIENTRY FLGCommit (PFLGEXTCODE   pExtCode)
```

パラメーター

pExtCode (PFLGEXTCODE) – 出力
理由コードに関連する拡張コードを指します。戻された理由コードに関連した意味のある拡張コードがあるかどうかについては、285ページの『付録D. 情報カタログ・マネージャーの理由コード』を参照してください。

APIRET
この API 呼び出しの実行結果を表します。

戻された理由コードの説明については、285ページの『付録D. 情報カタログ・マネージャーの理由コード』を参照してください。

使用方法

情報カタログ・データベースに対して行われた変更内容を永久的なものにするには、プログラムの中で、変更を行ったあとで FLGCommit を呼び出すようにしてください。一般に、プログラムがデータベースに対してなんらかの変更を行ったあとでは、常に FLGCommit を呼び出すようにすることができます。

特に以下のような状況では、データベースに対する変更をコミットすることをお勧めします。

- 一連の関連するメタデータ値を更新したあと。関連する情報が情報カタログ内で整合性を維持するようにするには、プログラムの中で、いくつかの関連する変更を行ったあとで FLGCommit を出すことができます。
- 新規のオブジェクト・タイプを完全に定義する一組の FLGCreateReg および FLGCreateType 呼び出し、あるいはオブジェクト・タイプを完全に除去する

一組の FLGDeleteReg および FLGDeleteType 呼び出しのあと。この時点であれば、プログラムがオブジェクト・タイプ定義の一部分だけをコミットすることがないことは明らかです。

- FLGDeleteTree および FLGDeleteTypeExt 呼び出しのあと。これらの呼び出しは情報カタログに重大な変更をするからです。
- FLGAppendType、FLGCreateReg、FLGCreateType、FLGDeleteType、および FLGDeleteTypeExt 呼び出しの前。これらの API 呼び出しは、重大なデータベース・エラーを検出すると自動的にデータベースをロールバックします。これらの 1 つまたは複数の API 呼び出しの前に FLGCommit 呼び出しを出すことにより、ロールバックが行われるときに、データベース・エラーに無関係なコミットされていない変更内容が情報カタログ・マネージャーによって除去されることが避けられます。
- FLGImport 呼び出しの前。情報カタログ・マネージャーは、FLGImport でエラーが発生するとデータベースをロールバックします。 FLGImport 呼び出しの前に行われたコミットされていない変更内容が情報カタログ・マネージャーによってロールバックされないようにするために、プログラムの中で、FLGImport を出す前に FLGCommit を出すようにしてください。

例

図45 は、FLGCommit を呼び出す C 言語コードを示しています。

```
APIRET        rc;           // Declare reason code from FLGCommit
FLGEXTCODE     ExtCode = 0;                // Declare extended code
  .
  .
rc = FLGCommit(&ExtCode);         // pass the address of
                              // extended code
```

図 45. C 言語による FLGCommit の呼び出しのサンプル

特別なエラー処理

FLGCommit でデータベース・エラーが発生すると、情報カタログ・マネージャーは、プログラム内で直前に行われたコミット点にデータベースをロールバックします。

このロールバックが正常に行われた場合、FLGCommit は理由コード FLG_SEVERR_DB_AUTO_ROLLBACK_COMPLETE を戻します。拡張コードに

は、情報カタログ・マネージャーがデータベースのロールバックを行う原因となったデータベース・エラーに関する SQL コードが含まれています。

考慮事項: このロールバックが失敗した場合には、FLGCommit は理由コード FLG_SEVERR_DB_AUTO_ROLLBACK_FAIL を戻します。拡張コードには、情報カタログ・マネージャーがデータベースのロールバックを行う原因となったデータベース・エラーに関する SQL コードが含まれています。この場合、データベースの保全性に関する重大な問題が発生している可能性があるため、プログラムで FLGTerm を呼び出して情報カタログ・マネージャーを終了させる必要があります。

データベースの状態によっては、バックアップ用データベース・ファイルを使用してデータベースを回復させなければならないことがあります。 情報カタログ・データベースの回復に関する詳細については、*情報カタログ・マネージャー 管理の手引き* を参照してください。

FLGConvertID

DP 名を与えられたオブジェクト・タイプのオブジェクト・タイプ ID、または FLGID を与えられたオブジェクト・インスタンスの名前を検索します。

許可

管理者またはユーザー

構文

```
APIRET  APIENTRY  FLGConvertID( PSZ             pszInBuffer,
                                PSZ             pszOutBuffer,
                                  FLGOPTIONS    Options,
                                  PFLGEXTCODE   pExtCode );
```

パラメーター

pszInBuffer (PSZ) − 入力
オブジェクト・インスタンス (FLGID) の 16 文字のシステム生成された固有の識別子、またはオブジェクト・タイプ (DP NAME) の 8 文字の省略形のいずれかを含む入力バッファーを指します。

pszOutBuffer (PSZ) − 出力
オブジェクト・インスタンスの 80 文字の外部名、または 6 文字のオブジェクト・タイプ ID のいずれかを含む出力バッファーを指します。

Options (FLGOPTIONS) − 入力
以下のいずれかのオプションを選択します。

FLG_DPNAME
入力バッファーが DP NAME を含むことを示します。

FLG_FLGID
入力バッファーが FLGID NAME を含むことを示します。

pExtCode (PFLGEXTCODE) − 出力
理由コードに関連する拡張コードを指します。戻された理由コードに関連した意味のある拡張コードがあるかどうかについては、285ページの『付録 D. 情報カタログ・マネージャーの理由コード』を参照してください。

理由コード (APIRET)
この API 呼び出しの実行結果を表します。

戻された理由コードの説明については、285ページの『付録D. 情報カタログ・マネージャーの理由コード』を参照してください。

例

図46 は、FLGConvertID 呼び出しを出すために必要な C 言語コードを示しています。このサンプル・コードは、特定のオブジェクト・タイプのオブジェクト・タイプ識別子を検索します。

```
APIRET         rc;                          // reason code
PSZ              pszInBuffer;         // pointer to input buffer
PSZ              pszOutBuffer;        // pointer to output buffer
FLGOPTIONS       options=FLG_DPNAME; // option flag
FLGEXTCODE       xc = 0;              // extended code
  .
  .
  .
strcpy (pszInBuffer,"CHARTS");        // object type's DP NAME
  .
  .
  .
rc = FLGConvertID (pszInBuffer,
                   pszOutBuffer,
                   options,
                     &xc);
```

図 46. C 言語による FLGConvertID の呼び出しのサンプル

FLGCreateInst

指定されたオブジェクト・タイプの新規インスタンスを作成するために使用します。

許可

管理者または許可ユーザー (すべてのオブジェクト・タイプ); ユーザー (Comments オブジェクト・タイプのみ)

構文

```
APIRET  APIENTRY    FLGCreateInst( PFLGHEADERAREA   pObjInstStruct,
                                   PSZ              pszFLGID,
                                  PFLGEXTCODE  pExtCode );
```

パラメーター

pObjInstStruct (PFLGHEADERAREA) – 入力
新規のオブジェクト・インスタンスに関する特性の仕様と値を含む入力構造を指します。

pszFLGID (PSZ) – 出力
新規のオブジェクト・インスタンスに関する 16 文字のシステム生成 ID を指します。

この ID の 1 文字目から 6 文字目までは、このインスタンスのオブジェクト・タイプを識別します。これらの文字の値は、入力構造ヘッダー・レコードの 33 バイトから 38 バイトまでと同じです。

この ID の 7 文字目から 16 文字目までは、システム生成された固有のインスタンス ID です。

他の API 呼び出しがこのインスタンスを参照するときには、ここで戻された pszFLGID が使用されます。

FLGCreateInst API 呼び出しが正常に実行されなかった場合には、pszFLGID は NULL にセットされます。

pExtCode (PFLGEXTCODE) – 出力
理由コードに関連する拡張コードを指します。戻された理由コードに関連した意味のある拡張コードがあるかどうかについては、285ページの『付録 D. 情報カタログ・マネージャーの理由コード』を参照してください。

理由コード (APIRET)

この API 呼び出しの実行結果を表します。

戻された理由コードの説明については、285ページの『付録D. 情報カタログ・マネージャーの理由コード』を参照してください。

入力構造

FLGCreateInst を使用するには、図47 に示す入力構造を定義しなければなりません。この構造はヘッダー域、定義域、およびオブジェクト域からなります。

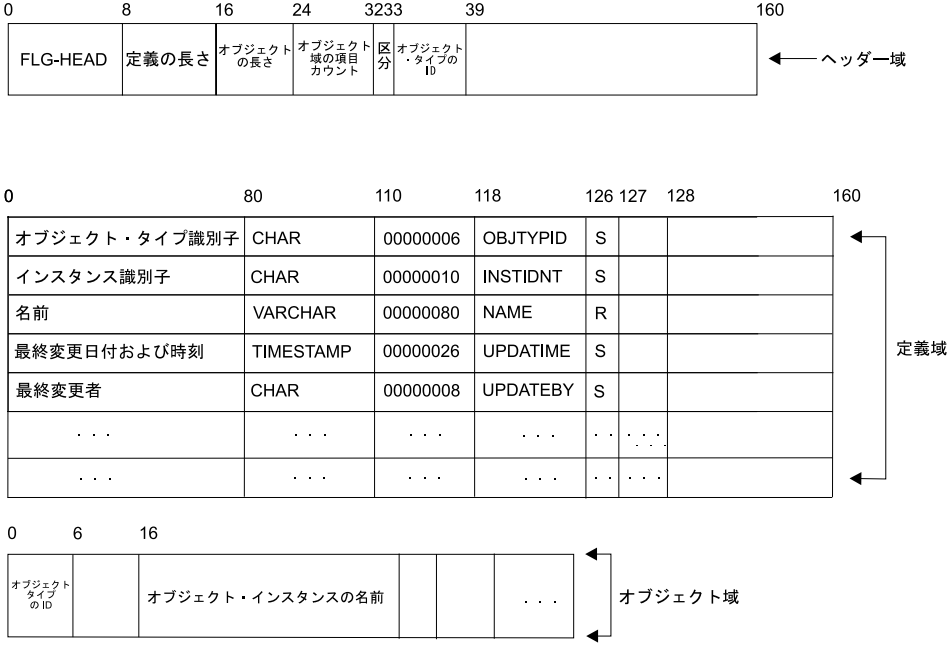


図 47. FLGCreateInst 入力構造

バイト・オフセットの意味については、36ページの『情報カタログ・マネージャー API 入力構造』を参照してください。

使用方法

前提条件

– オブジェクト・インスタンスを使用するには、そのオブジェクト・タイプがすでに情報カタログに存在していなければなりません。オブジェク

ト・タイプが存在しない場合には、FLGCreateReg 呼び出しを出してからさらに FLGCreateType 呼び出しを出して、オブジェクト・タイプの登録と作成を行う必要があります。

– FLGCreateInst 呼び出しを出すには、新規インスタンスを定義するために必要な特性に関する情報を得ておく必要があります。そのためには、FLGCreateType を出すか、あるいは FLGGetType を出してこの情報を取り出すかしてください。

制約事項

オブジェクト管理タスクの実行を許可されていないが、 Comments オブジェクト・インスタンスを作成している場合は、 Creator 特性値が自分のログオン・ユーザー ID になるようにしなくてはなりません。

入力要件

ヘッダー域

ヘッダー・レコードのすべてのフィールドが必要です。

定義域

定義域は、オブジェクト・インスタンスを作成しているオブジェクト・タイプの定義済みの特性のいくつか、またはすべてを含むことができます。以下の規則が適用します。

- まずはじめに、以下の順序で 5 つの情報カタログ・マネージャーの必須特性をすべて指定します。
  OBJTYPID、INSTIDNT、NAME、UPDATIME、および UPDATEBY。
- 126 桁目に R で示されている、他の必須特性をすべて指定します。
- 以下の指定について、情報カタログ・マネージャーはすべての指定された特性をオブジェクト・タイプ定義と比較します。

  データ・タイプ

  データ長

  特性省略名

  値フラグ

  UUI 番号

オブジェクト域

- 以下の特性に関する値は必ず指定しなければなりません。

OBJTYPID

33 桁目から 38 桁目のヘッダー域と同じでなくてはなりません。

NAME ブランクだけであってはなりません。

特性 NAME の値は、オブジェクト・タイプ内で固有である必要はありません。重複した項目を作成することもできます。ただし、重複する項目を作成する場合には、オブジェクト・インスタンスを区別することができるように、別の特性の値としてなんらかの記述情報を指定するようにしてください。

- 以下の特性の値はシステムで生成されるため、ブランクのままにしておかなければなりません。
  - INSTIDNT
  - UPDATIME
  - UPDATEBY
- 定義域の 126 桁目に R が入って定義されている必須特性に、値が指定されていない場合は、オブジェクト域の該当スペースを次のように初期設定しなければなりません。

データ・タイプ	初期設定値
CHAR	非適用記号のあとに、特性の長さ全体にわたってブランクが続きます。
TIMESTAMP	最大設定可能値に設定。 9999-12-31-24.00.00.000000
VARCHAR LONG VARCHAR	00000001; 8 バイトで指定された長さフィールド。非適用記号が続きます。

- 任意選択特性の値を指定しなかった場合には、オブジェクト域の該当スペースを次のように初期設定しなければなりません。

データ・タイプ	初期設定値
CHAR TIMESTAMP	特性の長さ全体にわたってブランク。
VARCHAR LONG VARCHAR	00000000。8 バイトの長さフィールドが存在していて、しかもゼロにセットされていなければなりません。

- 情報カタログ・マネージャーは、データ・タイプが VARCHAR または LONG VARCHAR のオブジェクト域の値に含まれる後続ブランクをすべて除去し、その区域の長さを自動的に調整します。
- HANDLES 特性に指定するオブジェクト・タイプは情報カタログに存在しているものでなければならず、しかも非 Program オブジェクト・タイプでなければなりません。 PARMLIST 特性に指定する特性は、HANDLES に指定されたオブジェクト・タイプの特性でなければなりません。詳細については、29ページの『プログラムを開始するための Programs オブジェクトの設定』を参照してください。

- 各オブジェクト・インスタンスの UUI 特性の値は固有でなければなりません。作成されるオブジェクト・インスタンスと同じ UUI 値のオブジェクト・インスタンスがすでに存在している場合には、エラーが発生します。

情報カタログ更新の制御

プログラムを可能な限り、情報カタログと同期化させるために、FLGCreateInst が正常に完了したあとで、呼び出しを FLGCommit (82ページの『FLGCommit』参照) に組み込む必要があります。 FLGCreateInst が正常に完了しない場合は、呼び出しを FLGRollback (225ページの『FLGRollback』参照) に組み込む必要があります。

例

図48 は、FLGCreateInst 呼び出しを出すために必要な C 言語コードを示しています。

このサンプル・コードは Grouping オブジェクト・タイプの新規インスタンスを作成します。

```
APIRET         rc;                    // Declare reason code
PFLGfEADERAREA  pObjInstStruct;        // Pointer to the input structure
UCHAR           pszFLGID[FLG_ID_LEN+1]; // Returns system-generated ID
FLGEXTCODE     ExtCode = 0;               // Declare extended code
  .
  /* creating pObjInstStruct object Instance by providing property values */
  .
  .
rc = FLGCreateInst (pObjInstStruct,    // input structure
                    pszFLGID,          // Returned ID of created instance
                   &ExtCode);             // Pass pointer to extended code
```

図 48. C 言語による FLGCreateInst の呼び出しのサンプル

92ページの図49 は、この FLGCreateInst 呼び出しのための入力構造を示しています。 pObjInstStruct パラメーターはこの構造を指します。新規のオブジェクト・インスタンスに関する特性および値の情報はこの構造に入っています。

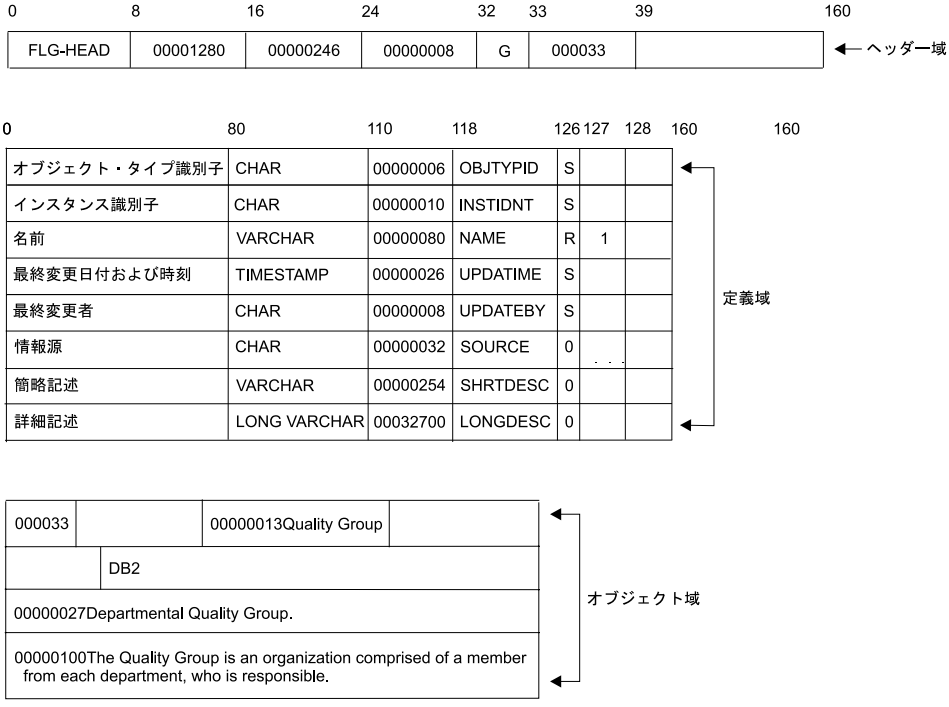


図 49. FLGCreateInst のための入力構造のサンプル

注:

1. ヘッダー・レコードの 33 桁目から 38 桁目までは、このオブジェクト・タイプの登録時に FLGCreateReg によって生成されたオブジェクト・タイプ識別子 (000033) が入っています。これと同じ値をオブジェクト域の OBJTYPID に指定しなければなりません。この例では、オブジェクト域の最初の値として指定されています。
2. このオブジェクト・タイプでは、最初に 5 つの必須特性 (OBJTYPID、INSTIDNT、NAME、UPDATIME、UPDATEBY) が指定され、さらに、ユーザーによって追加された 3 つの特性が指定されています。

FLGCreateReg

オブジェクト・タイプに関する登録情報を情報カタログ内に作成するために使用します。

この API 呼び出しはオブジェクト・タイプ自体を作成するためのものではなく、オブジェクト・タイプの作成が行えるようにオブジェクト・タイプを登録するためのものです。 FLGCreateReg が情報カタログに保管する登録情報には、オブジェクト・タイプを記述する登録値が含まれています。

Program および Attachment 区分を除き、どの区分にもタイプを登録することができます。これらの区分はそれぞれ、情報カタログ・マネージャーが情報カタログ内に自動的に作成する Programs および Comments タイプしか含むことができないからです。

許可

管理者

構文

```
APIRET  APIENTRY  FLGCreateReg( PFLGHEADERAREA   pObjRegStruct,
                                 PSZ              pszIconFileID,
                                PSZ              pszObjTypeID,
                                  PFLGEXTCODE  pExtCode );
```

パラメーター

pObjRegStruct (PFLGHEADERAREA) − 入力
新規のオブジェクト・タイプ登録に関する特性の仕様と値を含む入力構造を指します。

pszIconFileID (PSZ) − 入力
新規のオブジェクト・タイプ登録のアイコンを含むファイルのドライブ、ディレクトリー、およびファイル名が含まれます。このパラメーターが NULL の場合、新規のオブジェクト・タイプ登録にはアイコンが関連付けられません。

pszObjTypeID (PSZ) − 出力
登録済みのオブジェクト・タイプに関する 6 文字のシステム生成された固有の識別子 (オブジェクト・タイプ ID) を指します。

ここで戻された ObjTypeID は、オブジェクト・タイプを識別するために別の API 呼び出しによって使用されます。オブジェクト・タイプが正常に登録されなかった場合、このパラメーターは NULL にセットされます。

pExtCode (PFLGEXTCODE) － 出力

理由コードに関連する拡張コードを指します。戻された理由コードに関連した意味のある拡張コードがあるかどうかについては、285ページの『付録 D. 情報カタログ・マネージャーの理由コード』を参照してください。

理由コード (APIRET)

この API 呼び出しの実行結果を表します。

戻された理由コードの説明については、285ページの『付録D. 情報カタログ・マネージャーの理由コード』を参照してください。

入力構造

FLGCreateReg を使用するには、図50 に示す入力構造を定義しなければなりません。この構造はヘッダー域、定義域、およびオブジェクト域からなります。

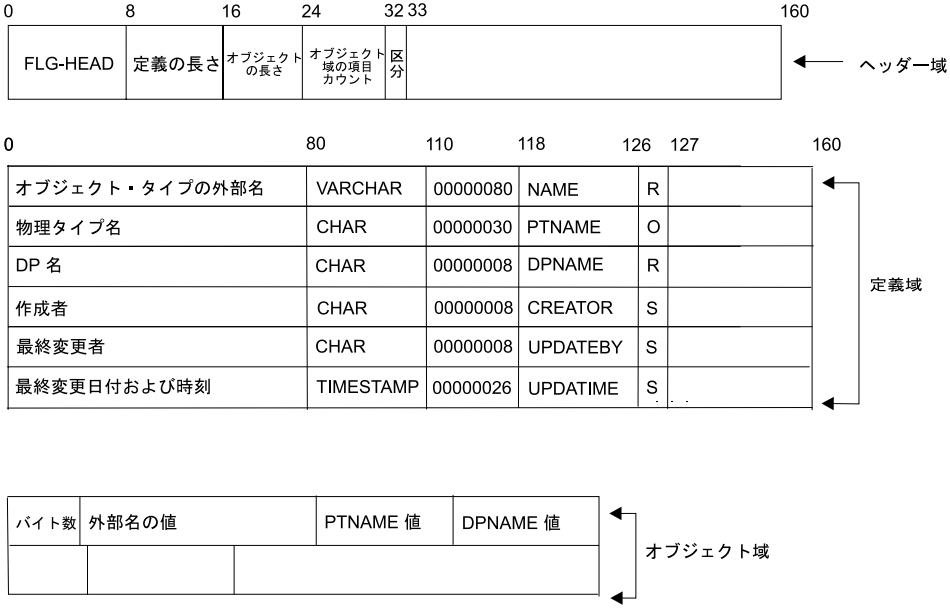


図 50. FLGCreateReg 入力構造

入力構造では、ここで示された順番どおりに 6 つの特性を定義域に指定する必要があります。

英語版以外の情報カタログ・マネージャーを使用している場合には、これらの必須特性の名前は翻訳されており、また FLGInit の出力構造に戻されます。

表15

表 15. オブジェクト・タイプ登録に必要な特性

特性省略名	特性名[1]	説明	オブジェクト域での値の指定
NAME	オブジェクト・タイプの外部名	オブジェクト・タイプの名前を 80 バイトで指定。あとで修正可能。	必須。
PTNAME	物理タイプ名	情報カタログにおけるオブジェクト・タイプを含んでいる実際の表の名前を 30 文字で指定。	任意選択。デフォルトは DPNAME の値。
DPNAME	DP 名	オブジェクト・タイプを表す 8 文字の省略名。	この値は FLGCreateReg を使用して指定する必要がある。 必須。
CREATOR	作成者	オブジェクト・タイプを作成した管理者のユーザー ID を 8 文字で指定。	不可。この値は、オブジェクト・タイプに関して FLGCreateType が出されたときに情報カタログ・マネージャーによって設定される。
UPDATEBY	最終変更者	オブジェクト・タイプを最後に修正した管理者のユーザー ID を 8 文字で指定。	不可。この値は、オブジェクト・タイプに関して FLGAppendType が出されたときに情報カタログ・マネージャーによって指定される。
UPDATIME	最終変更日付および時刻	オブジェクト・タイプが最後に修正された日付と時刻のタイム・スタンプを 26 文字で指定。	不可。この値は、オブジェクト・タイプに関して FLGCreateType または FLGAppendType が出されたときに情報カタログ・マネージャーによって指定される。

注:

1. この列の特性名は、情報カタログ・マネージャーの英語バージョンに適用します。情報カタログ・マネージャーの翻訳バージョンを使用している場合は、特性名も翻訳されます。

バイト・オフセットの意味の一般的な説明については、 36ページの『情報カタログ・マネージャー API 入力構造』を参照してください。

使用方法

制約事項

– 新規の Program オブジェクト・タイプを追加することができないため、Program 区分 (P) に関して新規のオブジェクト・タイプを登録することはできません。情報カタログを作成する際は、区分 Program の許可されたオブジェクト・タイプ (″Programs″) のみが組み込まれます。

– 新規の Attachment オブジェクト・タイプを追加することができないため、 Attachment 区分 (A) に関して新規のオブジェクト・タイプを登録することはできません。情報カタログを作成する際は、区分 Attachment の許可されたオブジェクト・タイプ (″Comments″) のみが組み込まれます。

– アイコンをオブジェクト・タイプに割り当てるには、FLGManageIcons を使用します (198ページの『FLGManageIcons』参照)。

– FLGCreateReg を使って、オブジェクト・タイプを定義したあとで、FLGUpdateReg または FLGManageIcons 呼び出しを発行し、オブジェクト・タイプに関連したアイコンを変更したり、もともと定義されていないアイコン関連を追加することができます。アイコンをオブジェクト・タイプから除去するのに、FLGManageIcons を使用することもできます。

入力要件

ヘッダー域

94ページの図50 のヘッダー・レコードに示されたすべての情報を指定する必要があります。

定義域

- 定義域には、94ページの図50 に示された 6 つのそれぞれの登録特性に関する定義を含める必要があります。翻訳済みの特性名 (95ページの表15 参照) を除き、各登録特性の定義は、表示されているとおりに指定しなければなりません。

- それぞれの必須特性名 (各特性に関するバイト 0 からバイト 80 まで) は、 94ページの図50 に示した英語の特性名からサポートされる各国語の特性名に翻訳することができます。これらの必須特性の翻訳名は、FLGInit の出力構造に入れて戻されます。

オブジェクト域

- バイト 126 の値が S で定義されている特性については、オブジェクト域の値をブランクのままにしてください。これらの特性の値はユーザーがオブジェクト・タイプを作成または追加したときにシステムによって生成されるため、値を指定しても情報カタログ・マネージャー

によって無視されます。 CREATOR、UPDATEBY、および UPDATIME の各特性がこれに該当します。

- PTNAME に関する規則
  - オブジェクト・タイプの PTNAME は情報カタログ・マネージャー・カタログ内で固有でなければなりません。
  - 情報カタログ・マネージャーにおける PTNAME 値の最大長は FLG_PTNAME_LEN (30) です。ただし、データベースの制約により、ユーザーの情報カタログ環境における最大長が短くなっている可能性があります。この最大値の設定に関する詳細については、*情報カタログ・マネージャー 管理の手引き* を参照してください。
  - 後続ブランクを含まない PTNAME の有効文字数が使用環境で許される最大値 (FLGInit によって戻された STOR ENVSIZE の値) を超えている場合には、登録要求は拒絶されます。
  - PTNAME の指定は任意選択です。 PTNAME を指定しない場合、情報カタログ・マネージャーはこの値をデフォルトによって DPNAME の値にセットします。
  - PTNAME に関する制約事項は以下のとおりです。
    - SBCS だけで指定しなければなりません。
    - 先頭文字は、A から Z または a から z まで、@、#、$ でなければなりません。
    - 先頭以外の文字には、A から Z または a から z まで、0 から 9 まで、 @ または #、$ または _ (下線) を使用することができます。
    - 先行または組み込みブランクを使用することはできません。
    - 使用されるデータベースに関する SQL 予約語を PTNAME として使用することはできません。
- オブジェクト・タイプの DPNAME に関する制約事項
  - 組織内のすべての情報カタログ内で固有でなければなりません。
  - SBCS だけで指定しなければなりません。
  - 先頭文字は、A から Z または a から z まで、@ または #、あるいは $ でなければなりません。
  - 先頭以外の文字には、A から Z または a から z まで、0 から 9 まで、 @ または #、$ または _ (下線) を使用することができます。
  - 先行または組み込みブランクを使用することはできません。

- NAME 値はローカル情報カタログ内で固有でなければなりません。

出力情報

システム生成のオブジェクト・タイプ識別子は、出力パラメーター pszObjTypeID に入って戻されます。情報カタログ・マネージャーがこの番号を戻した場合には、後続の FLGDeleteReg または FLGGetReg などの呼び出しでこの番号を使用して、オブジェクト・タイプ登録を個別に識別することができます。

情報カタログ更新の制御

FLGCreateReg が正常に完了しない場合は、呼び出しを FLGRollback (225ページの『FLGRollback』参照) に組み込む必要があります。 FLGCreateReg が正常に完了したあとで、FLGCommit を呼び出さずに、FLGCreateType への呼び出しが完了するまで待ってください。

例

図51 は、FLGCreateReg 呼び出しを出すために必要な C 言語コードを示しています。このサンプル・コードは、Elemental 区分に属する MYIMAGE という新規のオブジェクト・タイプに関する登録情報を作成します。

```
APIRET         rc;                    // Declare reason code
PFLGHEADERAREA pObjRegStruct;         // Pointer to the input structure
UCHAR          pszIconFileID[FLG_ICON_FILE_ID_MAXLEN+1]; // Path/File name of ICON
UCHAR          pszObjTypeID[FLG_OBJTYPID_LEN+1];      // Returned system-generated ID
FLGEXTCODE     ExtCode = 0;                 // Declare extended code
  .
  .  /* creating pObjRegStruct object Type Registration by providing values  */
  .
strcpy (pszIconFileID,"Y:¥¥FLGICON2.ICO");
rc = FLGCreateReg (pObjRegStruct,   // input structure
                   pszIconFileID,     // Path/File name of file containing the ICON
                   pszObjTypeID,      //  Returned id of registered object type
                   &ExtCode);              // Pass extended code pointer
```

図 51. C 言語による FLGCreateReg の呼び出しのサンプル

99ページの図52 は、この FLGCreateReg 呼び出しのための入力構造を示しています。 pObjRegStruct ポインターは、新規のオブジェクト・タイプの登録に必要な特性および値に関する情報を指します。

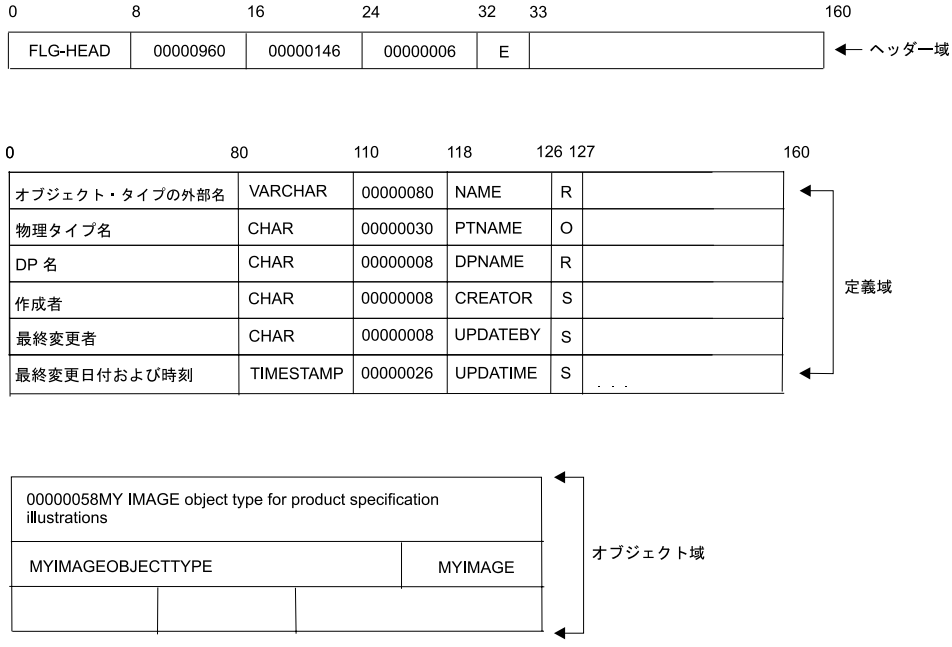


図 52. FLGCreateReg のための入力構造のサンプル

特別なエラー処理

FLGCreateReg でデータベース・エラーが発生すると、情報カタログ・マネージャーは、プログラム内で最後に行われたコミット点にデータベースをロールバックします。

このロールバックが正常に行われた場合、FLGCreateReg は理由コード FLG_SEVERR_DB_AUTO_ROLLBACK_COMPLETE を戻します。拡張コードには、情報カタログ・マネージャーがデータベースのロールバックを行う原因となったデータベース・エラーに関する SQL コードが含まれています。

考慮事項: このロールバックが失敗した場合には、FLGCreateReg は理由コード FLG_SEVERR_DB_AUTO_ROLLBACK_FAIL を戻します。拡張コードには、情報カタログ・マネージャーがデータベースのロールバックを行う原因となったデータベース・エラーに関する SQL コードが含まれています。この場合、データベースの保全性に関する重大な問題が発生している可能性があるため、プログラムで FLGTerm を呼び出して情報カタログ・マネージャーを終了させる必要があります。

データベースの状態によっては、バックアップ用データベース・ファイルを使用してデータベースを回復させなければならないことがあります。 情報カタログ・データベースの回復に関する詳細については、*情報カタログ・マネージャー 管理の手引き* を参照してください。

FLGCreateReg エラーとは無関係なコミットされていない変更内容が情報カタログ・マネージャーによって除去されないようにするために、プログラム内のこの呼び出しの直前に FLGCommit 呼び出しを組み込んでください。

FLGCreateType

新規のユーザー定義オブジェクト・タイプを作成します。

管理者は、Program および Attachment 区分を除いて、どの区分にもタイプを作成することができます。これらの区分は、情報カタログ内で情報カタログ・マネージャーが自動的に作成する Programs および Comments タイプのみを含むことができるからです。

許可

管理者

構文

```
APIRET  APIENTRY   FLGCreateType( PFLGHEADERAREA   pObjTypeStruct,
                                  PFLGEXTCODE  pExtCode );
```

パラメーター

pObjTypeStruct (PFLGHEADERAREA) – 入力
入力構造を指します。入力構造にはこのオブジェクト・タイプに関する特性の仕様が含まれています。

pExtCode (PFLGEXTCODE) – 出力
理由コードに関連する拡張コードを指します。戻された理由コードに関連した意味のある拡張コードがあるかどうかについては、285ページの『付録D. 情報カタログ・マネージャーの理由コード』を参照してください。

理由コード (APIRET)
この API 呼び出しの実行結果を表します。

戻された理由コードの説明については、285ページの『付録D. 情報カタログ・マネージャーの理由コード』を参照してください。

入力構造

FLGCreateType を使用するには、102ページの図53 に示す入力構造を定義しなければなりません。この構造はヘッダー域と定義域だけからなります。

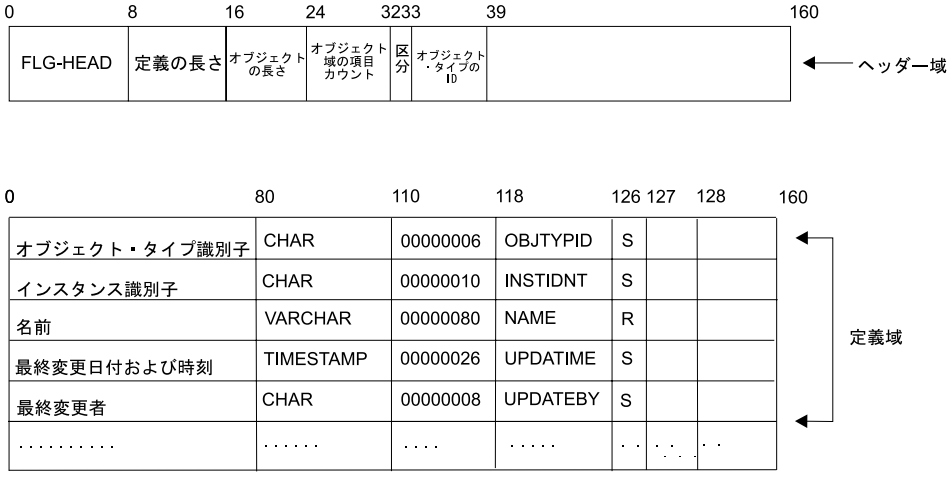


図 53. FLGCreateType 入力構造

バイト・オフセットの意味については、36ページの『情報カタログ・マネージャー API 入力構造』を参照してください。

使用方法

前提条件

FLGCreateType API を呼び出して新規オブジェクト・タイプを作成する前に、この新規タイプを登録するために FLGCreateReg API を呼び出す必要があります。

FLGCreateType を呼び出すときには、FLGCreateReg API によって戻されたオブジェクト・タイプ識別子を指定する必要があります。

入力要件

ヘッダー域

入力構造のヘッダーに指定するオブジェクト・タイプは、登録済みであって、しかもまだ作成されていないものでなければなりません。

定義域

- 定義域に定義する最初の 5 つの特性は、情報カタログ・マネージャーの 5 つの必須特性 OBJTYPID、 INSTIDNT、NAME、UPDATIME、および UPDATEBY に対応するものでなければなりません。これらの特性がこのとおりの順番で指定されていない場合には、作成は失敗します。

OBJTYPID
オブジェクト・タイプを表す固有のシステム生成識別子 (ID)。

INSTIDNT
オブジェクトを表す固有のシステム生成 ID。

NAME オブジェクトを表すユーザー指定の名前。

UPDATIME
オブジェクトが最後に更新された時期を表すシステム生成のタイム・スタンプ。

UPDATEBY
オブジェクトを最後に更新した管理者またはユーザーを表すシステム生成のユーザー ID。

- 必須特性に関する規則
  - これらの各必須特性のデータ・タイプ、長さ、特性省略名、および値フラグ (vf) は固定されており、102ページの図53 に示したとおりに指定しなければなりません。
  - UUI 順序 (us) は 4 つの各システム生成 (S) 特性についてはブランクに固定されていますが、NAME 特性については 1、2、3、4、5、またはブランクにすることができます。
  - 80 バイトの特性名は固定されていますが、サポートされる各国語版では翻訳可能です。これらの必須特性の翻訳名は、FLGInit の出力構造に入れて戻されます。
- 定義における特性の合計数は、FLG_MAX_PROPERTIES (255) を超えてはなりません。
- 定義におけるデータ・タイプが LONG VARCHAR の特性の合計数は、情報カタログ・マネージャーの限界である FLG_MAX_NUM_LONG_VARCHARS (14) を超えてはなりません。
- UUI に関する規則
  - 作成するそれぞれのオブジェクト・タイプについて、少なくとも 1 つは UUI 特性を定義しなければなりません。
  - オブジェクト・タイプ内では、UUI の番号付けは 1 から始める必要があり、番号を飛ばすことはできません。たとえば、あるオブジェクト・タイプ内で 1、2、および 3 という一連の UUI 順序番号値を指定することは有効ですが、2、3、および 5 という指定は無効です。

  - 同じオブジェクト・タイプ内で同一の UUI 順序番号を 2 回以上指定することはできません。
  - UUI として指定する特性の長さは、254 バイトを超えてはなりません。
  - UUI として指定する特性は、必須特性 (桁 126 に ″R″ 値フラグがある特性) でなければなりません。
  - UUI として指定する特性 (桁 127) のデータ・タイプを LONG VARCHAR にすることはできません。
  - UUI 特性を定義するときには、そのオブジェクトの各インスタンスが個別に識別できるようにしてください。このオブジェクトのインスタンスが複数の関連する情報カタログに存在していても、これらの特性によって特定のインスタンスが識別できるようにする必要があります。

- 必須特性以外にも、作成したオブジェクト・タイプを業務の必要に応じて調整するために、さらに特性を追加することができます。

  情報カタログ・マネージャーは、すべての必須特性が常に指定されていることを確認するために、バイト 118 からバイト 125 までの特性省略名をキーとして使用しますので、定義域におけるこれらの追加特性の順序は影響を与えません。 FLGCreateType 呼び出しの場合の定義域における特性の順序は、情報カタログ・マネージャーによって呼び出し元アプリケーションに戻された特性の順番どおりになります。
- 新規の特性名は、そのオブジェクト・タイプの中では固有のものでなければなりません。
- 新規の特性省略名は、そのオブジェクト・タイプの中では固有のものでなければなりません。
- この特性が 2 つ以上の関連する情報カタログで共用されるオブジェクト・タイプに属する場合に、情報を共用するために情報カタログ・マネージャー・データのインポートおよびエクスポートを計画しているときには、データ・タイプ、データ長、特性省略名、値フラグ、および UUI 順序の値は、他の情報カタログにおける同一オブジェクト・タイプに関する値と同じにしなければなりません。特性名は異なっていても差し支えありません。
- 特性省略名は以下の規則に従っている必要があります。
  - SBCS だけで指定しなければなりません。
  - 先頭文字は、A から Z または a から z まで、@ または #、あるいは $ でなければなりません。

- 先頭以外の文字には、A から Z または a から z まで、0 から 9 まで、 @ または #、$ または _ (下線) を使用することができます。
- 先行または組み込みブランクを使用することはできません。
- 使用されるデータベースに関する SQL 予約語を使用することはできません。

- 1 つのオブジェクト・タイプに関するすべての特性の合計長が、使用するデータベースの行限界を超えてはなりません。行の長さの計算方法については、使用するデータベースに関する資料を参照してください。

情報カタログ更新の制御

プログラムを可能な限り、情報カタログと同期化させるために、FLGCreateType が正常に完了したあとで、呼び出しを FLGCommit (82ページの『FLGCommit』参照) に組み込む必要があります。
FLGCreateType が正常に完了しない場合は、呼び出しを FLGRollback (225ページの『FLGRollback』参照) に組み込む必要があります。

例

図54 は、FLGCreateType API 呼び出しを出すために必要な C 言語コードを示しています。

このサンプル・コードは新規のオブジェクト・タイプを作成します。この新規オブジェクト・タイプは、構造ヘッダー域に E が指定されていることから分かるように Elemental 区分に属し、オブジェクト・タイプ ID は 000044 です。このオブジェクト・タイプには、情報カタログ・マネージャー必須特性のほかに、イメージ・カラー、イメージ・サイズ、および説明の 3 つの追加必須特性が含まれています。

```
APIRET         rc;                    // Declare reason code
PFLGHEADERAREA   pObjTypeStruct;  // Pointer to the input structure
FLGEXTCODE     ExtCode=0;                 // Declare extended code
  .
  .  /* creating pObjTypeStruct object type by */
     /* providing object type's properties  */
  .
rc = FLGCreateType (pObjTypeStruct,
                    &ExtCode);             // Pass pointer to extended code
```

図 54. C 言語による FLGCreateType の呼び出しのサンプル

図55 は、この FLGCreateType API 呼び出しのための入力構造を示しています。 pObjTypeStruct ポインターは、新規のオブジェクト・タイプの特性情報が入っている構造を指します。

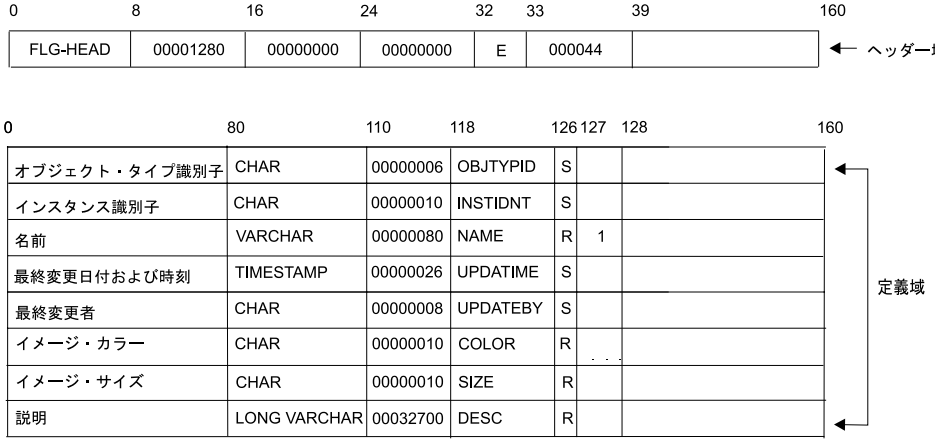


図 55. FLGCreateType のための入力構造のサンプル

特別なエラー処理

FLGCreateType でデータベース・エラーが発生すると、情報カタログ・マネージャーは、プログラム内で最後に行われたコミット点にデータベースをロールバックします。

このロールバックが正常に行われた場合、FLGCreateType は理由コード FLG_SEVERR_DB_AUTO_ROLLBACK_COMPLETE を戻します。拡張コードには、情報カタログ・マネージャーがデータベースのロールバックを行う原因となったデータベース・エラーに関する SQL コードが含まれています。

考慮事項: このロールバックが失敗した場合には、FLGCreateType は理由コード FLG_SEVERR_DB_AUTO_ROLLBACK_FAIL を戻します。拡張コードには、情報カタログ・マネージャーがデータベースのロールバックを行う原因となったデータベース・エラーに関する SQL コードが含まれています。この場合、データベースの保全性に関する重大な問題が発生している可能性があるため、プログラムで FLGTerm を呼び出して情報カタログ・マネージャーを終了させる必要があります。

データベースの状態によっては、バックアップ用データベース・ファイルを使用してデータベースを回復させなければならないことがあります。 情報カタロ

グ・データベースの回復に関する詳細については、*情報カタログ・マネージャー 管理の手引き* を参照してください。

FLGCreateType エラーとは無関係なコミットされていない変更内容が情報カタログ・マネージャーによって除去されないようにするために、作成中のオブジェクト・タイプの FLGCreateReg プログラム内のこの呼び出しの直前に FLGCommit 呼び出しを組み込んでください。

FLGDeleteInst

あるオブジェクト・タイプの、単一の特定オブジェクト・インスタンスを削除します。

許可

管理者または許可ユーザー (すべてのオブジェクト・タイプ); ユーザー (Comments オブジェクト・タイプのみ)

構文

```
APIRET  APIENTRY    FLGDeleteInst( PSZ              pszFLGID,
                                   PFLGEXTCODE  pExtCode );
```

パラメーター

pszFLGID (PSZ) – 入力
削除するインスタンスに関する 16 文字のシステム生成された固有の ID を指します。

この ID の 1 文字目から 6 文字目までは、このインスタンスのオブジェクト・タイプを識別します。

この ID の 7 文字目から 16 文字目までは、システム生成された固有のインスタンス ID です。

pExtCode (PFLGEXTCODE) – 出力
理由コードに関連する拡張コードを指します。戻された理由コードに関連した意味のある拡張コードがあるかどうかについては、285ページの『付録 D. 情報カタログ・マネージャーの理由コード』を参照してください。

理由コード (APIRET)
この API 呼び出しの実行結果を表します。

戻された理由コードの説明については、285ページの『付録D. 情報カタログ・マネージャーの理由コード』を参照してください。

使用方法

前提条件

pszFLGID 入力パラメーターに指定する値は、存在している値でなければなりません。

制約事項

オブジェクト管理タスクの実行を許可されていないユーザーの場合は、ログオン・ユーザー ID の値が作成者の値と同じである Comments インスタンスのみを削除することができます。

関連するオブジェクト・インスタンスに関する規則

Attachment 関係に参加しているインスタンスの場合、

- インスタンスが 1 つ以上の関連する Comments インスタンスをもつ場合は、オブジェクト・インスタンス自体を削除すると、すべての Comments インスタンス、および、すべてのこのような関係も削除されます。
- インスタンスが Comments である場合は、インスタンスは Attachment 関係にあるので、すべてこのような関係は、Comments オブジェクト・インスタンスそのものが削除されると、同時に削除されます。

インスタンスの中に別のインスタンスが含まれている場合、

- 包含する側のインスタンスを削除するときには、FLGDeleteInst を使ってインスタンスを削除する前に、含まれているオブジェクト・インスタンスとの間のすべての関係を削除しなければなりません。包含する側のインスタンスと含まれているオブジェクト・インスタンスとのすべての関係を削除するときには、代わりに FLGDeleteTree を使うことができます (113ページの『FLGDeleteTree』参照)。
- 別のオブジェクトに含まれているインスタンスの場合には、それを含むオブジェクトとの関係を削除せずにこのインスタンスを削除することができます。相互の関係もインスタンス自体も、ともに自動的に削除されます。

Contact 関係にあるインスタンスの場合、

- 何らかの Contact 関係にあるインスタンスの場合には、オブジェクト・インスタンス自体を削除すると、これらのすべての関係も削除されます。
- インスタンスが Contact 関係における Contact である場合には、Contact オブジェクト・インスタンス自体を削除すると、これらのすべての関係も削除されます。

リンク関係にあるインスタンスの場合、

何らかのリンク関係にあるインスタンスの場合には、オブジェクト・インスタンス自体を削除すると、これらのすべての関係も削除されます。

非 Program オブジェクト・タイプに関連付けられた Programs インスタンスの場合、

Programs インスタンスをいつ削除しても、関連するオブジェクト・タイプは影響を受けません。

情報カタログ更新の制御

プログラムを可能な限り、情報カタログと同期化させるために、FLGDeleteInst が正常に完了したあとで、呼び出しを FLGCommit (82ページの『FLGCommit』参照) に組み込む必要があります。 FLGDeleteInst が正常に完了しない場合は、呼び出しを FLGRollback (225ページの『FLGRollback』参照) に組み込む必要があります。

例

図56 は、FLGDeleteInst 呼び出しを出すために必要な C 言語コードを示しています。このサンプル・コードは、オブジェクト・タイプを削除します。

```
APIRET         rc;                    // Declare reason code
UCHAR         pszFLGID[FLG_ID_LEN + 1];   // Unique instance identifier
FLGEXTCODE     ExtCode = 0;                // Declare extended code
  .
  .  /* Get FLGID for object instance using FLGSearch. */
  .
strcpy (pszFLGID,"0000330000001234");
rc = FLGDeleteInst (pszFLGID,             // Instance ID
                    &ExtCode);
```

図 56. C 言語による FLGDeleteInst の呼び出しのサンプル

FLGDeleteReg

情報カタログから特定のオブジェクト・タイプ登録を削除するために使用します。

Program および Attachment 区分以外のどの区分のタイプの登録でも削除することができます。情報カタログ・マネージャーは、情報カタログを作成中にこれらの区分を供給するからです。

許可

管理者

構文

```
APIRET  APIENTRY  FLGDeleteReg( PSZ           pszObjTypeID,
                                PFLGEXTCODE  pExtCode );
```

パラメーター

pszObjTypeID (PSZ) – 入力
登録を削除するオブジェクト・タイプを表す 6 文字の固有のシステム生成識別子 (オブジェクト・タイプ ID) を指します。

pExtCode (PFLGEXTCODE) – 出力
理由コードに関連する拡張コードを指します。戻された理由コードに関連した意味のある拡張コードがあるかどうかについては、285ページの『付録 D. 情報カタログ・マネージャーの理由コード』を参照してください。

理由コード (APIRET)
この API 呼び出しの実行結果を表します。

戻された理由コードの説明については、285ページの『付録D. 情報カタログ・マネージャーの理由コード』を参照してください。

使用方法

このアクションは、オブジェクト・タイプ自体を削除するものではありません。オブジェクト・タイプの登録を削除するものです。

制約事項

入力パラメーター pszObjTypeID の値は、情報カタログ内のオブジェクト登録に関して存在している値でなければなりません。

オブジェクト・タイプの登録を削除するためには、オブジェクト・タイプ自体が存在していてはなりません。オブジェクト・タイプが存在している場合には、FLGDeleteType を使用してオブジェクト・タイプを削除しなければなりません。

情報カタログ更新の制御

プログラムを可能な限り、情報カタログと同期化させるために、FLGDeleteReg が正常に完了したあとで、呼び出しを FLGCommit (82ページの『FLGCommit』参照) に組み込む必要があります。 FLGDeleteReg が正常に完了しない場合は、呼び出しを FLGRollback (225ページの『FLGRollback』参照) に組み込む必要があります。

例

図57 は、FLGDeleteReg API 呼び出しを出すために必要な C 言語コードを示しています。このサンプル・コードは、あるオブジェクト・タイプに関する登録情報を情報カタログから削除します。

```
APIRET         rc;                    // Declare reason code
UCHAR           pszObjTypeID[FLG_OBJTYPID_LEN + 1];
FLGEXTCODE     ExtCode = 0;                // Declare extended code
  .
  .  /* Get object type ID using FLGConvertID. */
  .
strcpy (pszObjTypeID,"000044");
rc = FLGDeleteReg (pszObjTypeID,    // object type ID
                       &ExtCode);
```

図 57. C 言語による FLGDeleteReg の呼び出しのサンプル

図57 に示される例は、(FLGCreateReg を使用して) オブジェクト登録を作成したときにオブジェクト・タイプ ID として 000044 が戻されていることを前提としています。

FLGDeleteTree

Grouping オブジェクト・タイプの特定のインスタンス、それに付加されたすべての Comments インスタンス、およびそれが参加するすべての ATTACHMENT、 CONTACT、 CONTAIN、および LINK 関係を削除するために使用します。また、任意に Grouping 区分のオブジェクト・インスタンスに含まれたすべてのオブジェクト・インスタンス、それに付加されたすべての Comments インスタンス、参加しているすべての ATTACHMENT、CONTACT、および LINK 関係を削除するために使用します。

許可

管理者または許可ユーザー

構文

```
APIRET  APIENTRY   FLGDeleteTree( PSZ              pszFLGID,
                                  FLGOPTIONS   Options,
                                 PFLGHEADERAREA * ppListStruct,
                                  PFLGEXTCODE  pExtCode );
```

パラメーター

pszFLGID (PSZ) − 入力

削除する Grouping 区分のインスタンス (包含側) に関する 16 文字のシステム生成された固有の ID を指します。

この ID の 1 文字目から 6 文字目までは、このインスタンスのオブジェクト・タイプを識別します。

この ID の 7 文字目から 16 文字目までは、システム生成された固有のインスタンス ID です。

Options (FLGOPTIONS) − 入力

以下のいずれかのオプションを選択します。

FLG_DELTREE_ALL

Grouping 区分のオブジェクト・インスタンス、それに付加されたすべての Comments インスタンス、および参加するすべての ATTACHMENT、CONTACT、および LINK 関係を削除するために使用します。 Grouping 区分のオブジェクト・インスタンスに含まれたすべてのオブジェクト・インスタンス、それに付加されたすべての Comments インスタンス、および参加するすべての

ATTACHMENT、CONTACT、および LINK 関係を削除するために使用します。 115ページの図58 から 116ページの図60 にこのオプションが図示されています。

FLG_DELTREE_REL

Grouping 区分のオブジェクト・インスタンス、それに付加されたすべての Comments インスタンス、および参加するすべての ATTACHMENT、CONTACT、および LINK 関係を削除するために使用します。基礎となる CONTAIN 関係のツリー構造を削除するために使用します。 115ページの図58 から 116ページの図60 にこのオプションが図示されています。

ppListStruct (PFLGHEADERAREA) － 出力

削除されたオブジェクト・インスタンスが入っている出力構造のポインターのアドレスを指します。

この出力構造には、削除されたオブジェクト・インスタンスそれぞれの 16 文字の FLGID が入ります。

このパラメーターが NULL のとき、出力構造は戻されません。出力構造がない場合、出力構造へのポインターは NULL にセットされます。

pExtCode (PFLGEXTCODE) － 出力

理由コードに関連する拡張コードを指します。戻された理由コードに関連した意味のある拡張コードがあるかどうかについては、285ページの『付録 D. 情報カタログ・マネージャーの理由コード』を参照してください。

理由コード (APIRET)

この API 呼び出しの実行結果を表します。

戻された理由コードの説明については、285ページの『付録D. 情報カタログ・マネージャーの理由コード』を参照してください。

使用方法

前提条件

指定されたオブジェクト・インスタンス ID (FLGID) が存在しなくてはなりません。

制約事項

削除されるオブジェクト以外の Grouping オブジェクトに含まれるオブジェクト・インスタンスは (116ページの図59 に図示) 削除されません。

出力構造に割り振られたメモリーの解放

FLGDeleteTree が出力構造にあるデータを戻した場合、出力構造に戻されたデータを保管してから、FLGFreeMem を呼び出さなくてはなりません (137ページの『FLGFreeMem』参照)。メモリーを解放するのに、たとえば C 言語指示のような他の方式は使用しないでください。

情報カタログ更新の制御

プログラムを可能な限り、情報カタログと同期化させるために、FLGDeleteTree が正常に完了したあとで、呼び出しを FLGCommit (82ページの『FLGCommit』参照) に組み込む必要があります。 FLGDeleteTree が正常に完了しない場合は、呼び出しを FLGRollback (225ページの『FLGRollback』参照) に組み込む必要があります。

例

図58 から 116ページの図60 は、 2 つの削除オプションの効果を図示しています。 図58 は、3 つのグループ・オブジェクト、 A、Z、および Y の情報カタログを表示しています。オブジェクト B は、FLGDeleteTree を使って削除されます。

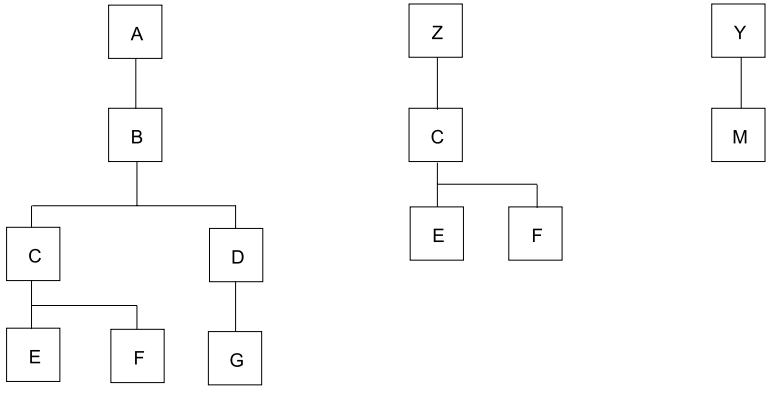


図 58. 削除前の情報カタログのサンプル

FLG_DELTREE_REL オプションを使って、オブジェクト・インスタンス B と B の下にあるいくつかの関係が削除されました。 C は、別の木である Z に含まれているので、オブジェクト C と、それが含むものは影響を受けていません。オブジェクト D は、他のオブジェクトに含まれていないので、カスケード効果の影響を受けます。

116ページの図59 は、B が削除されたあとの情報カタログを図示しています。

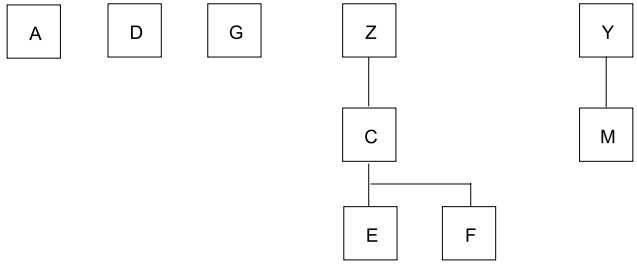


図 59. FLG_DELTREE_REL オプションの例

FLG_DELTREE_ALL オプションを使って、オブジェクト・インスタンス B とその下にあるいくつかのインスタンスがカタログから削除されました。オブジェクト・インスタンス C とそれが含むものは、 Z にも含まれているので、保持されています。

図60 は、FLG_DELTREE_ALL オプションを使って B が削除されたあとの情報カタログを表しています。

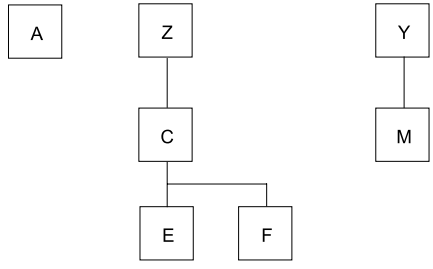


図 60. FLG_DELTREE_ALL オプションの例

117ページの図61 は、FLGDeleteTree 呼び出しを出すために必要な C 言語コードを示しています。このサンプル・コードは、DEPT001 Grouping 区分のオブジェクト・インスタンス、それに付加されたすべての Comments インスタンス、およびそれが参加するすべての ATTACHMENT、CONTACT、および LINK 関係を削除します。このサンプル・コードはまた、DEPT001 オブジェクト・インスタンスに含まれるすべてのオブジェクト・インスタンス、それらに付加されたすべての Comments インスタンス、およびそれらが参加するすべての ATTACHMENT、 CONTACT、および LINK 関係を削除します。

```
APIRET         rc;                    // Declare reason code
FLGOPTIONS   ulOptMask=0;
UCHAR          pszFLGID[FLG_ID_LEN + 1];
PFLGHEADERAREA  pDelStruct=NULL;
FLGEXTCODE   xc = 0;       // Declare extended code
  .
  .  set value for pszFLGID
  .
ulOptMask = ulOptMask | FLG_DELTREE_ALL;   // delete whole tree
rc = FLGDeleteTree (pszFLGID, ulOptMask,
                    &pDelStruct, &xc);
```

図 61. C 言語による FLGDeleteTree の呼び出しのサンプル

図62 は、この FLGDeleteTree 呼び出しのための出力構造を示しています。

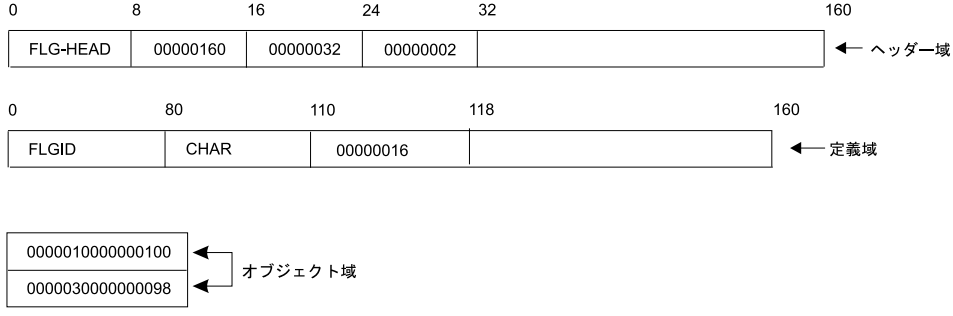


図 62. FLGDeleteTree の出力構造のサンプル

FLGDeleteType

ユーザー定義のオブジェクト・タイプを削除するために使用します。

Program および Attachment 区分以外のどの区分のオブジェクト・タイプでも削除することができます。情報カタログ・マネージャーは、情報カタログを作成中にこれらの区分を供給するからです。

許可

管理者

構文

```
APIRET  APIENTRY   FLGDeleteType( PSZ              pszObjTypeID,
                                  PFLGEXTCODE  pExtCode );
```

パラメーター

pszObjTypeID (PSZ) – 入力
削除するオブジェクト・タイプに関する 6 文字のシステム生成された固有の ID (オブジェクト・タイプ ID) を指します。

pExtCode (PFLGEXTCODE) – 出力
理由コードに関連する拡張コードを指します。戻された理由コードに関連した意味のある拡張コードがあるかどうかについては、285ページの『付録 D. 情報カタログ・マネージャーの理由コード』を参照してください。

理由コード (APIRET)
この API 呼び出しの実行結果を表します。

戻された理由コードの説明については、285ページの『付録D. 情報カタログ・マネージャーの理由コード』を参照してください。

使用方法

前提条件

入力パラメーターとして指定するオブジェクト・タイプ ID は、存在しているものでなければなりません。

このオブジェクト・タイプのオブジェクト・インスタンスが存在していてはなりません。このオブジェクト・タイプのインスタンスが存在している場合には、 FLGDeleteInst を使用してそれを削除してからでなければ、オブジェクト・タイプを削除することはできません。 FLGDeleteInst を使用して各イ

ンスタンスを個別に削除することも、タグ言語ファイルをインポートしてインスタンスを一括して削除することもできます。

情報カタログ内に自動的に作成された Programs オブジェクト・タイプは、削除することができません。 ただし、1 つまたは複数の Program インスタンスに関連するオブジェクト・タイプは削除することができます。 Program インスタンスは自動的に更新され、HANDLES 特性と PARMLIST 特性の値が消去されます。

情報カタログ内に自動的に作成された Comments オブジェクト・タイプは、削除することができません。

情報カタログ更新の制御

FLGDeleteType が正常に完了しない場合は、呼び出しを FLGRollback (225 ページの『FLGRollback』参照) に組み込む必要があります。
FLGDeleteType が正常に完了したあとで、FLGCommit を呼び出さずに、FLGDeleteReg への呼び出しが完了するまで待ってください。

例

図63 は、FLGDeleteType API 呼び出しを出すために必要な C 言語コードを示しています。このサンプル・コードは、情報カタログからオブジェクト・タイプを削除します

```
APIRET         rc;                    // Declare reason code
UCHAR           pszObjTypeID[FLG_OBJTYPID_LEN + 1];
FLGEXTCODE     ExtCode = 0;               // Declare extended code
  .
  .  /* Get the object type ID of MYIMAGE using FLGConvertID. */
  .
rc = FLGDeleteType (pszObjTypeID,
                    &ExtCode);
  .
  . // if (rc == 0)
  . //   rc = FLGDeleteReg (pszObjTypeID, &ExtCode);
  . // if (rc == 0)
  . //   rc = FLGCommit (&ExtCode);
  . // else
  . //   rc = FLGRollback (&ExtCode);
  .
```

図 63. C 言語による FLGDeleteType の呼び出しのサンプル

MYIMAGE のインスタンスが存在している場合には、それらを削除してからでなければ MYIMAGE オブジェクト・タイプを削除することはできません。管

理者のユーザー・インターフェースを使用して各インスタンスを個別に削除することも、タグ言語ファイルをインポートしてインスタンスを一括して削除することもできます。

特別なエラー処理

FLGDeleteType でデータベース・エラーが発生すると、情報カタログ・マネージャーは、プログラム内で最後に行われたコミット点にデータベースをロールバックします。

このロールバックが正常に行われた場合、FLGDeleteType は理由コード FLG_SEVERR_DB_AUTO_ROLLBACK_COMPLETE を戻します。拡張コードには、情報カタログ・マネージャーがデータベースのロールバックを行う原因となったデータベース・エラーに関する SQL コードが含まれています。

考慮事項: このロールバックが失敗した場合には、FLGDeleteType は理由コード FLG_SEVERR_DB_AUTO_ROLLBACK_FAIL を戻します。拡張コードには、情報カタログ・マネージャーがデータベースのロールバックを行う原因となったデータベース・エラーに関する SQL コードが含まれています。この場合、データベースの保全性に関する重大な問題が発生している可能性があるため、プログラムで FLGTerm を呼び出して情報カタログ・マネージャーを終了させる必要があります。

データベースの状態によっては、バックアップ用データベース・ファイルを使用してデータベースを回復させなければならないことがあります。 情報カタログ・データベースの回復に関する詳細については、*情報カタログ・マネージャー 管理の手引き* を参照してください。

FLGDeleteType エラーとは無関係なコミットされていない変更内容が情報カタログ・マネージャーによって除去されないようにするために、プログラム内のこの呼び出しの直前に FLGCommit 呼び出しを組み込んでください。

FLGDeleteTypeExt

ユーザー定義のオブジェクト・タイプおよびそのオブジェクト・タイプのインスタンス、それらのインスタンスに付加された Comments オブジェクト、およびそれらのインスタンスが参加している関係を削除します。オブジェクト・タイプ登録の削除もします。

Program および Attachment 区分以外のどの区分のオブジェクト・タイプでも削除することができます。情報カタログ・マネージャーは、情報カタログを作成中にこれらの区分を供給するからです。

許可

管理者

構文

```
APIRET  APIENTRY   FLGDeleteTypeExt( PSZ             pszObjTypeID,
                                     PFLGEXTCODE  pExtCode );
```

パラメーター

pszObjTypeID (PSZ) – 入力
削除するオブジェクト・タイプに関する 6 文字のシステム生成された固有の ID (オブジェクト・タイプ ID) を指します。

pExtCode (PFLGEXTCODE) – 出力
理由コードに関連する拡張コードを指します。戻された理由コードに関連した意味のある拡張コードがあるかどうかについては、285ページの『付録D. 情報カタログ・マネージャーの理由コード』を参照してください。

理由コード (APIRET)
この API 呼び出しの実行結果を表します。

戻された理由コードの説明については、285ページの『付録D. 情報カタログ・マネージャーの理由コード』を参照してください。

使用方法

前提条件

入力パラメーターとして指定するオブジェクト・タイプ ID は、存在しているものでなければなりません。

制約事項

FLGDeleteTypeExt は、異なったオブジェクト・タイプを持つオブジェクトのインスタンスを含む Grouping 区分のオブジェクト・インスタンスは削除しません。このような Grouping 区分のインスタンスが存在する場合は、オブジェクト・タイプを削除するまえに FLGDeleteTree を使って、それらを削除しなくてはなりません。

情報カタログ内に自動的に作成された Programs オブジェクト・タイプは、削除することができません。ただし、1 つまたは複数の Program インスタンスに関連するオブジェクト・タイプは削除することができます。 Program インスタンスは自動的に更新され、HANDLES 特性と PARMLIST 特性の値が消去されます。

情報カタログ内に自動的に作成された Comments オブジェクト・タイプは、削除することができません。

情報カタログ更新の制御

FLGDeleteTypeExt は、オブジェクト・タイプと共に、オブジェクト・タイプのすべてのインスタンスを削除してしまうので、FLGDeleteTypeExt を呼び出すまえに、あるタイプのオブジェクトを検索して、削除したいオブジェクト・タイプの存在するオブジェクトのどれも保存しなくてよいことを確認したほうがよいでしょう。

プログラムを可能な限り、情報カタログと同期化させるために、FLGDeleteTypeExt が正常に完了したあとで、呼び出しを FLGCommit (82ページの『FLGCommit』参照) に組み込む必要があります。FLGDeleteTypeExt が正常に完了しない場合は、呼び出しを FLGRollback (225ページの『FLGRollback』参照) に組み込む必要があります。

例

123ページの図64 は、FLGDeleteTypeExt 呼び出しを出すために必要な C 言語コードを示しています。このサンプル・コードは情報カタログから、MYIMAGE オブジェクト・タイプ、 MYIMAGE オブジェクト・タイプのすべてのインスタンス、 MYIMAGE オブジェクト・タイプのインスタンスに付加されたされたすべてのコメント、 MYIMAGE インスタンスが参加するすべての関係、および MYIMAGE オブジェクト・タイプの登録を削除します。

```
APIRET         rc;                    // Declare reason code
UCHAR        pszTypeID[FLG_OBJTYPID_LEN+1];
FLGEXTCODE   xc = 0;       // Declare extended code
  .
  .  /* processing */
  .
rc = FLGDeleteTypeExt (pszTypeID,
                       &xc);
```

図 64. C 言語による FLGDeleteTypeExt の呼び出しのサンプル

特別なエラー処理

FLGDeleteTypeExt でデータベース・エラーが発生すると、情報カタログ・マネージャーは、プログラム内で最後に行われたコミット点にデータベースをロールバックします。

このロールバックが正常に行われた場合、FLGDeleteTypeExt は理由コード FLG_SEVERR_DB_AUTO_ROLLBACK_COMPLETE を戻します。拡張コードには、情報カタログ・マネージャーがデータベースのロールバックを行う原因となったデータベース・エラーに関する SQL コードが含まれています。

考慮事項: このロールバックが失敗した場合には、FLGDeleteTypeExt は理由コード FLG_SEVERR_DB_AUTO_ROLLBACK_FAIL を戻します。拡張コードには、情報カタログ・マネージャーがデータベースのロールバックを行う原因となったデータベース・エラーに関する SQL コードが含まれています。この場合、データベースの保全性に関する重大な問題が発生している可能性があるため、プログラムで FLGTerm を呼び出して情報カタログ・マネージャーを終了させる必要があります。

データベースの状態によっては、バックアップ用データベース・ファイルを使用してデータベースを回復させなければならないことがあります。情報カタログ・データベースの回復に関する詳細については、*情報カタログ・マネージャー 管理の手引き* を参照してください。

FLGDeleteTypeExt エラーとは無関係なコミットされていない変更内容が情報カタログ・マネージャーによって除去されないようにするために、プログラム内のこの呼び出しの直前に FLGCommit 呼び出しを組み込んでください。

FLGExport

情報カタログからメタデータを検索してファイル内のタグ言語に変換するために使用します。

許可

管理者または許可ユーザー

構文

```
APIRET  APIENTRY   FLGExport( PSZ              pszTagFileID,
                                  PSZ              pszLogFileID,
                              PSZ              pszIcoPath,
                                   PFLGHEADERAREA   pListStruct,
                                  PFLGEXTCODE  pExtCode );
```

パラメーター

pszTagFileID (PSZ) − 入力
出力タグ言語ファイルの名前を指します。このパラメーターは必須パラメーターです。

このパラメーターにはドライブ、ディレクトリー・パス、およびファイル名が含まれており、ファイル割り振りテーブル (FAT) または HPFS ファイルとして有効な値でなければなりません。 このファイルのターゲット・ドライブは、固定ディスクであっても、取り外し可能ディスクであっても差し支えありません。 ファイル名だけを入力すると、情報カタログ・マネージャー側は、タグ言語ファイルを DGWPATH 環境変数で示されるドライブおよびパスに配置します。

このターゲット・タグ言語ファイルが存在していてはなりません。情報カタログ・マネージャーは既存のタグ・ファイルへの上書きは行いません。

(ドライブおよびディレクトリーを除く) ファイル名とエクステンションは、 240 文字を超えてはなりません。タグ言語ファイル ID が全体で 259 文字を超えてはなりません。

pszLogFileID (PSZ) − 入力
ログ・ファイルの名前を指します。このパラメーターは必須パラメーターです。

このパラメーターにはドライブ、ディレクトリー・パス、およびファイル名が含まれており、FAT または HPFS ファイルとして有効な値でなければ

なりません。 ログ・ファイルのターゲット・ドライブは、固定ディスクでなければなりません。ログ・ファイル ID は 259 文字を超えてはなりません。ファイル名だけを指定すると、情報カタログ・マネージャー側は、ログ・ファイルを DGWPATH 環境変数で示されるドライブおよびパスに配置します。

このパラメーターで指定されたログ・ファイルが存在していない場合には、新しいファイルが作成されます。 このパラメーターで指定されたログ・ファイルがすでに存在している場合には、 FLGExport API 呼び出しはそれに対して行われます。

pszIcoPath (PSZ) − 入力

OS/2® または Windows のアイコン・ファイルを含むパスの仕様を指します。

このパラメーターは任意選択です。このパラメーターが NULL の場合、アイコン・ファイルのエクスポートは行われません。

このパラメーターにはドライブおよびディレクトリーが含まれており、FAT または HPFS ファイルとして有効な値でなければなりません。このパラメーターは、246 文字を超えてはなりません。

このパラメーターを指定する場合、アイコン・ファイルのターゲット・ドライブは固定ディスクでなければなりません。

pListStruct (PFLGHEADERAREA) − 入力

エクスポートされるオブジェクトのリストとエクスポート・オプションを含んでいる入力構造を指します。

pExtCode (PFLGEXTCODE) − 出力

理由コードに関連する拡張コードを指します。戻された理由コードに関連した意味のある拡張コードがあるかどうかについては、285ページの『付録 D. 情報カタログ・マネージャーの理由コード』を参照してください。

理由コード (APIRET)

この API 呼び出しの実行結果を表します。

戻された理由コードの説明については、285ページの『付録D. 情報カタログ・マネージャーの理由コード』を参照してください。

入力構造

FLGExport を使用するには、126ページの図65 に示す入力構造を定義しなければなりません。この構造はヘッダー域、定義域、およびオブジェクト域からなります。

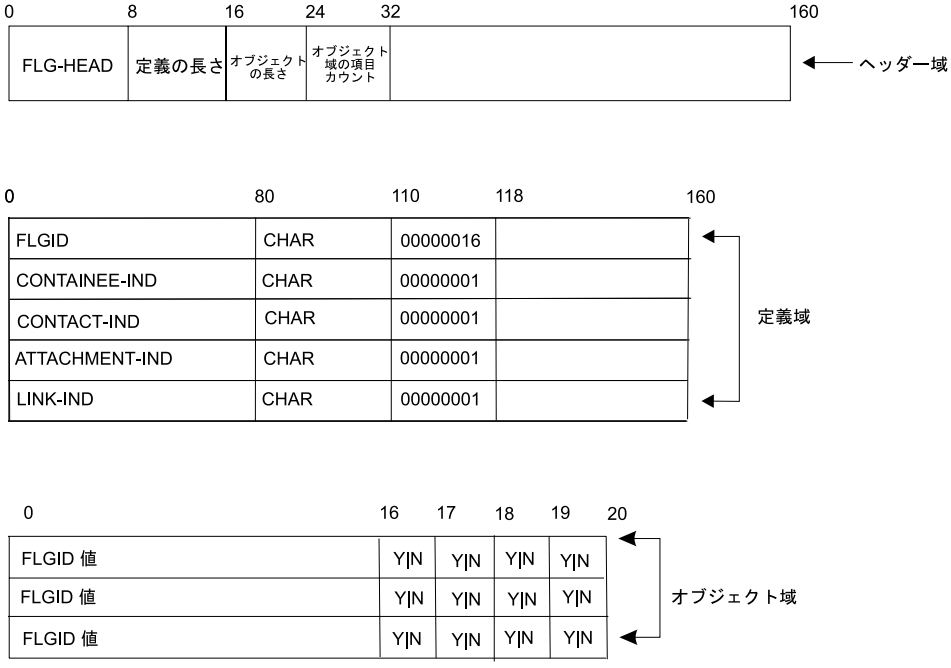


図 65. FLGExport 入力構造

バイト・オフセットの意味については、36ページの『情報カタログ・マネージャー API 入力構造』を参照してください。

使用方法

入力構造

FLGExport 入力構造の定義域は、図65 に示すとおりに指定しなければなりません。

FLGExport の入力構造には、以下の情報が入ります。

FLGID エクスポートされるインスタンスの 16 文字のシステム生成された固有の識別子。

この ID の 1 文字目から 6 文字目までは、このインスタンスのオブジェクト・タイプを識別します。

この ID の 7 文字目から 16 文字目までは、システム生成された固有のインスタンス ID です。

任意の情報カタログ・マネージャー・オブジェクト・インスタンスをエクスポートすることができます。

CONTAINEE-IND

情報カタログ・マネージャーがこのオブジェクトに含まれるすべてのオブジェクトをエクスポートするのかどうかを指定する、1 文字のインディケーター (Y | N) です。このインディケーターは Grouping オブジェクトだけに適用され、それ以外のすべてのオブジェクトについては無視されます。

CONTACT-IND

情報カタログ・マネージャーが Grouping および Elemental オブジェクトの関連 Contact オブジェクトをすべてエクスポートするのかどうかを指定する、1 文字のインディケーター (Y | N) です。このインディケーターは Grouping および Elemental オブジェクトだけに適用され、それ以外のすべてのオブジェクトについては無視されます。

ATTACHMENT-IND

情報カタログ・マネージャーが指定されたオブジェクト・インスタンスに付加された Attachment オブジェクトをすべてエクスポートするのかどうかを指定する、1 文字のインディケーター (Y | N) です。このインディケーターは、指定されたオブジェクトが Attachment オブジェクトであると、無視されます。

LINK-IND

情報カタログ・マネージャーが指定されたオブジェクト・インスタンスにリンクしたすべての Grouping および Elemental オブジェクト・インスタンスをすべてエクスポートするのかどうかを指定する、1 文字のインディケーター (Y | N) です。このインディケーターは Grouping および Elemental オブジェクトだけに適用され、それ以外のすべてのオブジェクトについては無視されます。

生成されるタグ言語ファイル

FLGExport は、エクスポートされる各オブジェクト・インスタンスのタグを含むタグ言語ファイルを生成します。インディケーターに何を指定するかによって、表16 に示されているようにオブジェクト・インスタンスがエクスポートされます。

表 16. インディケーターの組み合わせによってタグ言語ファイルにエクスポートされるオブジェクト・インスタンス

インディケーターの値				
CONTAINEE	**CONTACT**	**ATTACHMENT**	**LINK**	**エクスポート:**
Y	Y	Y	Y	a から j

表 16. インディケーターの組み合わせによってタグ言語ファイルにエクスポートされるオブジェクト・インスタンス (続き)

インディケーターの値				
CONTAINEE	**CONTACT**	**ATTACHMENT**	**LINK**	**エクスポート:**
Y	Y	Y	N	a、b、c、d、g、h、i、j
Y	Y	N	Y	a、b、e、f、g、h
Y	Y	N	N	a、b、g、h
Y	N	Y	Y	a、b、c、d、e、f
Y	N	Y	N	a、b、c、d
Y	N	N	N	a、b
Y	N	N	Y	a、b、e、f
N	Y	Y	Y	a、c、e、g、i
N	Y	Y	N	a、c、g、i
N	Y	N	Y	a、e、g
N	N	Y	Y	a、c、e
N	N	Y	N	a、c
N	N	N	Y	a、e
N	N	N	N	a のみ

注:

a 指定されたオブジェクト・インスタンス
b a に含まれるオブジェクト・インスタンス
c a に付加されたコメント
d b に付加されたコメント
e a へのリンク
f b へのリンク
g a への連絡
h b への連絡
i g に付加されたコメント
j h に付加されたコメント

FLGExport は、頻繁に使用される COMMIT タグをタグ言語ファイル内に生成します。

FLGExport は、各オブジェクト・タイプに関連付けられたアイコンのコピーを、指定されたアイコン・パスに収めます。このオブジェクト・タイプに別のアイコンが関連付けられていない場合、FLGExport はデフォルトの区

分をエクスポートしません。エクスポートされたアイコン・ファイルの名前は、オブジェクト・タイプの DP 名 (省略名) に、OS/2 アイコンの場合は .ICO、Windows アイコンの場合は .ICW のエクステンションが付いたものになります。

メタデータをディスケットにエクスポートする際の VisualAge C++ プログラムの連係

C 言語プログラムによって情報カタログ・マネージャー情報をディスケットにエクスポートする FLGExport 呼び出しが出される場合には、情報カタログ・マネージャーがプレゼンテーション・マネージャー® (PM) のインターフェース表示メッセージを使用して、必要なときにディスケットを挿入するようにユーザーに指示できるようにするために、そのプログラムを WINDOWAPI タイプのアプリケーションと連係させてください。

この連係は、次のいずれかの方法を使用して行うことができます。

– 次の連係ステートメント

ilink /NOFREE /PMTYPE:vio /NOI *filename***.obj,,,dgwapi.lib,,**

– モジュール定義ファイル。 NAME ステートメントで *apptype* として **WINDOWAPI** を指定してください。

例

130ページの図66 は、FLGExport API 呼び出しを出すために必要な C 言語コードを示しています。このサンプル・コードは 3 つの情報カタログ・マネージャー・オブジェクトをエクスポートします。これらの 3 つのオブジェクトは、すべて Grouping オブジェクトです。

- 最初のオブジェクト、それに含まれるすべてのオブジェクト、およびその Contact オブジェクトがエクスポートされます。
- 2 番目のオブジェクト、それに含まれるすべてのオブジェクト、およびそれに付加された Comments オブジェクトがエクスポートされます。
- 3 番目のオブジェクトのエクスポートでは、それに含まれるオブジェクトはエクスポートされません。

```
APIRET         rc;                    // Declare reason code
UCHAR   pszTagFileID[FLG_TAG_FILE_ID_MAXLEN + 1]; // Tag file id
UCHAR   pszLogFileID[FLG_LOG_FILE_ID_MAXLEN + 1]; // Log file id
UCHAR   pszIcoPath[FLG_ICON_PATH_MAXLEN + 1];  // icon files path
PFLGHEADERAREA  pListStruct;  // pointer to the input structure
FLGEXTCODE     ExtCode=0;   // declare an extended code for API
  .
  .  /* set values for Tag file/ Log file/ Icon path   */
  .  /* create object list             */
  .
rc  = FLGExport (pszTagFileID,
                 pszLogFileID,
                  pszIcoPath,
                  pListStruct,          // Pass input structure
                    &ExtCode);             // Pass pointer to extended code
```

図 66. C 言語による FLGExport の呼び出しのサンプル

図67 は、この FLGExport 呼び出しのための入力構造を示しています。

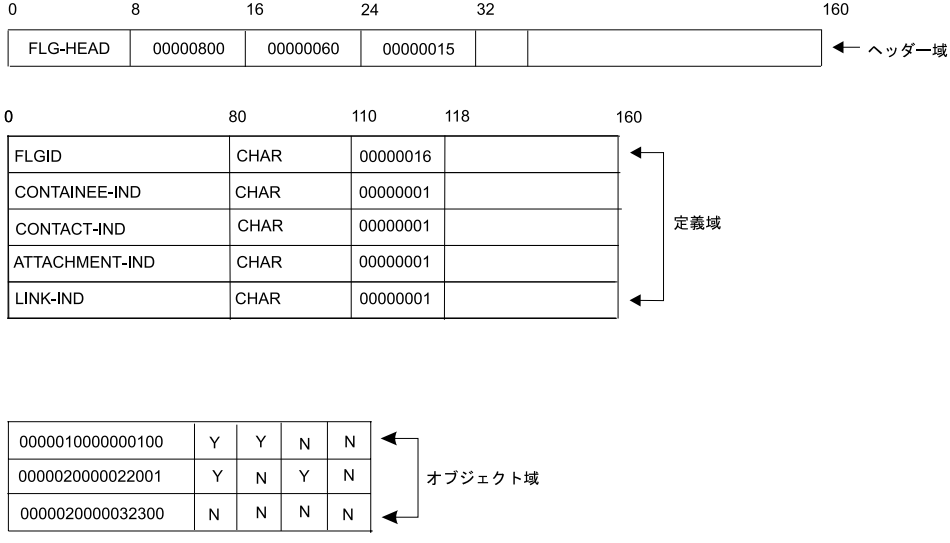


図 67. FLGExport のための入力構造のサンプル

FLGFoundIn

指定されたインスタンスが検出されたオブジェクト・インスタンスまたはオブジェクト・タイプのリストを検索します。 FLGFoundIn は、以下の検索をすることができます。

- 指定されたオブジェクト・インスタンスを含む Grouping オブジェクト・インスタンス
- 指定されたオブジェクト・インスタンスが Contact であるオブジェクト・インスタンス
- 指定されたオブジェクト・インスタンスが Comments オブジェクトであるオブジェクト・インスタンス
- 指定された Programs オブジェクト・インスタンスが関連するオブジェクト・タイプ

許可

管理者またはユーザー

構文

```
APIRET  APIENTRY   FLGFoundIn( PSZ              pszFLGID,
                               FLGOPTIONS   Options,
                              PFLGHEADERAREA * ppListStruct,
                               PFLGEXTCODE  pExtCode );
```

パラメーター

pszFLGID (PSZ) − 入力

その親のリストが検索されるオブジェクト・インスタンスの 16 文字のオブジェクト・インスタンス ID (FLGID) を指します。

この ID の 1 文字目から 6 文字目までは、このインスタンスのオブジェクト・タイプを識別します。

この ID の 7 文字目から 16 文字目までは、システム生成された固有のインスタンス ID です。

指定する FLGID は、何をリストしたいかによります。

Attachment

Attachment 区分のオブジェクト・インスタンスの FLGID (指定さ

れたオブジェクト・インスタンスが Comments オブジェクトとして付加されたオブジェクト・インスタンスを検索します。)

Contact

Contact 区分のオブジェクト・インスタンスの FLGID (指定されたオブジェクト・インスタンスが Contact であるオブジェクト・インスタンスを検索します。)

Containees

Elemental または Grouping 区分のオブジェクト・インスタンスの FLGID (指定されたオブジェクト・インスタンスを含む Grouping オブジェクト・インスタンスを検索します。)

Program

Program 区分のオブジェクト・インスタンスの FLGID (指定された Programs オブジェクト・インスタンスが関連するオブジェクト・タイプを検索します。)

Options (FLGOPTIONS) − 入力

以下のいずれかのオプションを選択します。

FLG_LIST_ATTACHMENT

指定されたオブジェクト・インスタンスが Comments オブジェクトとして付加されたオブジェクト・インスタンスを検索します。

FLG_LIST_CONTACT

指定されたオブジェクト・インスタンスが Contact であるオブジェクト・インスタンスを検索します。

FLG_LIST_CONTAIN

指定されたオブジェクト・インスタンスを含む Grouping オブジェクト・インスタンスを検索します。

FLG_LIST_PROGRAM

指定された Programs オブジェクト・インスタンスが関連するオブジェクト・タイプを検索します。

ppListStruct (PFLGHEADERAREA) − 出力

指定されたインスタンスが検出されたオブジェクト・インスタンスまたはオブジェクト・タイプのリストを含む出力構造に、ポインターのアドレスを指します。出力構造がない場合、出力構造へのポインターは NULL にセットされます。

それぞれの Contain、Contact、または Attachment 関係に、出力構造は、『found-in』オブジェクト・インスタンスに関する以下の情報を含みます。

- FLGID (16 文字)
- 名前 (80 文字)

すべてのインスタンスは、使用するデータベース管理システムの照合順序に従って、オブジェクト・タイプ名、オブジェクト・インスタンス名の昇順で分類されます。

各 Program 関連について、出力構造は『found-in』オブジェクト・タイプに関して以下の情報を含んでいます。

- オブジェクト・タイプ ID (6 文字)
- オブジェクト・タイプの 80 文字の外部名 (EXTERNAL NAME OF OBJ TYPE)

すべてのオブジェクト・タイプは、使用するデータベース管理システムの照合順序に従って、オブジェクト・タイプの 80 文字の外部名 (EXTERNAL NAME OF OBJ TYPE) の昇順で分類されます。

FLGFoundIn によって戻すことのできるオブジェクト・インスタンスまたはオブジェクト・タイプの最大数は、5000 です。

pExtCode (PFLGEXTCODE) － 出力

理由コードに関連する拡張コードを指します。戻された理由コードに関連した意味のある拡張コードがあるかどうかについては、285ページの『付録 D. 情報カタログ・マネージャーの理由コード』を参照してください。

理由コード (APIRET)

この API 呼び出しの実行結果を表します。

戻された理由コードの説明については、285ページの『付録D. 情報カタログ・マネージャーの理由コード』を参照してください。

使用方法

出力構造に割り振られたメモリーの解放

FLGFoundIn が出力構造にあるデータを戻した場合、出力構造に戻されたデータを保管してから、FLGFreeMem を呼び出さなくてはなりません (137ページの『FLGFreeMem』参照)。メモリーを解放するのに、たとえば C 言語指示のような他の方式は使用しないでください。

例

このサンプル・コードは、特定の Contact オブジェクトが検出されたオブジェクト・インスタンスのリストを検索します。 134ページの図68 は、FLGFoundIn 呼び出しを出すために必要な C 言語コードを示しています。

```
APIRET            rc;                // reason code from FLGFoundIn
UCHAR             pszInstID[FLG_ID_LEN + 1];
FLGOPTIONS        Option=0;          // association type
PFLGHEADERAREA  * ppReturnObjList;   // pointer to output structure ptr
FLGEXTCODE       xc=0;             // extended code
  .
  .  /* provide values for input parameters */
  .
Option = Option | FLG_LIST_CONTACT;
rc = FLGFoundIn (pszInstID,
                     Option,
                     ppReturnObjList,
                     &xc);
```

図 68. C 言語による FLGFoundIn の呼び出しのサンプル

135ページの図69 は、FLGFoundIn 呼び出しのための出力構造を表示しています。

図 69. FLGFoundIn の出力構造のサンプル

このサンプル・コードは、指定された Programs オブジェクト・インスタンスによりハンドルされたオブジェクト・タイプのリストを検索します。 136ページの図70 は、FLGFoundIn 呼び出しを出すために必要な C 言語コードを示しています。

```
APIRET            rc;                // reason code from FLGFoundIn
UCHAR             pszInstID[FLG_ID_LEN + 1];
FLGOPTIONS        Option=0;          // association type
PFLGHEADERAREA  * ppReturnObjList;   // pointer to output structure ptr
FLGEXTCODE       xc=0;             // extended code
  .
  .  /* provide values for input parameters */
  .
Option = Option | FLG_LIST_PROGRAM;
rc = FLGFoundIn (pszInstID,
                 Option,
                 ppReturnObjList,
                 &xc);
```

図 70. C 言語による FLGFoundIn の呼び出しのサンプル

図71 は、FLGFoundIn 呼び出しのための出力構造を表示しています。

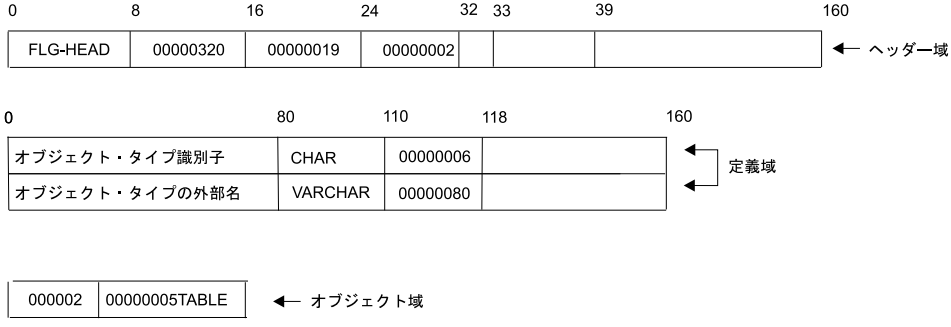


図 71. FLGFoundIn の出力構造のサンプル

FLGFreeMem

FLGListObjTypes や FLGNavigate などの情報カタログ・マネージャー API 呼び出しで作成された出力構造に割り振られたメモリーを解放するために使用します。

許可

管理者またはユーザー

構文

```
APIRET  APIENTRY    FLGFreeMem( PFLGHEADERAREA   pFLGOutputStruct,
                                PFLGEXTCODE  pExtCode );
```

パラメーター

pFLGOutputStruct (PFLGHEADERAREA) – 入力

割り振り解除する情報カタログ・マネージャー出力構造を指します。

出力構造を作成する API 呼び出しを出す場合には、情報カタログ・マネージャーによって生成されて PFLGHEADERAREA データ・タイプで示されるアドレスに保管される、出力構造を指すポインターの値を保管する必要があります。割り振られたメモリーを解放するために、このポインターをパラメーターとして FLGFreeMem に渡すことができるようにするためです。

FLGFreeMem は、情報カタログ・マネージャー API 呼び出しで作成された出力構造だけに作用します。

pExtCode (PFLGEXTCODE) – 出力

理由コードに関連する拡張コードを指します。戻された理由コードに関連した意味のある拡張コードがあるかどうかについては、285ページの『付録D. 情報カタログ・マネージャーの理由コード』を参照してください。

理由コード (APIRET)

この API 呼び出しの実行結果を表します。

戻された理由コードの説明については、285ページの『付録D. 情報カタログ・マネージャーの理由コード』を参照してください。

例

図72 は、FLGFreeMem API 呼び出しを出すために必要な C 言語コードを示しています。このサンプル・コードは、メモリー内の情報カタログ・マネージャー出力構造を解放します。

```
PFLGHEADERAREA pFLGOutputStruct; // pointer to the FLG output structure
APIRET         rc;                        // reason code
FLGEXTCODE     ExtCode = 0;                    // Extended code
rc = FLGFreeMem ( pFLGOutputStruct, &ExtCode );
```

図 72. C 言語による FLGFreeMem の呼び出しのサンプル

FLGGetInst

指定されたオブジェクト・タイプに関する単一のオブジェクト・インスタンスを検索するために使用します。

許可

管理者またはユーザー

構文

```
APIRET  APIENTRY     FLGGetInst( PSZ              pszFLGID,
                                 PFLGHEADERAREA * ppObjInstStruct,
                                  PFLGEXTCODE  pExtCode );
```

パラメーター

pszFLGID (PSZ) − 入力
検索するインスタンスを表す 16 文字のシステム生成された固有の ID を指します。

この ID の 1 文字目から 6 文字目までは、このインスタンスのオブジェクト・タイプを識別します。

この ID の 7 文字目から 16 文字目までは、システム生成された固有のインスタンス ID です。

ppObjInstStruct (PFLGHEADERAREA) − 出力
出力構造を指すポインターのアドレスを指します。 FLGGetInst が失敗した場合には、このポインターは NULL にセットされます。

pExtCode (PFLGEXTCODE) − 出力
理由コードに関連する拡張コードを指します。戻された理由コードに関連した意味のある拡張コードがあるかどうかについては、285ページの『付録D. 情報カタログ・マネージャーの理由コード』を参照してください。

理由コード (APIRET)
この API 呼び出しの実行結果を表します。

戻された理由コードの説明については、285ページの『付録D. 情報カタログ・マネージャーの理由コード』を参照してください。

出力構造

FLGGetInst が作成する出力構造には、図73 に示すように、要求されたオブジェクト・インスタンスの特性の仕様および値が含まれています。

この出力構造のオブジェクト域には、要求されたオブジェクト・インスタンスの特性の値が入ります。

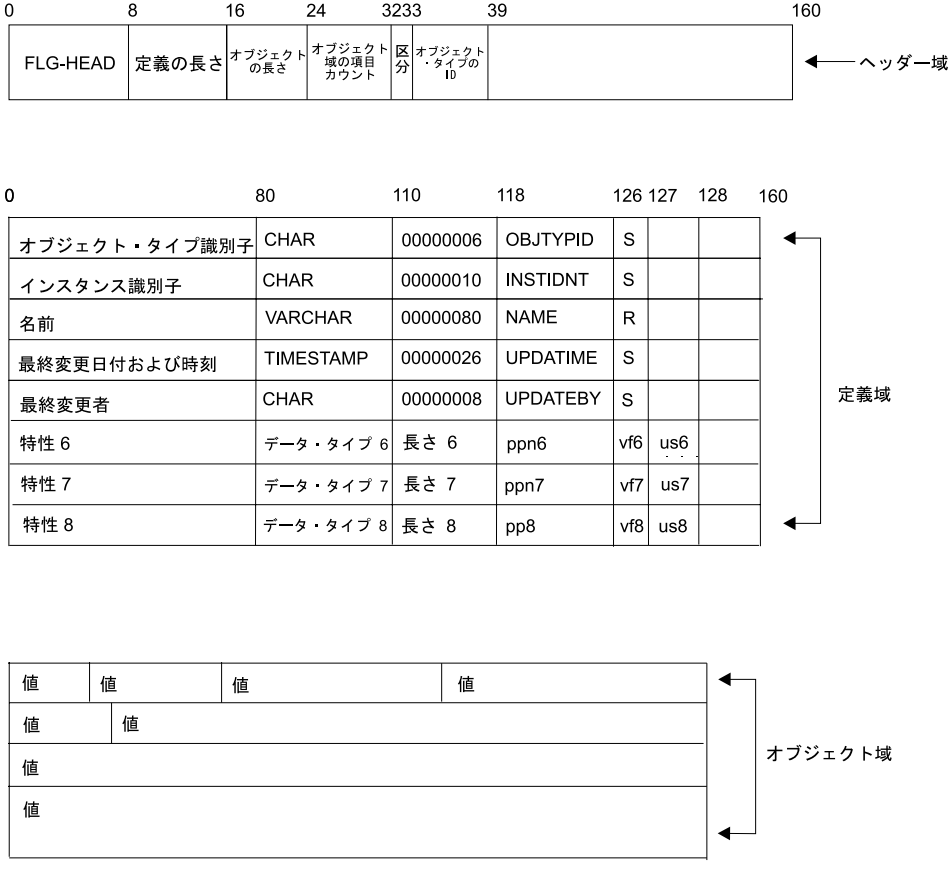


図 73. FLGGetInst の出力構造

バイト・オフセットの意味については、56ページの『情報カタログ・マネージャーの API 出力構造』を参照してください。

使用方法

前提条件

pszFLGID 入力パラメーターの値は、既存のオブジェクト・インスタンスを参照するものでなければなりません。

出力構造に割り振られたメモリーの解放

FLGGetInst が出力構造にあるデータを戻した場合、出力構造に戻されたデータを保管してから、FLGFreeMem を呼び出さなくてはなりません (137ページの『FLGFreeMem』参照)。メモリーを解放するのに、たとえば C 言語指示のような他の方式は使用しないでください。

情報カタログ更新の制御

FLGGetInst はデータベースに対して変更をコミットします。 FLGGetInst 呼び出しの前に行われた予期しない変更が情報カタログ・マネージャーによってコミットされないようにするために、プログラムの中で、FLGGetInst を出す前に FLGCommit または FLGRollback を出すようにしてください。

例

図74 は、FLGGetInst API 呼び出しを出すために必要な C 言語コードを示しています。このサンプル・コードは、Quality Group オブジェクト・インスタンスに関する情報を検索します。

```
APIRET         rc;                    // Declare reason code
UCHAR          pszFLGID[FLG_ID_LEN+1];   // Unique ID for "Quality Group"
PFLGHEADERAREA * ppObjInstStruct;        // Pointer to the output structure
FLGEXTCODE     ExtCode = 0;               // Declare extended code
  .
  .  /* Retrieving an object Instance  */
  .
strcpy (pszFLGID,"0000330000001234");
rc = FLGGetInst  (pszFLGID,        // Instance ID
                  ppObjInstStruct, // Structure pointer where output will be returned
                   &ExtCode);             // Pass pointer to extended code
```

図 74. C 言語による FLGGetInst の呼び出しのサンプル

142ページの図75 は、このオブジェクト・インスタンスの特性と値の情報が入っている出力構造を示しています。

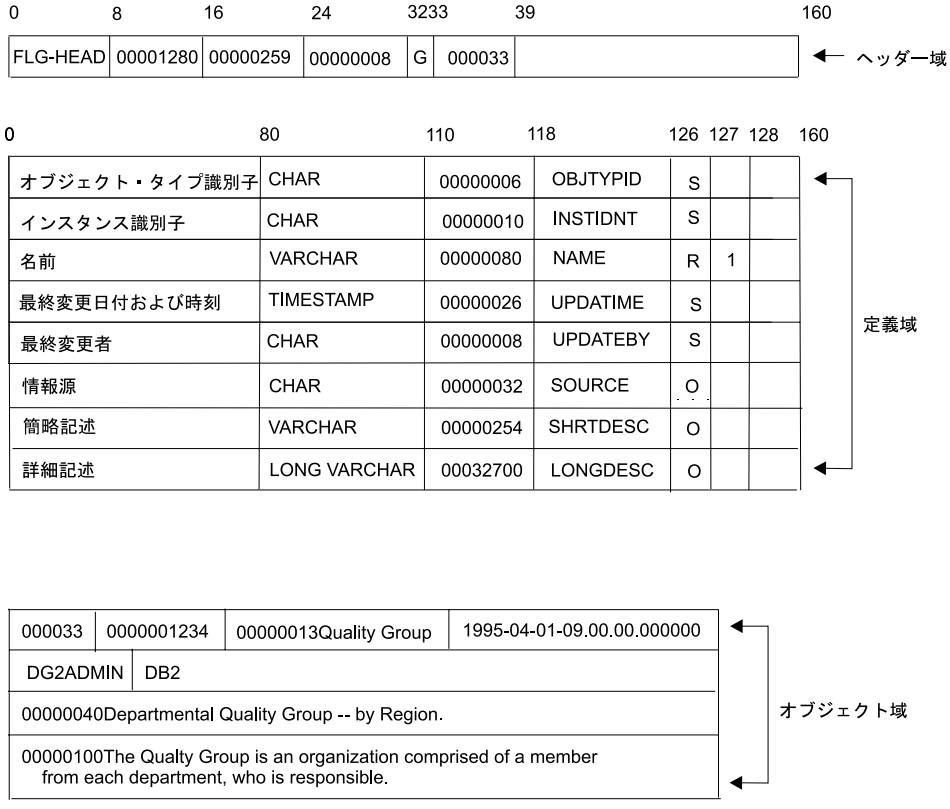


図 75. FLGGetInst のための出力構造のサンプル

FLGGetReg

指定されたオブジェクト・タイプに関する登録情報を情報カタログから検索するために使用します。

許可

管理者またはユーザー

構文

```
APIRET  APIENTRY   FLGGetReg( PSZ              pszObjTypeID,
                              PSZ                 pszIconFileID,
                           PFLGHEADERAREA * ppObjRegStruct,
                              PFLGEXTCODE  pExtCode );
```

パラメーター

pszObjTypeID (PSZ) − 入力
登録を削除するオブジェクト・タイプを表す 6 文字のシステム生成された固有の識別子 (オブジェクト・タイプ ID) を指します。

pszIconFileID (PSZ) − 入出力
入力に使用する場合には、登録されたオブジェクト・タイプの OS/2 アイコンを戻すために使用するファイルのファイル・パスと名前を指します。このパラメーターが NULL の場合、情報カタログ・マネージャーは登録されたオブジェクト・タイプのアイコンを検索しません。

出力に使用する場合には、登録されたオブジェクト・タイプの OS/2 アイコンを保管するために情報カタログ・マネージャーが使用するファイルのファイル・パスと名前を指します。オブジェクト・タイプ登録に関連付けられたアイコンがない場合には、このポインターは NULL にセットされます。

ppObjRegStruct (PFLGHEADERAREA) − 出力
出力構造を指すポインターのアドレスを指します。

この出力構造には、要求されたオブジェクト・タイプの登録情報に関する特性の仕様と値が入ります。 FLGGetReg が失敗した場合には、このポインターは NULL にセットされます。

pExtCode (PFLGEXTCODE) − 出力
理由コードに関連する拡張コードを指します。戻された理由コードに関連

した意味のある拡張コードがあるかどうかについては、285ページの『付録 D. 情報カタログ・マネージャーの理由コード』を参照してください。

理由コード (APIRET)

この API 呼び出しの実行結果を表します。

戻された理由コードの説明については、285ページの『付録D. 情報カタログ・マネージャーの理由コード』を参照してください。

出力構造

FLGGetReg が作成する出力構造には、図76 に示すように、要求されたオブジェクト・タイプ登録の特性の仕様および値が含まれています。

この出力構造のオブジェクト域には、要求されたオブジェクト・タイプの登録特性の値が入ります。

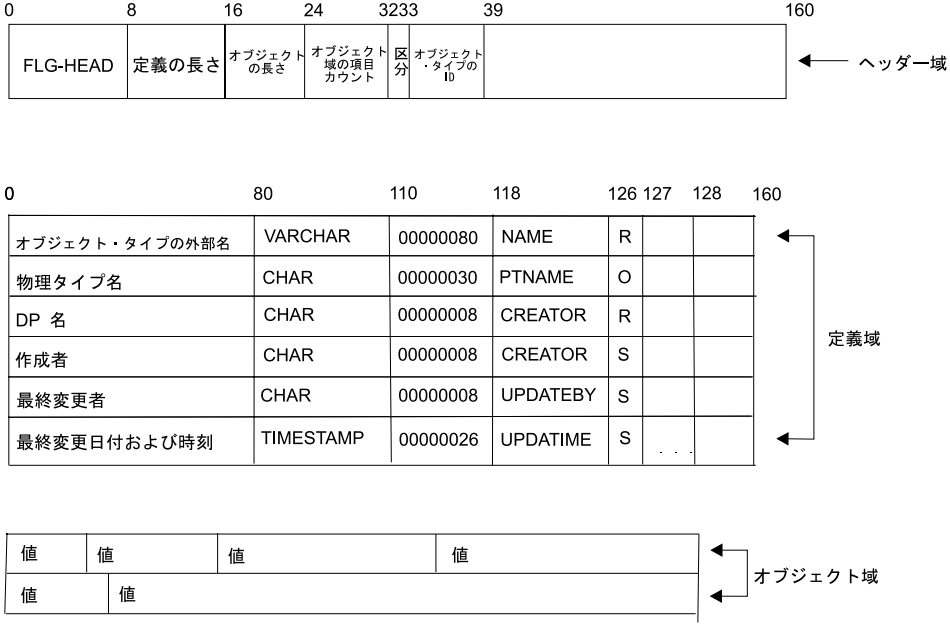


図 76. FLGGetReg 出力構造

バイト・オフセットの意味については、56ページの『情報カタログ・マネージャーの API 出力構造』を参照してください。

使用方法

制約事項

FLGGetReg を使うと、OS/2 アイコンしか検索できません。 Windows アイコンを検索するには、FLGManageIcons を使用します (198ページの『FLGManageIcons』を参照)。

前提条件

pszObjTypeID パラメーターの値は、情報カタログに登録された既存のオブジェクト・タイプ ID を参照するものでなければなりません。

出力構造に割り振られたメモリーの解放

FLGGetReg が出力構造にあるデータを戻した場合、出力構造に戻されたデータを保管してから、FLGFreeMem を呼び出さなくてはなりません (137ページの『FLGFreeMem』参照)。メモリーを解放するのに、たとえば C 言語指示のような他の方式は使用しないでください。

例

図77 は、FLGGetReg API 呼び出しを出すために必要な C 言語コードを示しています。このサンプル・コードは、MYIMAGE オブジェクト・タイプに関する登録情報を情報カタログから検索します。

```
APIRET         rc;                    // Declare reason code
UCHAR          pszObjTypeID[FLG_OBJTYPID_LEN+1]; // Unique ID for MYIMAGE ObjType
UCHAR          pszIconFileID[FLG_ICON_FILE_ID_MAXLEN+1]; // Path/File name for ICON
PFLGHEADERAREA * ppObjRegStruct;      // Ptr to pointer to the output structure
FLGEXTCODE     ExtCode = 0;                // Declare extended code
  .
  .  /* Retrieving an object Type Registration Instance  */
  .
strcpy (pszObjTypeID,"000044");
strcpy (pszIconFileID,"Y:¥¥FLGICON2.ICO");
rc = FLGGetReg    (pszObjTypeID, // id of the object type
                      pszIconFileID,// Path/File name of file to contain ICON
                      ppObjRegStruct,// Structure pointer where out  put will be returned
                    &ExtCode);             // Pass pointer to extended code
```

図 77. C 言語による FLGGetReg の呼び出しのサンプル

146ページの図78 は、MYIMAGE オブジェクト・タイプの登録に関する特性と値の情報が入っている出力構造を示しています。

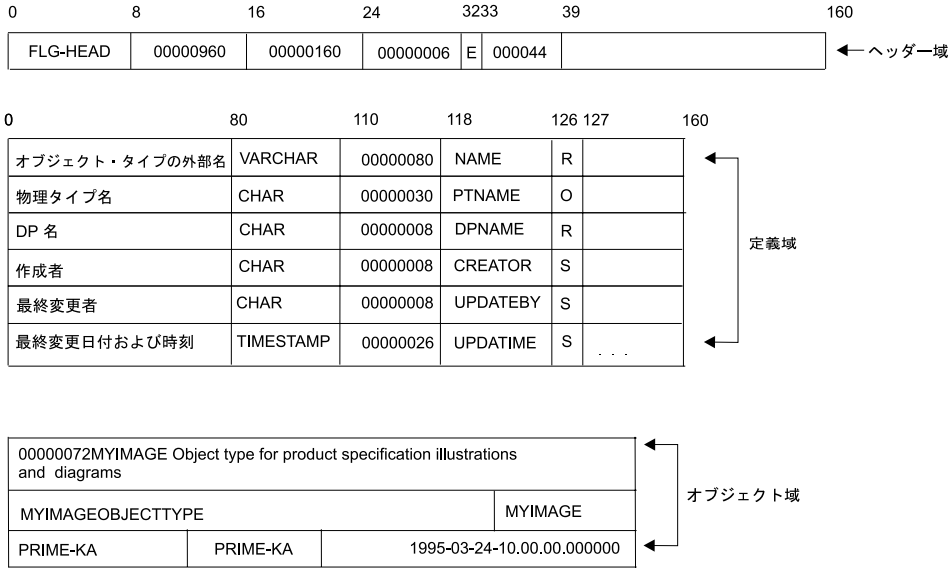


図 78. FLGGetReg のための出力構造のサンプル

この例で、ヘッダー・レコードの 33 桁目から 38 桁目までには、登録情報の検索対象となったオブジェクト・タイプのオブジェクト・タイプ ID (000044) が入っています。この ID は、pszObjTypeID パラメーターで入力として指定したオブジェクト・タイプ ID と一致しています。

FLGGetType

あるオブジェクト・タイプのすべての特性の定義を検索するために使用します。

許可

管理者またはユーザー

構文

```
APIRET  APIENTRY    FLGGetType( PSZ              pszObjTypeID,
                                PFLGHEADERAREA * ppObjTypeStruct,
                                  PFLGEXTCODE  pExtCode );
```

パラメーター

pszObjTypeID (PSZ) – 入力

オブジェクト・タイプが登録されたときに戻された 6 文字のシステム生成された固有の識別子 (オブジェクト・タイプ ID) を指します。

FLGConvertID または FLGListObjTypes API 呼び出しを使って、このオブジェクト・タイプ ID を検索することもできます。

ppObjTypeStruct (PFLGHEADERAREA) – 出力

出力構造を指すポインターのアドレスを指します。 FLGGetType が失敗した場合には、このポインターは NULL にセットされます。

pExtCode (PFLGEXTCODE) – 出力

理由コードに関連する拡張コードを指します。戻された理由コードに関連した意味のある拡張コードがあるかどうかについては、285ページの『付録 D. 情報カタログ・マネージャーの理由コード』を参照してください。

理由コード (APIRET)

この API 呼び出しの実行結果を表します。

戻された理由コードの説明については、285ページの『付録D. 情報カタログ・マネージャーの理由コード』を参照してください。

出力構造

FLGGetType は、148ページの図79 に示すように、要求されたオブジェクト・タイプの特性仕様を含む出力構造を作ります。

出力構造の定義域には、オブジェクト・タイプが作成されたときに指定された順に、要求されたオブジェクト・タイプの特性が入ります。

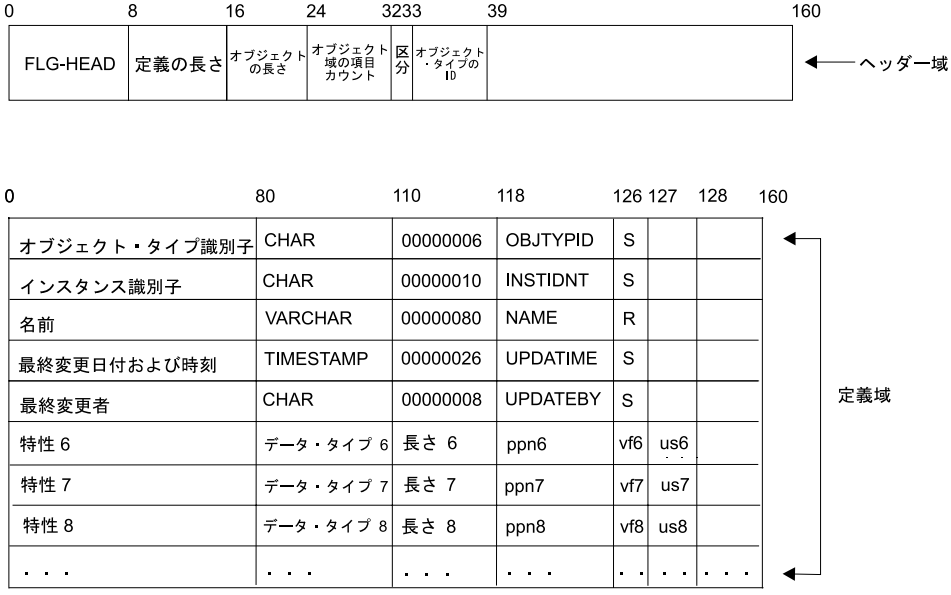


図 79. FLGGetType の出力構造

バイト・オフセットの意味については、35ページの『第4章 情報カタログ・マネージャー入力および出力構造』を参照してください。

使用方法

前提条件

pszObjTypeID パラメーターの値は、情報カタログに登録された既存のオブジェクト・タイプ ID を参照するものでなければなりません。

出力構造に割り振られたメモリーの解放

FLGGetType が出力構造にあるデータを戻した場合、出力構造に戻されたデータを保管してから、FLGFreeMem を呼び出さなくてはなりません (137ページの『FLGFreeMem』参照)。メモリーを解放するのに、たとえば C 言語指示のような他の方式は使用しないでください。

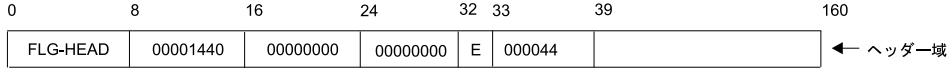
例

図80 は、FLGGetType API 呼び出しを出すために必要な C 言語コードを示しています。このサンプル・コードは、MYIMAGE オブジェクト・タイプの特性に関する情報を情報カタログから検索します。

```
APIRET         rc;                    // Declare reason code
UCHAR          pszObjTypeID[FLG_OBJTYPID_LEN + 1]; // Set to ID of MYIMAGE (000044)
PFLGHEADERAREA * ppObjTypeStruct;     // Pointer to the output structure
FLGEXTCODE     ExtCode=0;                  // Declare extended code
  .
  .  /* retrieving a user-defined object type - MYIMAGE  */
strcpy (pszObjTypeID,"000044");
  .
rc = FLGGetType (pszObjTypeID,
                 ppObjTypeStruct,
                    &ExtCode);             // Pass pointer to extended code
```

図 80. C 言語による FLGGetType の呼び出しのサンプル

図81 は、MYIMAGE オブジェクト・タイプの特性情報が入っている出力構造を示しています。



0	80	110	118	126	127	128	160
オブジェクト・タイプ識別子	CHAR	00000006	OBJTYPID		S		
インスタンス識別子	CHAR	00000010	INSTIDNT		S		
名前	VARCHAR	00000080	NAME		R	1	
最終変更日付および時刻	TIMESTAMP	00000026	UPDATIME		S		
最終変更者	CHAR	00000008	UPDATEBY		S		
イメージ・カラー	CHAR	00000010	COLOR		R		
イメージ・サイズ	CHAR	00000010	SIZE		R		
説明	LONG VARCHAR	00032700	DESC		R		
密度	CHAR	00000010	DENS		O		

定義域

図 81. FLGGetType のための出力構造のサンプル

FLGImport

タグ言語形式のフラット・ファイルからメタデータを情報カタログ・マネージャーにインポートするために使用します。

許可

管理者

構文

```
APIRET  APIENTRY   FLGImport( PSZ              pszTagFileID,
                                  PSZ              pszLogFileID,
                              PSZ              pszIcoPath,
                              FLGRESTARTOPTION RestartOpt,
                                  PFLGEXTCODE  pExtCode );
```

パラメーター

pszTagFileID (PSZ) − 入力

タグ言語ファイルを識別します。このパラメーターは必須パラメーターです。

このパラメーターにはドライブ、ディレクトリー・パス、およびファイル名が含まれており、FAT または HPFS ファイルとして有効な値でなければなりません。 このドライブは、取り外し可能ドライブであっても差し支えありません。 (ドライブとディレクトリーを除く) ファイル名とエクステンションは、 240 文字を超えてはなりません。ファイル名だけを入力すると、情報カタログ・マネージャー側は、タグ言語ファイルを DGWPATH 環境変数で示されるドライブおよびパスに存在するものとみなします。

pszTagFileID によって識別されるファイルには、インポートされる情報カタログ・マネージャー・オブジェクトと関連メタデータが入っています。

pszLogFileID (PSZ) − 入力

ログ・ファイルのロケーションと名前を指定します。このパラメーターは必須パラメーターです。

このパラメーターにはドライブ、ディレクトリー・パス、およびファイル名が含まれており、FAT または HPFS ファイルとして有効な値でなければなりません。 このドライブは、取り外し可能ドライブであってはなりません。ファイル名だけを指定すると、情報カタログ・マネージャー側は、ログ・ファイルを DGWPATH 環境変数で示されるドライブおよびパスに配置します。

このパラメーターで指定されたログ・ファイルが存在していない場合には、新しいファイルが作成されます。このパラメーターで指定されたログ・ファイルがすでに存在している場合には、情報カタログ・マネージャーはそれに対して行われます。

pszLogFileID によって識別されたファイルには、ログ情報、および FLGImport API 呼び出しの処理中に検出された警告とエラーが入ります。

pszIcoPath (PSZ) – 入力

OS/2 および Windows のアイコン・ファイルのロケーションを指定します。このパラメーターにはドライブおよびディレクトリーが含まれており、取り外し可能ドライブの FAT または HPFS ファイルとして有効な値でなければなりません。アイコン・パスの最大長は 246 文字です。

このパラメーターは任意選択です。このパラメーターを指定しないと、オブジェクト・タイプに関連するアイコンをインポートする指示がタグ言語ファイルに含まれていても、アイコン・ファイルはインポートされません。

このパラメーターを指定すると、インポート機能がこのパスを調べて、pszTagFileID によって識別されたタグ言語ファイルで参照されているアイコン・ファイルを探します。ただし、アイコンをあるオブジェクト・タイプに関連付けるようにタグ言語ファイルで指定されている場合、そのアイコンがアイコン・パスにないと、警告がログ・ファイルに記録されます。

RestartOpt (FLGRESTARTOPTION) – 入力

情報カタログ・マネージャーが入力タグ言語ファイルを先頭から処理するのか、チェックポイントから処理するのかを指定します。有効な値は以下のとおりです。

B 先頭

以前に同じ名前のタグ言語ファイルが指定され、実行時エラーのためにその一部だけしか処理されなかったことがあっても、タグ言語ファイルは先頭から処理されます。

C チェックポイント

同じタグ言語ファイルが一部だけ処理されています。システムは、そのファイルの実行を再開する箇所を示すチェックポイント・ラベル情報を保管しています。この場合、タグ言語ファイルは保管されたチェックポイント・ラベルを探索し、一致するラベルが見つかると、そこからインポートを再開します。一致するラベルが見つからない場合には、FLGImport API 呼び出しは失敗します。

それまでタグ言語ファイルが処理されたことがない場合に RestartOpt の値として C が指定されると、情報カタログ・マネージャーはタグ言語ファイルの先頭から処理を行います。

pExtCode (PFLGEXTCODE) – 出力

理由コードに関連する拡張コードを指します。戻された理由コードに関連した意味のある拡張コードがあるかどうかについては、285ページの『付録 D. 情報カタログ・マネージャーの理由コード』を参照してください。

理由コード (APIRET)

この API 呼び出しの実行結果を表します。

戻された理由コードの説明については、285ページの『付録D. 情報カタログ・マネージャーの理由コード』を参照してください。

使用方法

インポート・エラーのデバッグ

情報カタログ・マネージャーは、タグ言語ファイルをインポートする際に*ログ・ファイル* とエコー・*ファイル* を作成します。

ログ・ファイルには、インポート処理で発生した事象が記録されます。 これには、インポート処理が開始および終了した日付と時刻が含まれます。また、処理中に発生した問題に関する警告やエラー・メッセージも含まれます。 pszLogFileID パラメーターによりログ・ファイルが識別されます。

エコー・ファイルには、情報カタログ・マネージャーによって処理されたタグがリストされます。エコー・ファイルの名前はインポート・タグ言語ファイルと同じになり、ログ・ファイルと同じディレクトリーおよびパスに保管されますが、ファイル・エクステンションは `ech` になります。

エコー・ファイルとログ・ファイルを使用して、インポート・エラーの原因となっているタグを調べることができます。エコー・ファイルの最後の 1 行または 2 行には、タグ言語ファイルのうちのどのタグが原因でインポート処理が停止したのかが示されています。

削除履歴タグ・ファイルのインポート

特に Grouping オブジェクト・インスタンスやオブジェクト・タイプを削除している場合は、他の情報カタログで、誤って削除をしないために、削除履歴タグ・ファイルを、他の情報カタログにエクスポートするまえに内容を点検してください。

ディスケットからインポートする際の VisualAge® C++ プログラムの連係

C 言語プログラムによって情報カタログ・マネージャー情報をディスケットからインポートする FLGImport 呼び出しが出される場合には、情報カタ

ログ・マネージャーが PM インターフェースを使用して、必要なときにディスケットを挿入するようにユーザーに指示するメッセージを表示できるようにするために、そのプログラムを WINDOWAPI タイプのアプリケーションと連係させてください。

この連係は、次のいずれかの方法を使用して行うことができます。

– 次の連係ステートメント

  ilink /NOFREE /PMTYPE:vio /NOI *filename***.obj,,,dgwapi.lib,,**

– モジュール定義ファイル。 NAME ステートメントで *apptype* として **WINDOWAPI** を指定してください。

FLGImport を使用する前の変更のコミット

情報カタログ・マネージャーは、FLGImport でエラーが発生するとデータベースをロールバックします。 FLGImport 呼び出しの前に行われたコミットされていない変更内容が情報カタログ・マネージャーによってロールバックされないようにするために、プログラムの中で、FLGImport を出す前に FLGCommit を出すようにしてください。

例

図82 に示すサンプル・コードは、TAGFILE1.TAG というタグ言語ファイルをインポートします。情報カタログ・マネージャーは処理情報を TAGFILE1.LOG にログします。

```
APIRET         rc;                    // Declare reason code
UCHAR            pszTagFileID[FLG_TAG_FILE_ID_MAXLEN+1];  // ID for Tag Language file
UCHAR            pszLogFileID[FLG_LOG_FILE_ID_MAXLEN+1];  // ID for Log file
UCHAR            pszIconPath[FLG_ICON_PATH_MAXLEN+1];  // Path for Icon files
FLGRESTARTOPTION RestartOpt;                            // Restart option
FLGEXTCODE       ExtCode=0;            // Returned extended code
  .
  .  /* Importing the Tag Language file TAGFILE1.TAG              */
  .
strcpy (pszTagFileID,"c:¥¥DGdata¥¥TAGFILE1.TAG");
strcpy (pszLogFileID,"c:¥¥DGdata¥¥TAGFILE1.LOG");
strcpy (pszIconPath,"c:¥¥DGdata");
RestartOpt = FLG_RESTART_BEGIN;
rc = FLGImport (pszTagFileID,
                pszLogFileID,
                pszIconPath,
                RestartOpt,
                &ExtCode);        // Pass extended code by reference
```

図 82. C 言語による FLGImport の呼び出しのサンプル

FLGInit

情報カタログ・マネージャー API DLL を使用するために初期設定し、アプリケーションをデータベースに接続し、他の API 呼び出しで使用できる環境情報を検索するために使用します。

許可

管理者またはユーザー

構文

```
APIRET  APIENTRY   FLGInit( PSZ              pszUserID,
                            PSZ              pszPassword,
                            PSZ              pszDatabaseName,
                            FLGADMIN         Admin,
                                 PFLGHEADERAREA * ppListStruct,
                                  PFLGEXTCODE  pExtCode );
```

パラメーター

pszUserID (PSZ) – 入力
情報カタログ・データベースにログオンするためのユーザー ID を含む、NULL 文字で終了するストリングを指します。

pszPassword (PSZ) – 入力
ユーザーのパスワードを含む、NULL 文字で終了するストリングを指します。

pszDatabaseName (PSZ) – 入力
情報カタログを表すデータベース別名を含む、NULL 文字で終了するストリングを指します。

admin (FLGADMIN) – 入力
希望するユーザー・オプションを指定します。

FLG_YES
管理者としてログオンします。

FLG_NO
デフォルトです。ユーザーとしてログオンします。

ppListStruct (PFLGHEADERAREA) – 出力
出力構造を指すポインターのアドレスを指します。出力構造の形式については、56ページの『情報カタログ・マネージャーの API 出力構造』を参照してください。

出力構造がない場合には出力構造を指すポインターは NULL にセットされ、情報カタログ・マネージャーは理由コードとともにエラー条件を戻します。

pExtCode (PFLGEXTCODE) – 出力

理由コードに関連する拡張コードを指します。戻された理由コードに関連した意味のある拡張コードがあるかどうかについては、285ページの『付録D. 情報カタログ・マネージャーの理由コード』を参照してください。

理由コード (APIRET)

この API 呼び出しの実行結果を表します。

戻された理由コードの説明については、285ページの『付録D. 情報カタログ・マネージャーの理由コード』を参照してください。

出力構造

FLGInit は、156ページの図83 に示すように、情報カタログ・マネージャー環境に関する情報が入っている出力構造を作成します。

出力構造のオブジェクト域には、必要な登録およびオブジェクトの特性がユーザーの使用する国語で示されます。オブジェクト域には、ユーザーの情報カタログ・マネージャー環境に関する情報を示す値も含まれています。

図 83. FLGInit 出力構造

使用方法

出力構造は、すべてのオブジェクト・タイプ登録、オブジェクト・タイプ、およびオブジェクトに関して必要な 80 バイトの特性名を戻します。

英語版以外の情報カタログ・マネージャーでは、これらの各特性のオブジェクト域の値は翻訳されています。 FLGCreateReg および FLGCreateType で使用できるように、これらの翻訳された値を保管しておく必要があります。

出力構造に割り振られたメモリーの解放

FLGInit が出力構造にあるデータを戻した場合、出力構造に戻されたデータを保管してから、FLGFreeMem を呼び出さなくてはなりません (137ページの『FLGFreeMem』参照)。メモリーを解放するのに、たとえば C 言語指示のような他の方式は使用しないでください。

表17 は、FLGInit によって戻される必須特性を示しています。

表 17. FLGInit によって戻される必須特性名

特性名	説明
オブジェクト・タイプの外部名	オブジェクト・タイプ登録における最初の登録特性
物理タイプ名	オブジェクト・タイプ登録における 2 番目の登録特性
DP 名	オブジェクト・タイプ登録における 3 番目の登録特性
作成者	オブジェクト・タイプ登録における 4 番目の登録特性
最終変更者	オブジェクト・タイプ登録における 5 番目の登録特性
最終変更日付および時刻	オブジェクト・タイプ登録における 6 番目の登録特性
オブジェクト・タイプ識別子	オブジェクト・タイプにおける最初の必須特性
インスタンス識別子	オブジェクト・タイプにおける 2 番目の必須特性
名前	オブジェクト・タイプにおける 3 番目の必須特性
最終変更日付および時刻	オブジェクト・タイプにおける 4 番目の必須特性
最終変更者	オブジェクト・タイプにおける 5 番目の必須特性

この出力構造は、環境値も戻します。他の API 呼び出しで使用できるように、これらの値を保管してください。

表 18. FLGInit によって戻される環境値

特性名	説明
記憶環境	V*x*R*x*M*x* 形式のリリース番号によるデータベース製品名称。たとえば、次のものがあります。 **DB2/NT V07R01M0** DB2 UDB (Windows NT 版) プロダクト
記憶環境サイズ	この環境における情報カタログの PTNAME の最大長を示す値。
非適用記号	未指定のデータ・フィールドを表す 1 文字の情報カタログ・マネージャー環境のデフォルト・トークン。この値は、導入中に設定されたものです。
バージョン	情報カタログ・マネージャーのバージョンを表す 20 文字のインディケーター。
言語	3 文字の各国語コード。たとえば、ENU は英語を表します。有効な値は以下のとおりです。 **CHS** 中国語 (簡体字) **CHT** 中国語 (繁体字) **DAN** デンマーク語 **DEU** ドイツ語 **ENU** 米国英語 **ESP** スペイン語 **FIN** フィンランド語 **FRA** フランス語 **ITA** イタリア語 **JPN** 日本語 **KOR** 韓国語 **NLB** ベルギー系フランス語 **NOR** ノルウェー語 **PTB** ブラジル・ポルトガル語 **SVE** スウェーデン語
管理者のユーザー ID	現在ログオンしている管理者を表す 8 文字のユーザー ID。
プロダクト・パス	260 文字による情報カタログ・マネージャーの完全な作業パス。
システム・パス長	システムの最大パス長を表す 8 文字の値。
コード・ページ	4 文字のコード・ページ識別子

表 18. FLGInit によって戻される環境値 (続き)

特性名	説明
ユーザー・タイプ	1 文字の識別子。以下のとおりに設定。 **A** ログオン・ユーザー ID が 1 次的管理担当者。 **B** ログオン・ユーザー ID がバックアップ管理担当者。 **D** ログオン・ユーザー ID がオブジェクト管理タスクの実行権限を持つユーザー。 **W** ログオン・ユーザー ID がユーザー。

例

図84 は、FLGInit API 呼び出しを出すために必要な C 言語コードを示しています。このサンプル・コードは情報カタログ・マネージャー API DLL を初期設定して、情報アプリケーションが情報カタログ・マネージャー API 呼び出しを出すことができるようにします。

```
UCHAR          pszUserID[FLG_USERID_LEN + 1];
UCHAR          pszPassword[FLG_PASSWORD_LEN + 1];
UCHAR          pszDatabaseName[FLG_DATABASENAME_LEN + 1];
FLGADMIN       admin = FLG_YES;
APIRET         rc;                          // reason code
PFLGHEADERAREA * ppListStruct;   // pointer to output structure pointer
FLGEXTCODE     ExtCode = 0;                    // Extended code
.
.  // IA specific code
.
strcpy( pszUserID, "LAUTZ" );
strcpy( pszPassword, "MYPASSWD");
strcpy( pszDatabaseName, "CATALOG");
rc = FLGInit (pszUserName,
              pszPassword,
              pszDatabaseName,
              admin,
                      ppListStruct,
              &ExtCode );
.  // Issue FLGFreeMem to release the output structure created by FLGInit
.  // Calls to the FLG API
.  // When complete, call
.  // FLGTerm()
```

図 84. C 言語による FLGInit の呼び出しのサンプル

160ページの図85 にこの出力構造を示します。

図 85. FLGInit のための出力構造のサンプル

FLGListAnchors

Grouping 区分のすべてのアンカー・インスタンスのリストを検索します。アンカー とは、他のオブジェクトを含むが、他のオブジェクトには含まれない Grouping 区分のオブジェクトのことを言います。

許可

管理者またはユーザー

構文

```
APIRET  APIENTRY   FLGListAnchors( PFLGHEADERAREA * ppListStruct,
                                   PFLGEXTCODE  pExtCode );
```

パラメーター

ppListStruct (PFLGHEADERAREA) – 出力

アンカーをリストする出力構造を指すポインターのアドレスを指します。出力構造がない場合には、構造を指すポインターは NULL にセットされます。

この出力構造には、各アンカー・オブジェクト・インスタンスに関する以下の情報が入ります。

- FLGID (16 文字)
- 名前 (80 文字)

すべてのインスタンスは、情報カタログに使用されたデータベースの照合順序に従って、最初にオブジェクト・タイプ名、次に名前によって分類されます。

FLGListAnchors によって戻すことのできるオブジェクト・インスタンスの最大数は 1600 です。

pExtCode (PFLGEXTCODE) – 出力

理由コードに関連する拡張コードを指します。戻された理由コードに関連した意味のある拡張コードがあるかどうかについては、285ページの『付録 D. 情報カタログ・マネージャーの理由コード』を参照してください。

理由コード (APIRET)

この API 呼び出しの実行結果を表します。

戻された理由コードの説明については、285ページの『付録D. 情報カタログ・マネージャーの理由コード』を参照してください。

出力構造

FLGListAnchors は、図86 に示すように、アンカーのリストが入っている出力構造を作成します。

この出力構造のオブジェクト域にはアンカー・オブジェクト・インスタンスのリストが入ります。各オブジェクト・インスタンスは、FLGID の値と外部名で識別されます。

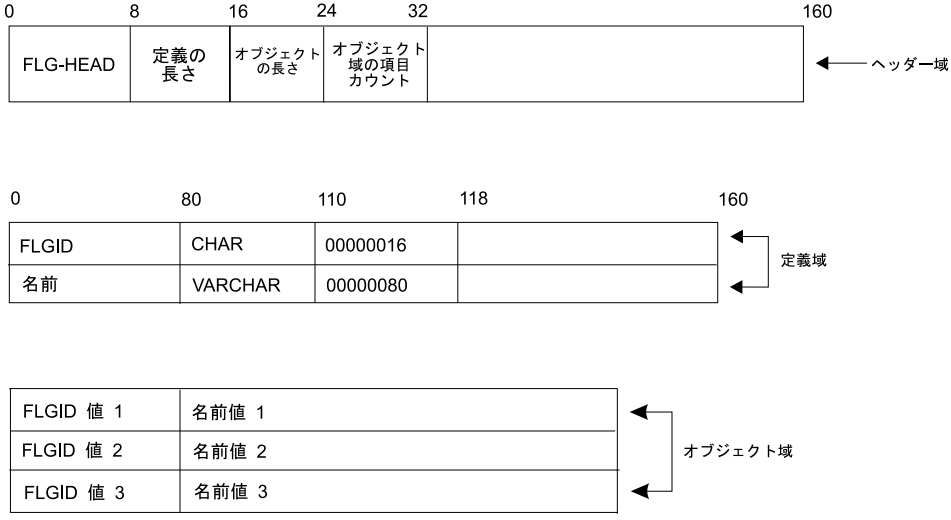


図 86. FLGListAnchors 出力構造

使用方法

出力構造に割り振られたメモリーの解放

FLGListAnchors が出力構造にあるデータを戻した場合、出力構造に戻されたデータを保管してから、FLGFreeMem を呼び出さなくてはなりません (137ページの『FLGFreeMem』参照)。メモリーを解放するのに、たとえば C 言語指示のような他の方式は使用しないでください。

例

163ページの図87 は、FLGListAnchors API 呼び出しを出すために必要な C 言語コードを示しています。このサンプル・コードは、情報カタログからアンカーのリストを検索します。

```
APIRET          rc;              // reason code from FLGListAnchors
PFLGHEADERAREA * ppListStruct;   // pointer to output structure pointer
FLGEXTCODE      ExtCode=0;            // Extended code
  .
  .
rc = FLGListAnchors (ppListStruct,   // address of output structure pointer
                     &ExtCode);
```

図 87. C 言語による FLGListAnchors の呼び出しのサンプル

図88 にこの出力構造を示します。

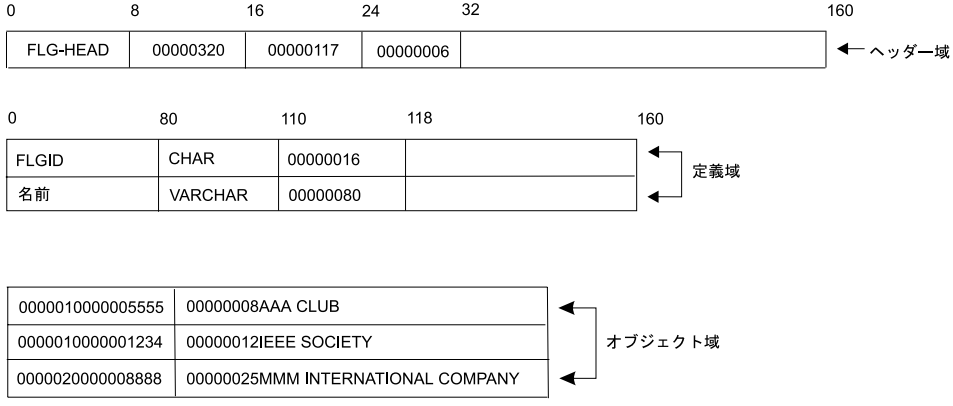


図 88. FLGListAnchors のための出力構造のサンプル

FLGListAssociates

指定されたオブジェクト・インスタンスまたはオブジェクト・タイプの関連するインスタンスのリストを検索します。関連とは、以下のことを指します。

- Grouping オブジェクト・インスタンスに含まれるインスタンス
- オブジェクト・インスタンスの Contact インスタンス
- オブジェクト・インスタンスの Attachment インスタンス
- オブジェクト・インスタンスにリンクしたインスタンス
- オブジェクト・タイプに関連した Program インスタンス

許可

管理者またはユーザー

構文

```
APIRET  APIENTRY   FLGListAssociates( PSZ              pszInBuffer,
                                      FLGOPTIONS   Options,
                                     PFLGHEADERAREA * ppListStruct,
                                      PFLGEXTCODE  pExtCode );
```

パラメーター

pszInBuffer (PSZ) − 入力

オブジェクト・インスタンスの 16 文字のシステム生成された固有の識別子、またはオブジェクト・タイプの 6 文字のシステム生成された固有の識別子のいずれかを含む入力バッファーを指します。これはリストするものによって異なります。

Attachment

非 Attachment 区分のオブジェクト・インスタンスのオブジェクト・インスタンス ID (FLGID)。

Comments

非 Comments タイプのオブジェクト・インスタンスの FLGID

Contact

Elemental または Grouping 区分のオブジェクト・インスタンスの FLGID

Containees

Grouping 区分のオブジェクト・インスタンスの FLGID

Links Elemental または Grouping 区分のオブジェクト・インスタンスの FLGID

Program
非 Program 区分オブジェクト・タイプのオブジェクト・タイプ ID

Options (FLGOPTIONS) − 入力
以下のいずれかのオプションを選択します。

FLG_LIST_ATTACHMENT
指定されたインスタンスと Attachment 関係にあるオブジェクト・インスタンスを検索します。

FLG_LIST_COMMENTS
指定されたインスタンスに付加された Comments オブジェクト・インスタンスを検索します。 FLG_LIST_COMMENTS は、FLG_LIST_ATTACHMENT と同じオブジェクト・インスタンスを検索しますが、各インスタンスについて、より多くの情報 (最終変更日付および時刻、作成者) を戻します。

FLG_LIST_CONTACT
指定されたインスタンスに付加された Contact オブジェクト・インスタンスを検索します。

FLG_LIST_CONTAIN
指定されたインスタンスに含まれるオブジェクト・インスタンスを検索します。

FLG_LIST_LINK
指定されたインスタンスにリンクしたオブジェクト・インスタンスを検索します。

FLG_LIST_PROGRAM
指定されたオブジェクト・タイプに関連した Programs オブジェクト・インスタンスを検索します。

ppListStruct (PFLGHEADERAREA) − 出力
関連をリストする出力構造を指すポインターのアドレスを指します。出力構造がない場合には、構造を指すポインターは NULL にセットされます。

各インスタンスの出力構造には、以下の情報が示されます。

FLGID (16 文字)

名前 (80 文字)

さらに、FLG_LIST_CONTAIN では、各インスタンスの出力構造が、それ自体が包含側であるかどうかを示すフラグ (CHILDIND) も示します。FLG_LIST_COMMENTS では、各インスタンスの出力構造は、以下の事項も含みます。

最終変更日付および時刻

作成者

すべてのインスタンスは、使用するデータベース管理システムの照合順序に従って、オブジェクト・タイプ名、オブジェクト・インスタンス名の昇順で分類されます。

FLGListAssociates によって戻すことのできるオブジェクト・インスタンスの最大数は、5000 です。

pExtCode (PFLGEXTCODE) − 出力

理由コードに関連する拡張コードを指します。戻された理由コードに関連した意味のある拡張コードがあるかどうかについては、285ページの『付録 D. 情報カタログ・マネージャーの理由コード』を参照してください。

理由コード (APIRET)

この API 呼び出しの実行結果を表します。

戻された理由コードの説明については、285ページの『付録D. 情報カタログ・マネージャーの理由コード』を参照してください。

使用方法

出力構造に割り振られたメモリーの解放

FLGListAssociates が出力構造にあるデータを戻した場合、出力構造に戻されたデータを保管してから、FLGFreeMem を呼び出さなくてはなりません(137ページの『FLGFreeMem』参照)。メモリーを解放するのに、たとえば C 言語指示のような他の方式は使用しないでください。

例

このサンプル・コードは、Grouping オブジェクト・タイプ、MYREGION に関連した Programs オブジェクト・インスタンスのリストを検索します。 167ページの図89 は、FLGListAssociates 呼び出しを出すために必要な C 言語コードを示しています。

```
APIRET         rc;                        // reason code
UCHAR           pszObjTypeID[FLG_OBJTYPID_LEN + 1];
PFLGHEADERAREA * ppReturnObjList;   // ptr to output structure ptr
FLGOPTIONS       Option=0;
FLGEXTCODE       xc=0;             // extended code
  .
  .
  .
Option=Option | FLG_LIST_PROGRAM;
rc = FLGListAssociates (pszObjTypeID,
                        Option,
                        ppReturnObjList,
                        &xc);
```

図 89. C 言語による FLGListAssociates の呼び出しのサンプル

図90 は、図89 にある FLGListAssociates 呼び出しのための出力構造を示しています。

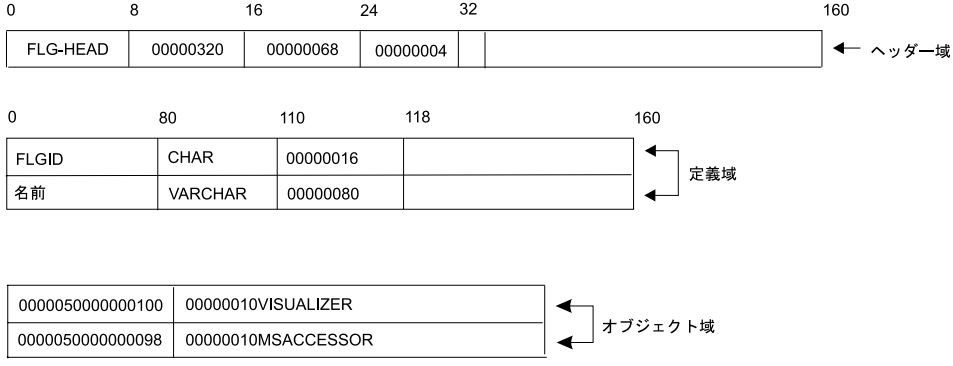


図 90. FLGListAssociates のための出力構造のサンプル

このサンプル・コードは、Grouping オブジェクト、MYBGROUP に含まれるオブジェクト・インスタンスを検索します。 168ページの図91 は、FLGListAssociates 呼び出しを出すために必要な C 言語コードを示しています。

```
APIRET         rc;                        // reason code
UCHAR            objid[FLG_ID_LEN + 1];
PFLGHEADERAREA * ppReturnObjList;   // ptr to output structure ptr
FLGOPTIONS       Option=0;
FLGEXTCODE       xc=0;             // extended code
  .
  .
  .
Option=Option | FLG_LIST_CONTAIN;
rc = FLGListAssociates (objid,
                        Option,
                        ppReturnObjList,
                        &xc);
```

図 91. C 言語による FLGListAssociates の呼び出しのサンプル

図92 は、図91 にある FLGListAssociates 呼び出しのための出力構造を示しています。

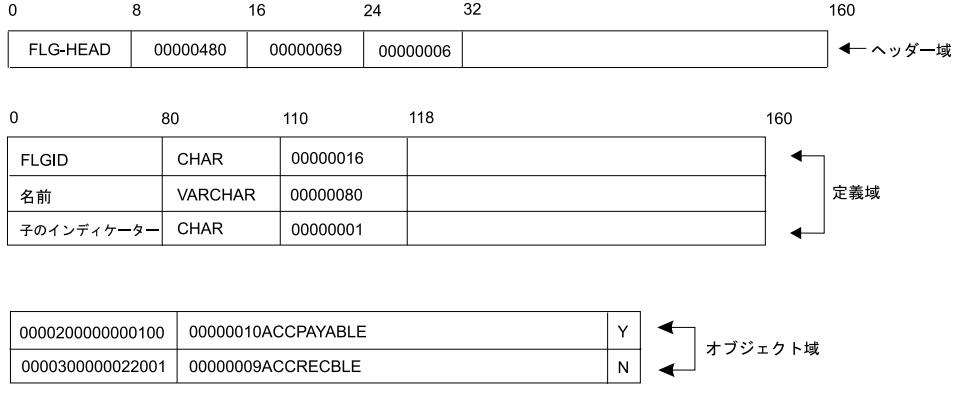


図 92. FLGListAssociates のための出力構造のサンプル

このサンプル・コードは、Grouping オブジェクト、 MYBGROUP の Contact オブジェクト・インスタンスを検索します。 169ページの図93 は、FLGListAssociates 呼び出しを出すために必要な C 言語コードを示しています。

```
APIRET         rc;                        // reason code
UCHAR            objid[FLG_ID_LEN + 1];
PFLGHEADERAREA * ppReturnObjList;   // ptr to output structure ptr
FLGOPTIONS       Option=0;
FLGEXTCODE       xc=0;             // extended code
  .
  .
  .
Option=Option | FLG_LIST_CONTACT;
rc = FLGListAssociates (objid,
                        Option,
                        ppReturnObjList,
                        &xc);
```

図 93. C 言語による FLGListAssociates の呼び出しのサンプル

図94 は、図93 にある FLGListAssociates 呼び出しのための出力構造を示しています。

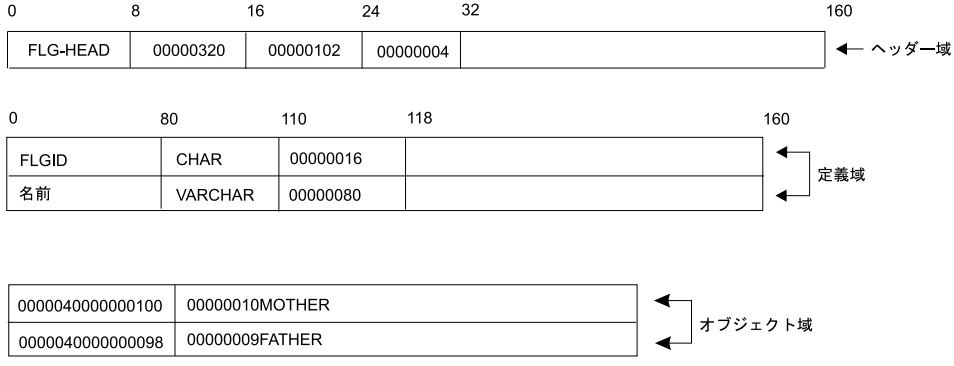


図 94. FLGListAssociates のための出力構造のサンプル

このサンプル・コードは、Grouping オブジェクト、 MYBGROUP の Attachment 区分のオブジェクト・インスタンスを検索します。 170ページの図 95 は、FLGListAssociates 呼び出しを出すために必要な C 言語コードを示しています。

```
APIRET         rc;                        // reason code
UCHAR            objid[FLG_ID_LEN + 1];
PFLGHEADERAREA * ppReturnObjList;   // ptr to output structure ptr
FLGOPTIONS       Option=0;
FLGEXTCODE       xc=0;             // extended code
  .
  .
  .
Option=Option | FLG_LIST_ATTACHMENT;
rc = FLGListAssociates (objid,
                        Option,
                        ppReturnObjList,
                        &xc);
```

図 95. C 言語による FLGListAssociates の呼び出しのサンプル

図96 は、図95 にある FLGListAssociates 呼び出しのための出力構造を示しています。

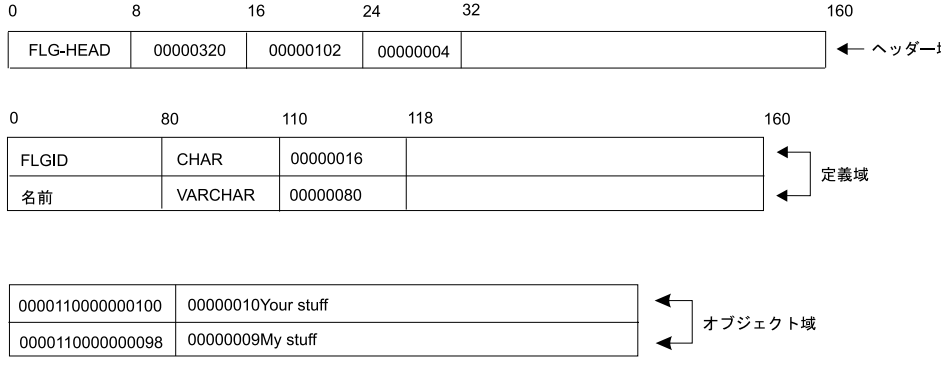


図 96. FLGListAssociates のための出力構造のサンプル

このサンプル・コードは、Elemental オブジェクト、MYREPORT に付加された Comments オブジェクト・インスタンスを検索します。 171ページの図97 は、FLGListAssociates 呼び出しを出すために必要な C 言語コードを示しています。

```
APIRET         rc;                        // reason code
UCHAR            objid[FLG_ID_LEN + 1];
PFLGHEADERAREA * ppReturnObjList;   // ptr to output structure ptr
FLGOPTIONS       Option=0;
FLGEXTCODE       xc=0;             // extended code
  .
  .
  .
Option=Option | FLG_LIST_COMMENTS;
rc = FLGListAssociates (objid,
                        Option,
                        ppReturnObjList,
                        &xc);
```

図 97. C 言語による FLGListAssociates の呼び出しのサンプル

図98 は、図97 にある FLGListAssociates 呼び出しのための出力構造を示しています。

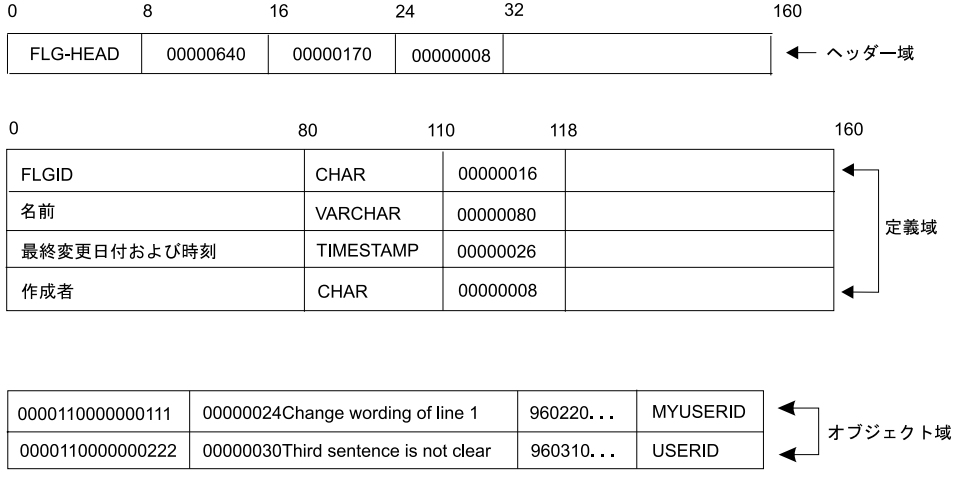


図 98. FLGListAssociates のための出力構造のサンプル

このサンプル・コードは、Grouping オブジェクト、MYBGROUP がリンクしているオブジェクト・インスタンスを検索します。 172ページの図99 は、FLGListAssociates 呼び出しを出すために必要な C 言語コードを示しています。

```
APIRET         rc;                          // reason code
UCHAR            objid[FLG_ID_LEN + 1];
PFLGHEADERAREA * ppReturnObjList;   // ptr to output structure ptr
FLGOPTIONS       Option=0;
FLGEXTCODE       xc=0;             // extended code
  .
  .
  .
Option=Option | FLG_LIST_LINK;
rc = FLGListAssociates (objid,
                        Option,
                        ppReturnObjList,
                        &xc);
```

図 99. C 言語による FLGListAssociates の呼び出しのサンプル

図100 は、図99 にある FLGListAssociates 呼び出しのための出力構造を示しています。

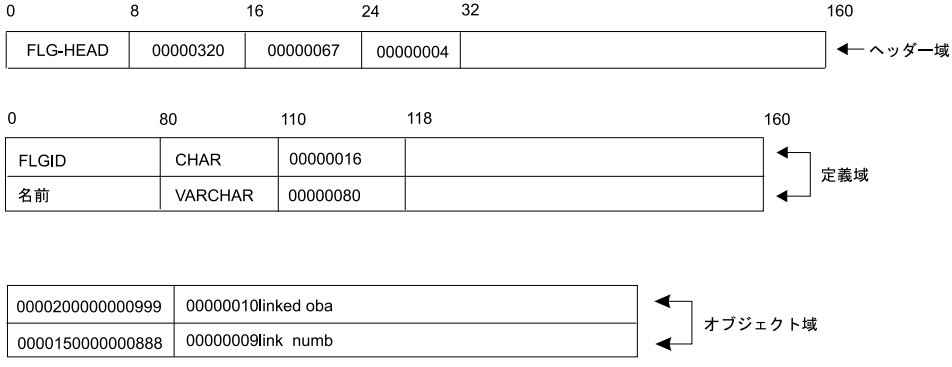


図 100. FLGListAssociates のための出力構造のサンプル

FLGListContacts

Elemental または Grouping オブジェクトに関する Contact オブジェクトのリストを検索するために使用します。

許可

管理者またはユーザー

構文

```
APIRET  APIENTRY  FLGListContacts( PSZ              pszFLGID,
                                   PFLGHEADERAREA * ppListStruct,
                                    PFLGEXTCODE  pExtCode );
```

パラメーター

pszFLGID (PSZ) − 入力

Contact が検索されるオブジェクト・インスタンスを表す 16 文字の FLGID を指します。

この ID の 1 文字目から 6 文字目までは、このインスタンスのオブジェクト・タイプを識別します。

この ID の 7 文字目から 16 文字目までは、システム生成された固有のインスタンス ID です。

ppListStruct (PFLGHEADERAREA) − 出力

Contact をリストする出力構造を指すポインターのアドレスを指します。出力構造がない場合には、構造を指すポインターは NULL にセットされます。

この出力構造には、各 Contact オブジェクトの 16 文字の FLGID と 80 文字の外部名が入ります。

リスト内の項目は、情報カタログによって使用されるデータベース管理システムで使用している照合順序に従って、最初にオブジェクト・タイプ名によって分類され、次に各インスタンスの名前特性の値によって分類されます。

FLGListContacts によって戻すことのできる Contact オブジェクト・インスタンスの最大数は計算機で使用可能な記憶域によって異なり、約 5000 です。

pExtCode (PFLGEXTCODE) − 出力

理由コードに関連する拡張コードを指します。戻された理由コードに関連した意味のある拡張コードがあるかどうかについては、285ページの『付録D. 情報カタログ・マネージャーの理由コード』を参照してください。

理由コード (APIRET)

この API 呼び出しの実行結果を表します。

戻された理由コードの説明については、285ページの『付録D. 情報カタログ・マネージャーの理由コード』を参照してください。

出力構造

FLGListContacts は、図101 に示すように、Contact のリストが入っている出力構造を作成します。

出力構造のオブジェクト域には、指定されたオブジェクト・インスタンスに関連する Contact オブジェクト・インスタンスのリストが入ります。これらの Contact オブジェクトは、各オブジェクト・インスタンスの FLGID の値と外部名によって識別されます。

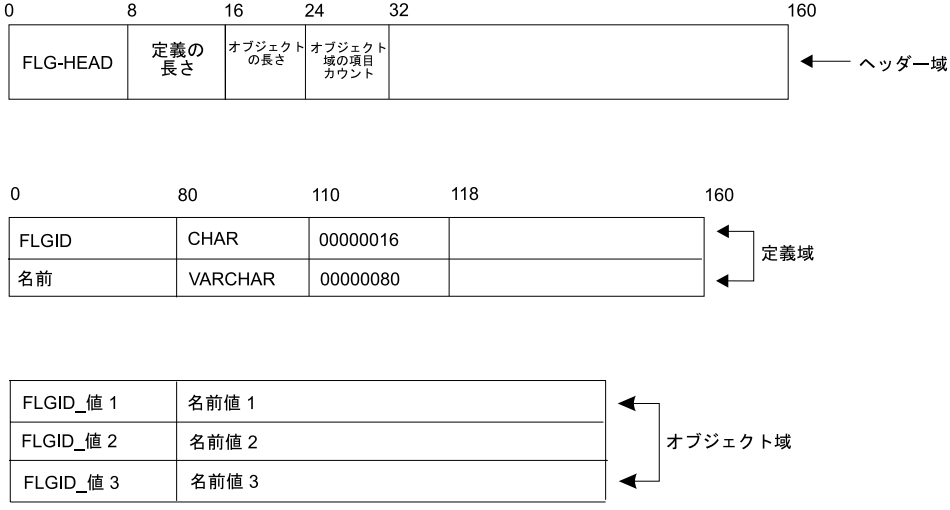


図 101. FLGListContacts 出力構造

使用方法

出力構造に割り振られたメモリーの解放

FLGListContacts が出力構造にあるデータを戻した場合、出力構造に戻されたデータを保管してから、FLGFreeMem を呼び出さなくてはなりません (137ページの『FLGFreeMem』参照)。メモリーを解放するのに、たとえば C 言語指示のような他の方式は使用しないでください。

例

図102 は、FLGListContacts API 呼び出しを出すために必要な C 言語コードを示しています。このサンプル・コードは、Elemental オブジェクト MYREPORT に関する Contact のリストを検索します。

```
APIRET           rc;               // reason code from FLGListContacts
UCHAR           pszFLGID[FLG_ID_LEN + 1];
PFLGHEADERarea * ppListStruct;   // pointer to output structure pointer
FLGEXTCODE       ExtCode=0;              // extended code
  .
  .  /* allocate storage for input parms                */
  .  /* set objid to FLGID of 'MYREPORT'               */
  .
rc = FLGListContacts (pszFLGID,
                   ppListStruct, // address of output structure pointer
                      &ExtCode);
```

図 102. C 言語による FLGListContacts の呼び出しのサンプル

176ページの図103 は、この API 呼び出しのための出力構造を示しています。

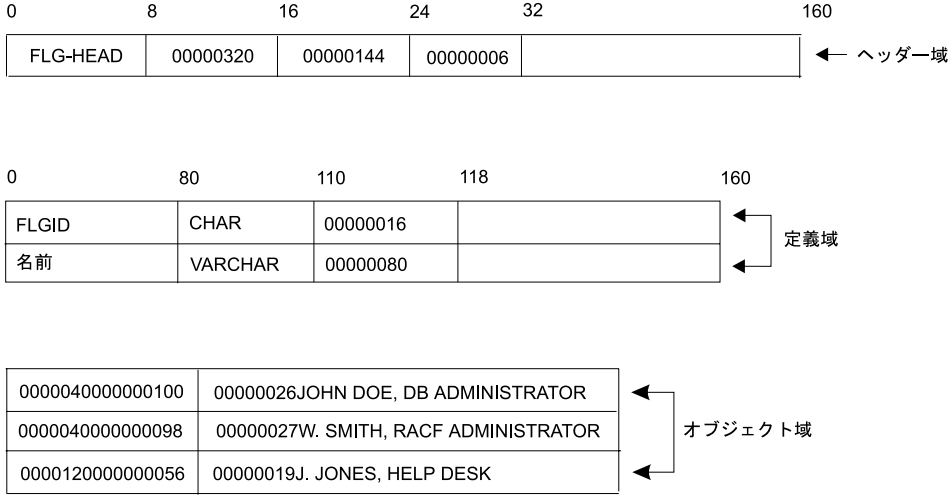


図 103. FLGListContacts のための出力構造のサンプル

FLGListObjTypes

情報カタログ・データベース内で現在登録され、作成されているすべてのオブジェクト・タイプを表示するために使用します。

許可

管理者またはユーザー

構文

```
APIRET  APIENTRY  FLGListObjTypes( PFLGHEADERAREA * ppListStruct,
                                   PFLGEXTCODE  pExtCode );
```

パラメーター

ppListStruct (PFLGHEADERAREA) − 出力

オブジェクト・タイプをリストする出力構造を指すポインターのアドレスを指します。出力構造がない場合には、構造を指すポインターは NULL にセットされます。

各項目には以下の情報が入ります。

- オブジェクト・タイプの ID
- オブジェクト・タイプの外部名 (80 バイト)
- オブジェクト・タイプの省略名 (8 バイトの DP 名)

各項目は 80 バイトのオブジェクト・タイプ外部名 (EXTERNAL NAME OF OBJ TYPE) によって分類されます。実際の順序は、情報カタログに使用されているデータベース管理システムで使用する照合順序に従います。

pExtCode (PFLGEXTCODE) − 出力

理由コードに関連する拡張コードを指します。戻された理由コードに関連した意味のある拡張コードがあるかどうかについては、285ページの『付録D. 情報カタログ・マネージャーの理由コード』を参照してください。

理由コード (APIRET)

この API 呼び出しの実行結果を表します。

戻された理由コードの説明については、285ページの『付録D. 情報カタログ・マネージャーの理由コード』を参照してください。

出力構造

FLGListObjTypes は、図104 に示すように、オブジェクト・タイプのリストが入っている出力構造を作成します。

出力構造のオブジェクト域には、情報カタログ内のすべてのオブジェクト・タイプのリストが入ります。これらのオブジェクト・タイプは、オブジェクト・タイプ ID の値、オブジェクト・タイプの外部名、およびオブジェクト・タイプの DP 名 (省略名) によって識別されます。

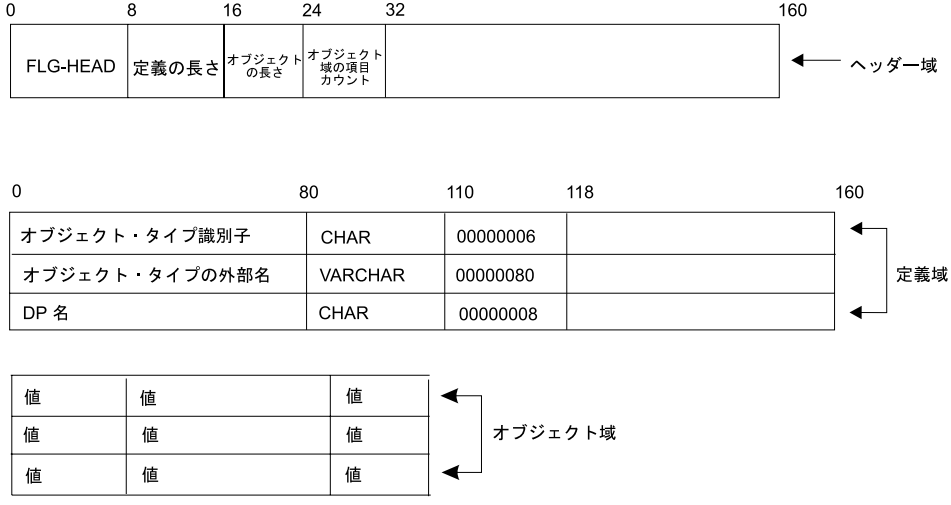


図 104. FLGListObjTypes 出力構造

バイト・オフセットの意味については、56ページの『情報カタログ・マネージャーの API 出力構造』を参照してください。

使用方法

出力構造に割り振られたメモリーの解放

FLGListObjTypes が出力構造にあるデータを戻した場合、出力構造に戻されたデータを保管してから、FLGFreeMem を呼び出さなくてはなりません (137ページの『FLGFreeMem』参照)。メモリーを解放するのに、たとえば C 言語指示のような他の方式は使用しないでください。

例

図105 は、FLGListObjTypes API 呼び出しを出すために必要な C 言語コードを示しています。このサンプル・コードは、情報カタログからすべてのオブジェクト・タイプのリストを検索します。

```
PFLGHEADERAREA * ppListStruct;   // pointer to output structure pointer
APIRET          rc;             // reason code from FLGListObjTypes
FLGEXTCODE       ExtCode=0;              // extended code
  .
  .
  .
rc = FLGListObjTypes (ppListStruct,  // address of output structure pointer
                      &ExtCode);
```

図 105. C 言語による FLGListObjTypes の呼び出しのサンプル

図106 は、この API 呼び出しのための出力構造を示しています。

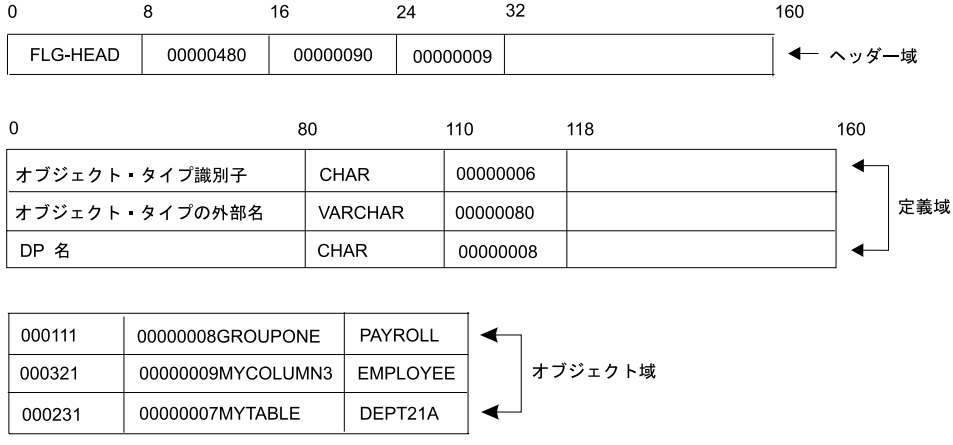


図 106. FLGListObjTypes のための出力構造のサンプル

FLGListOrphans

Attachment、Contact、または Program 区分のすべてのオーファン・インスタンスのリストを検索します。オーファン とは、ほかのオブジェクト・インスタンスに関連していない Attachment、または Contact オブジェクト、または、どのオブジェクト・タイプとも関連していない Program オブジェクトを指します。

このリストを使用して、オーファン・オブジェクト・インスタンスをほかのオブジェクトに関連付けたり、オーファン・インスタンスを削除したりして、情報カタログを整頓することができます。

許可

管理者またはユーザー

構文

```
APIRET  APIENTRY   FLGListOrphans( PSZ              pszObjTypeID,
                                    FLGOPTIONS   Options,
                                   PFLGHEADERAREA * ppListStruct,
                                    PFLGEXTCODE  pExtCode );
```

パラメーター

pszObjTypeID (PSZ) – 入力

存在しているが、現在はどのオブジェクト・インスタンスとも関連していないインスタンスのリストを検索するオブジェクト・タイプの 6 文字のシステム生成された固有の識別子 (オブジェクト・タイプ ID)。指定するオブジェクト・タイプ ID は、何をリストしたいかによります。

Attachment
Attachment 区分オブジェクト・タイプ ID

Comments
このパラメーターは無視されます。

Contact
Contact 区分オブジェクト・タイプ ID

Program
Program 区分オブジェクト・タイプ ID

pszObjTypeID が NULL のとき、情報カタログ・マネージャーは、Attachment 区分 (FLG_LIST_ATTACHMENT が指定されている場合)、また

は Contact 区分 (FLG_LIST_CONTACT が指定されている場合) にあるすべてのオブジェクト・タイプのオーファンを戻します。

Options (FLGOPTIONS) – 入力
以下のいずれかのオプションを選択します。

FLG_LIST_ATTACHMENT
現在、付加されていない Attachment 区分のオブジェクト・インスタンスを検索します。

FLG_LIST_COMMENTS
現在、付加されていない Comments オブジェクト・インスタンスを検索します。 FLG_LIST_COMMENTS は、FLG_LIST_ATTACHMENT と同じオブジェクト・インスタンスを検索しますが、各インスタンスについて、より多くの情報 (最終変更日付および時刻、作成者) を戻します。

FLG_LIST_CONTACT
現在、付加されていない Contact 区分のオブジェクト・インスタンスを検索します。

FLG_LIST_PROGRAM
現在、どのオブジェクト・タイプにも関連していない Programs オブジェクト・インスタンスを検索します。

ppListStruct (PFLGHEADERAREA) – 出力
オーファンをリストする出力構造を指すポインターのアドレスを指します。出力構造がない場合には、構造を指すポインターは NULL にセットされます。

各インスタンスの出力構造には、以下の情報が示されます。

FLGID (16 文字)

名前 (80 文字)

さらに、FLG_LIST_COMMENTS では、各インスタンスの出力構造は、以下の事項も含みます。

最終変更日付および時刻

作成者

すべてのインスタンスは、使用するデータベース管理システムの照合順序に従って、オブジェクト・タイプ名、オブジェクト・インスタンス名の昇順で分類されます。

FLGListOrphans によって戻すことのできるオブジェクト・インスタンスの最大数は、1600 です。

pExtCode (PFLGEXTCODE) − 出力

理由コードに関連する拡張コードを指します。戻された理由コードに関連した意味のある拡張コードがあるかどうかについては、285ページの『付録D. 情報カタログ・マネージャーの理由コード』を参照してください。

理由コード (APIRET)

この API 呼び出しの実行結果を表します。

戻された理由コードの説明については、285ページの『付録D. 情報カタログ・マネージャーの理由コード』を参照してください。

使用方法

制約事項

ユーザーが FLGListOrphans を使って、オーファンである Comments をリストすると、 FLGListOrphans は、ユーザーが作成者でもある Comments のみを戻します。

出力構造に割り振られたメモリーの解放

FLGListOrphans が出力構造にあるデータを戻した場合、出力構造に戻されたデータを保管してから、FLGFreeMem を呼び出さなくてはなりません (137ページの『FLGFreeMem』参照)。メモリーを解放するのに、たとえば C 言語指示のような他の方式は使用しないでください。

例

このサンプル・コードは、オーファンであるすべての Program 区分のオブジェクト・インスタンスを検索します。 図107 は、FLGListOrphans 呼び出しを出すために必要な C 言語コードを示しています。

```
APIRET         rc;                          // reason code
PFLGHEADERAREA * ppReturnObjList;   // ptr to output structure ptr
FLGOPTIONS       Option=0;
FLGEXTCODE       xc=0;              // extended code
  .
  .
  .
Option=Option | FLG_LIST_PROGRAM;
rc = FLGListOrphans (NULL,
                     Option,
                     ppReturnObjList,
                      &xc);
```

図 107. C 言語による FLGListOrphans の呼び出しのサンプル

図108 は、182ページの図107 にある FLGListOrphans 呼び出しのための出力構造を示しています。

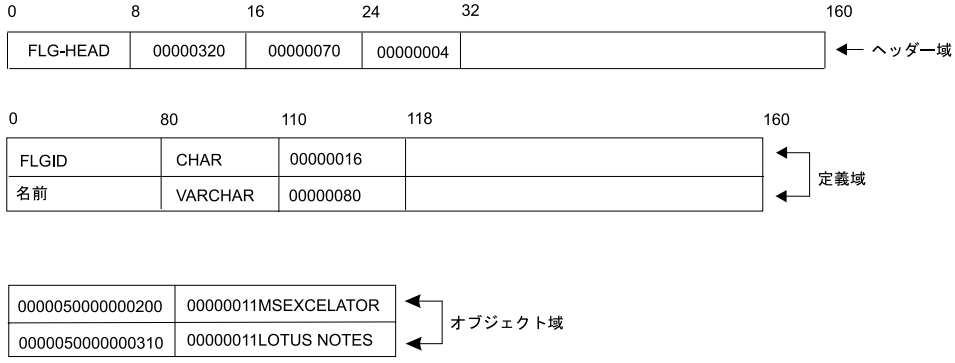


図 108. FLGListOrphans のための出力構造のサンプル

このサンプル・コードは、オーファンであるすべての Contact 区分のオブジェクト・インスタンスを検索します。 図109 は、FLGListOrphans 呼び出しを出すために必要な C 言語コードを示しています。

```
APIRET         rc;                        // reason code
PFLGHEADERAREA * ppReturnObjList;   // ptr to output structure ptr
FLGOPTIONS       Option=0;
FLGEXTCODE       xc=0;              // extended code
  .
  .
  .
Option=Option | FLG_LIST_CONTACT;
rc = FLGListOrphans (NULL,
                     Option,
                     ppReturnObjList,
                      &xc);
```

図 109. C 言語による FLGListOrphans の呼び出しのサンプル

184ページの図110 は、図109 にある FLGListOrphans 呼び出しのための出力構造を示しています。

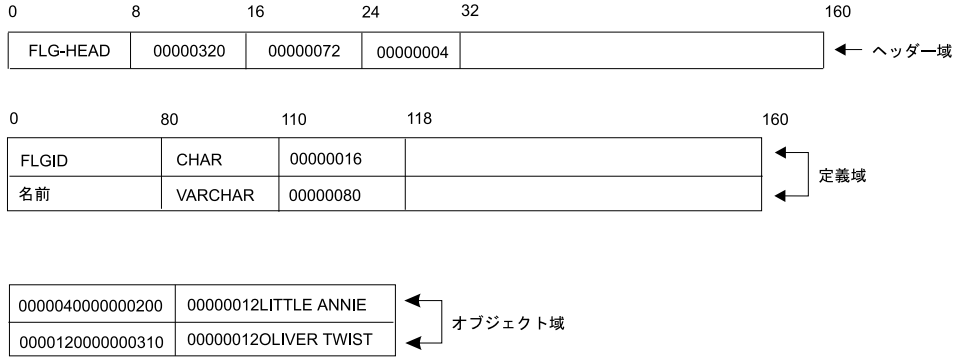


図 110. FLGListOrphans のための出力構造のサンプル

このサンプル・コードは、オーファンであるすべての Attachment 区分のオブジェクト・インスタンスを検索します。 図111 は、FLGListOrphans 呼び出しを出すために必要な C 言語コードを示しています。

```
APIRET         rc;                        // reason code
PFLGHEADERAREA * ppReturnObjList;   // ptr to output structure ptr
FLGOPTIONS       Option=0;
FLGEXTCODE       xc=0;              // extended code
  .
  .
  .
Option=Option | FLG_LIST_ATTACHMENT;
rc = FLGListOrphans (NULL,
                     Option,
                     ppReturnObjList,
                      &xc);
```

図 111. C 言語による FLGListOrphans の呼び出しのサンプル

185ページの図112 は、図111 にある FLGListOrphans 呼び出しのための出力構造を示しています。

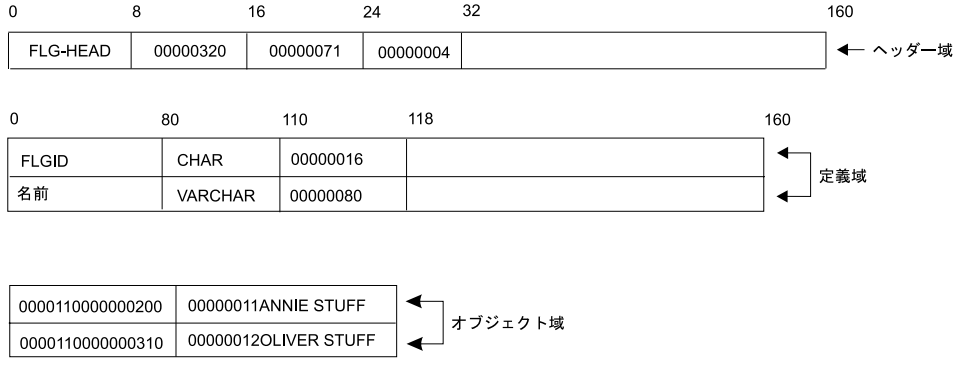


図 112. FLGListOrphans のための出力構造のサンプル

このサンプル・コードは、Comments オブジェクト・タイプである、すべてのオーファン Attachment 区分オブジェクト・インスタンスを検索します。 図 113 は、FLGListOrphans 呼び出しを出すために必要な C 言語コードを示しています。

```
APIRET         rc;                        // reason code
PFLGHEADERAREA * ppReturnObjList;   // ptr to output structure ptr
FLGOPTIONS       Option=0;
FLGEXTCODE       xc=0;             // extended code
  .
  .
  .
Option=Option | FLG_LIST_COMMENTS;
rc = FLGListOrphans (NULL,
                     Option,
                     ppReturnObjList,
                     &xc);
```

図 113. C 言語による FLGListOrphans の呼び出しのサンプル

186ページの図114 は、図113 にある FLGListOrphans 呼び出しのための出力構造を示しています。この特別な出力構造には、2 つの付加的な特性値があります。

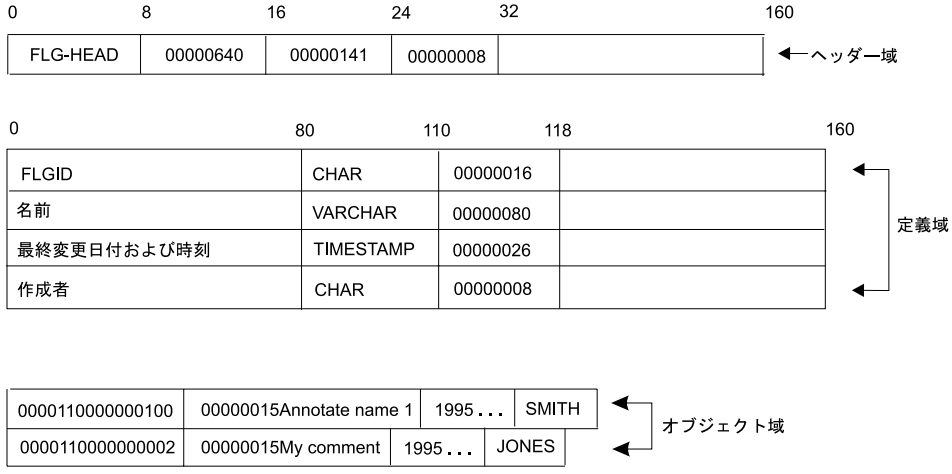


図 114. FLGListOrphans のための出力構造のサンプル

FLGListPrograms

Program 以外のオブジェクト・タイプに関する Programs オブジェクトのリストを検索します。

許可

管理者またはユーザー

構文

```
APIRET  APIENTRY  FLGListPrograms( PSZ              pszObjTypeID,
                                   PFLGHEADERAREA * ppListStruct,
                                    PFLGEXTCODE  pExtCode );
```

パラメーター

pszObjTypeID (PSZ) – 入力
関連する Programs オブジェクトのリストを検索するためにオブジェクト・タイプを表す 6 文字のシステム生成された固有の識別子 (オブジェクト・タイプ ID) を指します。

ppListStruct (PFLGHEADERAREA) – 出力
Programs インスタンスをリストする出力構造を指すポインターのアドレスを指します。出力構造がない場合には、構造を指すポインターは NULL にセットされます。

この出力構造には、Programs オブジェクト・インスタンスの 16 文字の FLGID と 80 文字の外部名が入ります。

リスト内の項目は外部名 (名前特性の値) によって分類されます。実際の順序は、情報カタログに使用されるデータベース管理システムで使用している照合順序に従います。

FLGListPrograms によって戻すことのできる Programs オブジェクト・インスタンスの最大数は、計算機で使用可能な記憶域によって異なり、約 5000 です。

pExtCode (PFLGEXTCODE) – 出力
理由コードに関連する拡張コードを指します。戻された理由コードに関連した意味のある拡張コードがあるかどうかについては、285ページの『付録 D. 情報カタログ・マネージャーの理由コード』を参照してください。

理由コード (APIRET)

この API 呼び出しの実行結果を表します。

戻された理由コードの説明については、285ページの『付録D. 情報カタログ・マネージャーの理由コード』を参照してください。

出力構造

FLGListPrograms は、図115 に示すように、Programs オブジェクトのリストが入っている出力構造を作成します。

出力構造のオブジェクト域には、指定されたオブジェクト・タイプに関連するすべての Programs オブジェクトのリストが入ります。これらの Programs オブジェクトは、オブジェクト・インスタンスの FLGID の値と外部名によって識別されます。

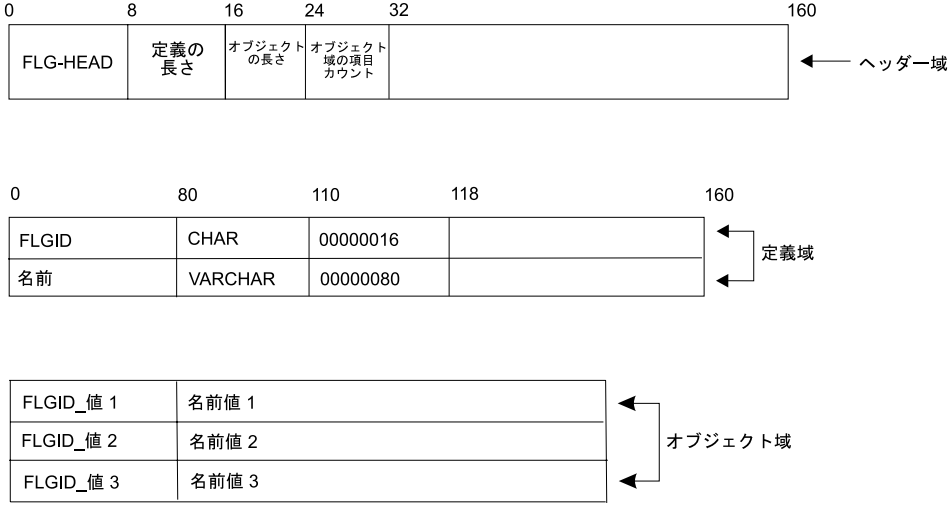


図 115. FLGListPrograms 出力構造

使用方法

出力構造に割り振られたメモリーの解放

FLGListPrograms が出力構造にあるデータを戻した場合、出力構造に戻されたデータを保管してから、FLGFreeMem を呼び出さなくてはなりません (137ページの『FLGFreeMem』参照)。メモリーを解放するのに、たとえば C 言語指示のような他の方式は使用しないでください。

例

図116 は、FLGListPrograms API 呼び出しを出すために必要な C 言語コードを示しています。

このサンプル・コードは、REPORT という名前のオブジェクト・タイプが関連付けられているプログラムのリストを検索します。

REPORT とともに使用するために作成されたプログラムとして、 Read report と Update report の 2 つがあります。

```
APIRET          rc;               // reason code from FLGListPrograms
UCHAR           pszObjTypeID[FLG_OBJTYPID_LEN + 1];
PFLGHEADERAREA * ppListStruct;   // pointer to output structure pointer
FLGEXTCODE       ExtCode=0;              // extended code
  .
  .  /* set object type ID to ID of 'REPORT'                */
  .
rc = FLGListPrograms (pszObjTypeID,
                      ppListStruct,
                      &ExtCode);
```

図 116. C 言語による FLGListPrograms の呼び出しのサンプル

図117 は、この API 呼び出しのための出力構造を示しています。

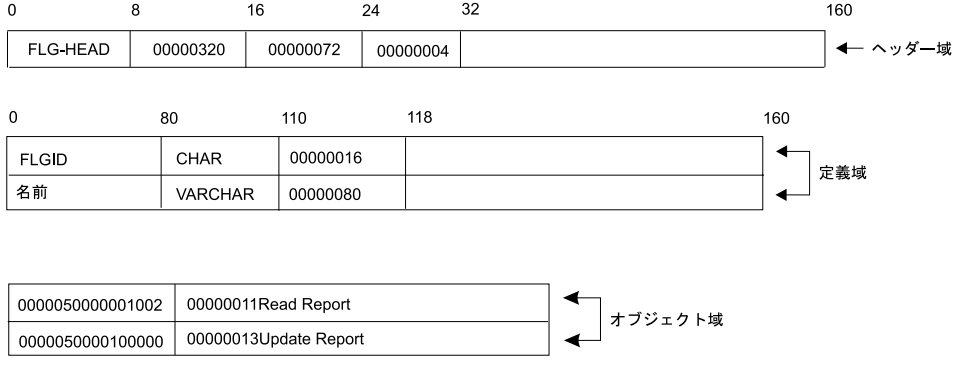


図 117. FLGListPrograms のための出力構造のサンプル

FLGManageCommentStatus

情報カタログ・マネージャー・インターフェースを使って、情報カタログで作成する Comments オブジェクトをユーザーが割り当てるのに、現在利用可能な選択項目のリストを設定します。たとえば、状況選択項目には、オープン、保留、必要なアクション、クローズなどがあります。

許可

管理者; ユーザー (FLG_ACTION_GET のみ)

構文

```
APIRET  APIENTRY   FLGManageCommentStatus( FLGOPTIONS       Action,
                                           FLGHEADERAREA  * pStatusStruct,
                                           PFLGHEADERAREA * ppStatusStruct,
                                  PFLGEXTCODE  pExtCode );
```

パラメーター

Action (FLGOPTIONS) – 入力

以下のいずれかのアクション・オプションを選択します。

FLG_ACTION_GET

Comments オブジェクト・インスタンスの現在の状況選択項目のリストを検索します。

FLG_ACTION_UPDATE

Comments オブジェクト・インスタンスの状況選択項目のリストから、項目を追加、変更、または削除します。

pStatusStruct (PFLGHEADERAREA) – 入力

FLG_ACTION_UPDATE の Comments オブジェクト・インスタンスの更新された状況選択項目リストを含む入力構造を指します。

ppStatusStruct (PFLGHEADERAREA) – 出力

FLG_ACTION_GET の Comments オブジェクト・インスタンスの状況選択項目の現在のリストを含む出力構造を指します。

pExtCode (PFLGEXTCODE) – 出力

理由コードに関連する拡張コードを指します。戻された理由コードに関連した意味のある拡張コードがあるかどうかについては、285ページの『付録 D. 情報カタログ・マネージャーの理由コード』を参照してください。

理由コード (APIRET)

この API 呼び出しの実行結果を表します。

戻された理由コードの説明については、285ページの『付録D. 情報カタログ・マネージャーの理由コード』を参照してください。

使用方法

FLGManageCommentStatus を呼び出すたびに、10 項目の定義域すべてと、それに対応するオブジェクト域にある 10 項目を組み込まなければなりません。ブランクにしておきたい状況域にはゼロを使用します (192ページの図 119 参照)。

出力構造に割り振られたメモリーの解放

FLGManageCommentStatus が出力構造にあるデータを戻した場合、出力構造に戻されたデータを保管してから、FLGFreeMem を呼び出さなくてはなりません (137ページの『FLGFreeMem』 参照)。メモリーを解放するのに、たとえば C 言語指示のような他の方式は使用しないでください。

情報カタログ更新の制御

プログラムを可能な限り、情報カタログと同期化させるために、FLGManageCommentStatus が正常に完了したあとで、呼び出しを FLGCommit (82ページの『FLGCommit』参照) に組み込む必要があります。FLGManageCommentStatus が正常に完了しない場合は、呼び出しを FLGRollback (225ページの『FLGRollback』参照) に組み込む必要があります。

例

このサンプル・コードは、状況構造を検索します。 192ページの図118 は、FLGManageCommentStatus 呼び出しを出すために必要な C 言語コードを示しています。

```
APIRET           rc;               // reason code for API
FLGOPTIONS       Action=0;
PFLGHEADERAREA   pStatusStruct;
FLGEXTCODE       xc=0;             // extended code
  .
  .  /*                                                */
  .
Action= Action | FLG_ACTION_GET;  //set get option
rc = FLGManageCommentStatus (Action,
                      NULL,
                            &pStatusStruct,
                     &xc);
```

図 118. C 言語による FLGManageCommentStatus の呼び出しのサンプル

図119 は、図118 にある FLGManageCommentStatus 呼び出しのための出力構造を示しています。

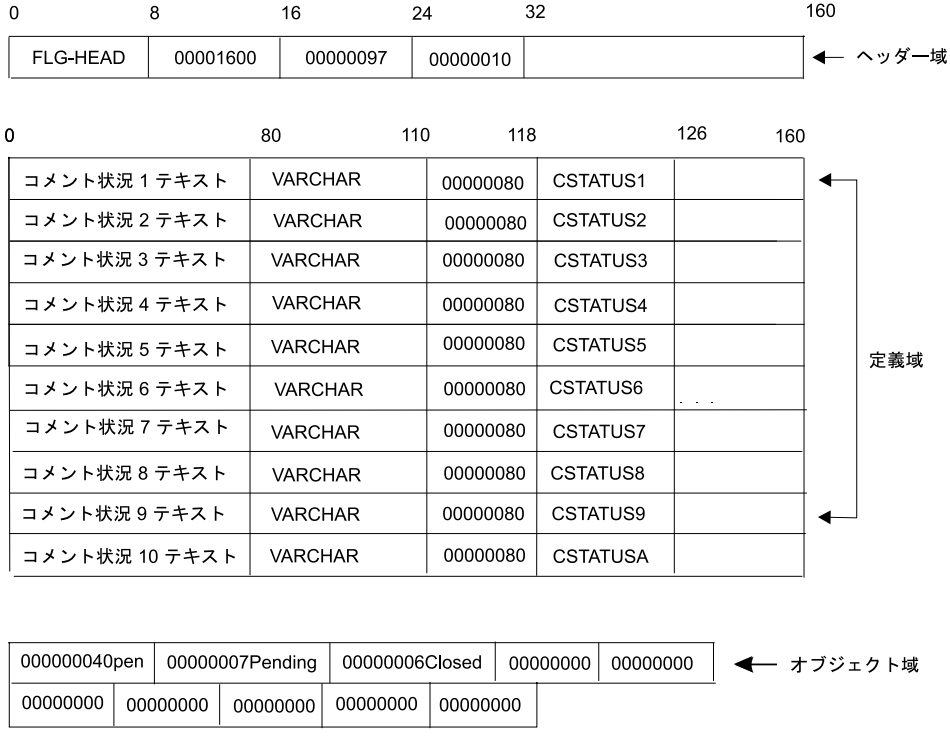


図 119. FLGManageCommentStatus のための出力構造のサンプル

このサンプル・コードは、状況構造に、追加の状況フィールドを付けて、更新します。 図120 は、FLGManageCommentStatus 呼び出しを出すために必要な C 言語コードを示しています。

```
APIRET           rc;               // reason code for API
FLGOPTIONS       Action=0;
PFLGHEADERAREA   pStatusStruct;
FLGEXTCODE       xc=0;             // extended code
  .
  .  /*                                                  */
  .
Action= Action | FLG_ACTION_UPDATE;  //update option
rc = FLGManageCommentStatus (Action,
                             pStatusStruct,
                      NULL,
                     &xc);
```

図 120. C 言語による FLGManageCommentStatus の呼び出しのサンプル

194ページの図121 は、図120 にある FLGManageCommentStatus 呼び出しのための入力構造を示しています。

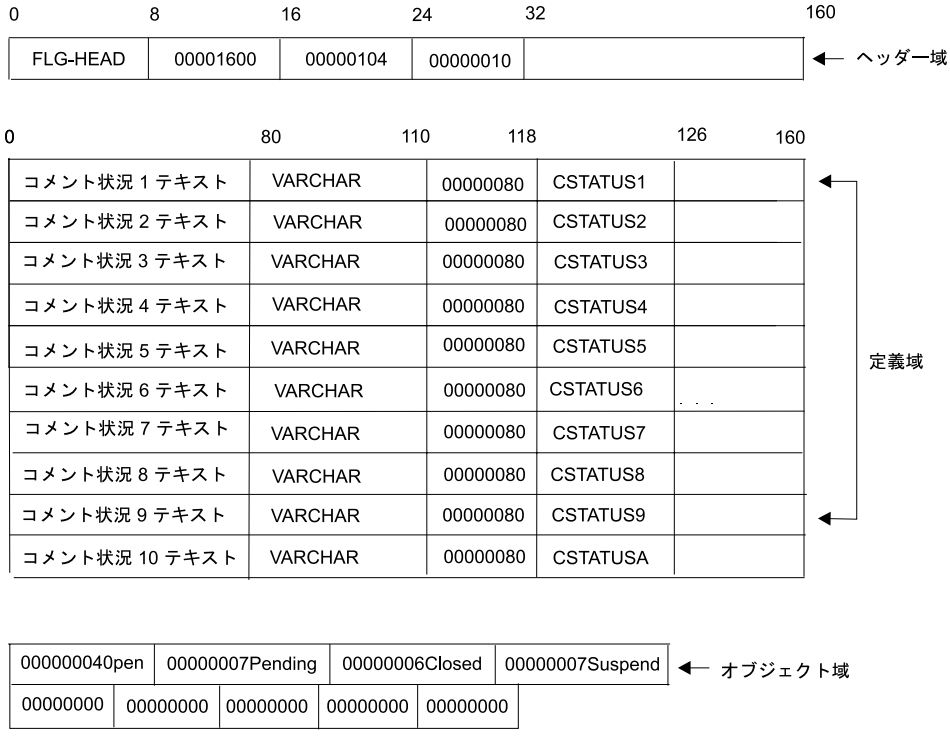


図 121. FLGManageCommentStatus のための入力構造のサンプル

FLGManageFlags

削除活動の記録を、照会、開始、または停止します。*削除履歴* とは、オンにしたり、オフにしたりできる削除活動のログのことです。

許可

管理者; ユーザー (FLG_ACTION_GET のみ)

構文

```
APIRET  APIENTRY   FLGManageFlags( FLGOPTIONS   Action,
                                   FLGOPTIONS   FlagType,
                                   UCHAR        chValue,
                                   UCHAR      * pchValue,
                                  PFLGEXTCODE  pExtCode );
```

パラメーター

Action (FLGOPTIONS) – 入力

以下のいずれかのアクション・オプションを選択します。

FLG_ACTION_GET

削除履歴のロギングが現在、使用可能か使用不能かどうかを示します。

FLG_ACTION_UPDATE

削除履歴のロギングをオン、またはオフにします。

FlagType (FLGOPTIONS) – 入力

フラグ・タイプを示します。この値は、FLG_HISTORY_TYPE_DELETE でなければなりません。

chValue (UCHAR) – 入力

FLG_ACTION_UPDATE に望まれるフラグ値を示します。以下のいずれかのフラグを選択します。

FLG_YES

削除履歴のロギングを使用可能にします。

FLG_NO

削除履歴のロギングを使用不能にします。

pchValue (UCHAR) – 出力

FLG_ACTION_GET によって戻される、以下のいずれかの状況を指します。

FLG_YES
削除履歴のロギングが使用可能になりました。

FLG_NO
削除履歴のロギングが使用不能になりました。

pExtCode (PFLGEXTCODE) － 出力
理由コードに関連する拡張コードを指します。戻された理由コードに関連した意味のある拡張コードがあるかどうかについては、285ページの『付録D. 情報カタログ・マネージャーの理由コード』を参照してください。

理由コード (APIRET)
この API 呼び出しの実行結果を表します。

戻された理由コードの説明については、285ページの『付録D. 情報カタログ・マネージャーの理由コード』を参照してください。

使用方法

情報カタログ更新の制御

プログラムを可能な限り、情報カタログと同期化させるために、FLGManageFlags が正常にフラグを更新したあとで、呼び出しを FLGCommit (82ページの『FLGCommit』参照) に組み込む必要があります。FLGManageFlags が、正常にフラグを更新しない場合は、呼び出しを FLGRollback に組み込む必要があります (225ページの『FLGRollback』参照)。

例

197ページの図122 は、FLGManageFlags 呼び出しを出すために必要な C 言語コードを示しています。このサンプル・コードは、削除履歴のロギングを可能にします。

```
APIRET           rc;               // reason code for API
FLGOPTIONS       Action=0;
FLGOPTIONS       Type=0;
UCHAR            chValue=FLG_YES;
FLGEXTCODE       xc=0;             // extended code
  .
  .
  .
Action= Action | FLG_ACTION_UPDATE;
Type = Type | FLG_HISTORY_TYPE_DELETE;
rc = FLGManageFlags (Action,
                        Type,
                        chValue,
                    NULL,
                    &xc);
```

図 122. C 言語による FLGManageFlags の呼び出しのサンプル

FLGManageIcons

代表的な OS/2 または Windows のアイコンを作成、削除、入手、照会、または更新します。

許可

管理者; ユーザー (FLG_ACTION_GET および FLG_ACTION_QUERY のみ)

構文

```
APIRET  APIENTRY   FLGManageIcons( PSZ          pszObjTypeID,
                                   PSZ            pszIconFileID,
                                     FLGOPTIONS   InOptions,
                                      PFLGOPTIONS      pOutOptions,
                                    PFLGEXTCODE  pExtCode );
```

パラメーター

pszObjTypeID (PSZ) − 入力
アイコンを検索、照会、作成、更新、または削除したいオブジェクト・タイプの 6 文字のシステム生成された固有の識別子 (オブジェクト・タイプ ID) を指します。

pszIconFileID (PSZ) − 入力
指定されたオブジェクト・タイプの、検索、作成、または更新したい OS/2 または Windows のアイコンを含むファイルのドライブ、ディレクトリー・パス、およびファイル名 (FAT または HPFS ファイルに有効) を含みます。このパラメーターは、FLG_ACTION_QUERY および FLG_ACTION_DELETE では、無視されます。

InOptions (FLGOPTIONS) − 入力
望まれるアクションおよびプラットフォーム・オプションを示します。以下のいずれかのアクション・オプションを選択します。

FLG_ACTION_CREATE
指定されたアイコンを指定されたオブジェクト・タイプに追加します。

FLG_ACTION_DELETE
指定されたアイコンを指定されたオブジェクト・タイプから取り外します。

FLG_ACTION_GET
指定されたアイコン・ファイルを検索します。

FLG_ACTION_QUERY
指定されたアイコン・ファイルが存在するかどうかを判別します。
FLG_ACTION_UPDATE
アイコンを指定されたオブジェクト・タイプに変更します。

以下のいずれかのプラットフォーム・オプションを選択します。
FLG_PLATFORM_OS2
OS/2 アイコンを管理します。
FLG_PLATFORM_WINDOWS
Windows アイコンを管理します。

pOutOptions (PFLGOPTIONS) − 出力
FLG_ACTION_QUERY によって戻される、以下のいずれかの状況を指します。
FLG_ICON_EXIST
FLG_ICON_NOTEXIST

pExtCode (PFLGEXTCODE) − 出力
理由コードに関連する拡張コードを指します。戻された理由コードに関連した意味のある拡張コードがあるかどうかについては、285ページの『付録 D. 情報カタログ・マネージャーの理由コード』を参照してください。

理由コード (APIRET)
この API 呼び出しの実行結果を表します。

戻された理由コードの説明については、285ページの『付録D. 情報カタログ・マネージャーの理由コード』を参照してください。

使用方法

前提条件

FLGManageIcons を呼び出すまえに、FLGCreateReg を呼び出して、アイコンを管理したいオブジェクト・タイプを登録する必要があります。

情報カタログ更新の制御

プログラムを可能な限り、情報カタログと同期化させるために、FLGManageIcons が正常にアイコンを作成、更新、または削除したあとで、呼び出しを FLGCommit (82ページの『FLGCommit』参照) に組み込む必要があります。 FLGManageIcons が、正常にアイコンを作成、更新、または削除しない場合は、呼び出しを FLGRollback に組み込む必要があります (225ページの『FLGRollback』参照)。

例

図123 は、FLGManageIcons 呼び出しを出すために必要な C 言語コードを示しています。このサンプル・コードは、情報カタログ・マネージャーにおける Windows アイコンを更新します。

```
APIRET            rc;                // reason code from FLGManageIcons
UCHAR           pszObjTypeID[FLG_OBJTYPID_LEN + 1];
UCHAR             pszIconFileID[FLG_ICON_FILE_ID_MAXLEN + 1];
FLGOPTIONS        Options = 0;       // initialize option
FLGEXTCODE       xc=0;             // extended code
  .
  .  /* provide values for input parameters */
  .
Options = Options | FLG_ACTION_UPDATE | FLG_PLATFORM_WINDOWS;
rc = FLGManageIcons (pszObjTypeID,
                     pszIconFileID,
                    Options,
                     NULL,
                     &xc);
```

図 123. C 言語による FLGManageIcons の呼び出しのサンプル

FLGManageTagBuf

現在の削除履歴を照会、またはリセットします。*削除履歴* とは、オンにしたり、オフにしたりできる削除活動のログのことです。

許可

管理者

構文

```
APIRET  APIENTRY   FLGManageTagBuf( FLGOPTIONS       InOptions,
                                    PFLGOPTIONS      pOutOptions,
                                  PFLGEXTCODE  pExtCode );
```

パラメーター

InOptions (FLGOPTIONS) – 入力

以下のいずれかのオプションを選択します。

FLG_TAGBUF_QUERY

削除履歴ログが現在、項目を含んでいるかどうかを照会します。

FLG_TAGBUF_RESET

削除履歴ログから、存在している項目をどれも取り外します。

pOutOptions (PFLGOPTIONS) – 出力

FLG_TAGBUF_QUERY によって戻される、以下のいずれかの状況を指します。

FLG_TAGBUF_STATUS_EMPTY
FLG_TAGBUF_STATUS_NOT_EMPTY

pExtCode (PFLGEXTCODE) – 出力

理由コードに関連する拡張コードを指します。戻された理由コードに関連した意味のある拡張コードがあるかどうかについては、285ページの『付録D. 情報カタログ・マネージャーの理由コード』を参照してください。

理由コード (APIRET)

この API 呼び出しの実行結果を表します。

戻された理由コードの説明については、285ページの『付録D. 情報カタログ・マネージャーの理由コード』を参照してください。

使用方法

情報カタログ更新の制御

プログラムを可能な限り、情報カタログと同期化させるために、FLGManageTagBuf が正常に削除履歴をリセットしたあとで、呼び出しを FLGCommit (82ページの『FLGCommit』参照) に組み込む必要があります。FLGManageTagBuf が、正常に削除履歴をリセットしない場合、呼び出しを FLGRollback に組み込む必要があります (225ページの『FLGRollback』参照)。

例

図124 は、FLGManageTagBuf 呼び出しを出すために必要な C 言語コードを示しています。このサンプル・コードは、削除履歴の現在の内容を削除します。

```
APIRET         rc;                      // reason code
FLGOPTIONS       Opt1=0;           //option
FLGEXTCODE       xc=0;             // extended code
  .
  .  /*                                                 */
  .
Opt1=Opt1 | FLG_TAGBUF_RESET;      //set reset option
rc = FLGManageTagBuf (Opt1,
                      NULL,        // not used.
                      &xc);
```

図 124. C 言語による FLGManageTagBuf の呼び出しのサンプル

FLGManageUsers

組織内の指定された情報カタログ・マネージャー・ユーザーに、通常は情報カタログ・マネージャー管理担当者によって実行される、以下のオブジェクト管理タスクを実行することを許可します。

- オブジェクトの作成
- オブジェクトの削除
- オブジェクトの更新
- オブジェクトのコピー
- オブジェクトのエクスポート
- 連絡先の関連づけ
- オブジェクト間のリンクの更新
- オブジェクトのグループの更新
- プログラムとオブジェクトとの関連づけ

FLGManageUsers は、情報カタログの 1 次およびバックアップ管理者の更新もします。

許可

管理者

構文

```
APIRET  APIENTRY   FLGManageUsers( FLGOPTIONS       Options,
                                    PFLGHEADERAREA   pListStruct,
                                  PFLGHEADERAREA * ppListStruct,
                                   PFLGEXTCODE  pExtCode );
```

パラメーター

Action (FLGOPTIONS) − 入力

以下のいずれかのアクション・オプションを選択します。

FLG_ACTION_CREATE

指定されたユーザーを、現在の情報カタログの付加的なオブジェクト管理タスクの実行を許可されたユーザーのリストに追加します。

FLG_ACTION_UPDATE

1 次またはバックアップ管理者を変更します。

FLG_ACTION_DELETE
指定されたユーザーを、現在の情報カタログの付加的なオブジェクト管理タスクの実行を許可されたユーザーのリストから取り外します。

FLG_ACTION_LIST
以下のリストを戻します。

管理者

バックアップ管理者

現在の情報カタログの付加的なオブジェクト管理タスクの実行を許可されたユーザー

pListStruct (PFLGHEADERAREA) − 入力
新規、変更、または削除されたユーザー ID を含む入力構造を指します。

ppListStruct (PFLGHEADERAREA) − 出力
1 次およびバックアップ管理者と、現在の情報カタログの付加的なオブジェクト管理タスクの実行を許可されたすべてのユーザーをリストする出力構造を指すポインターのアドレスを指します。

出力構造における各項目には以下の情報が入ります。
- USERID (8 文字)
- ユーザー・タイプ (1 文字) フラグ
  - **A** USERID は、1 次管理者です (FLG_USERTYPE_PADMIN)
  - **B** USERID は、バックアップ管理者です (FLG_USERTYPE_BADMIN)
  - **D** USERID は、付加的なオブジェクト管理タスクの実行を許可されたユーザーです (FLG_USERTYPE_POWERUSER)

すべてのユーザーは、基礎となるデータベース管理システムの照合順序の昇順で、初めにユーザー・タイプ、次に USERID で分類されます。

出力構造がない場合には、構造を指すポインターは NULL にセットされます。

pExtCode (PFLGEXTCODE) − 出力
理由コードに関連する拡張コードを指します。戻された理由コードに関連した意味のある拡張コードがあるかどうかについては、285ページの『付録 D. 情報カタログ・マネージャーの理由コード』を参照してください。

理由コード (APIRET)
この API 呼び出しの実行結果を表します。

戻された理由コードの説明については、285ページの『付録D. 情報カタログ・マネージャーの理由コード』を参照してください。

使用方法

制約事項

情報カタログ・マネージャーは、1 次およびバックアップ管理者を、それぞれ 1 人しか認めておらず、管理者だけが FLGManageUsers を呼び出すことができます。 FLGManageUsers が、ログオンされた管理者ユーザー ID に影響する場合、この変更は、現在の管理者がログオフするまで実行されません。

出力構造に割り振られたメモリーの解放

FLGManageUsers が出力構造にあるデータを戻した場合、出力構造に戻されたデータを保管してから、FLGFreeMem を呼び出さなくてはなりません (137ページの『FLGFreeMem』参照)。メモリーを解放するのに、たとえば C 言語指示のような他の方式は使用しないでください。

情報カタログ更新の制御

プログラムを可能な限り、情報カタログと同期化させるために、FLGManageUsers が正常にユーザーを作成、更新、または削除したあとで、呼び出しを FLGCommit (82ページの『FLGCommit』参照) に組み込む必要があります。 FLGManageUsers が、正常にユーザーを作成、更新、または削除しない場合は、呼び出しを FLGRollback に組み込む必要があります (225ページの『FLGRollback』参照)。

例

このサンプル・コードは、2 人のユーザーを、付加的なオブジェクト管理タスクの実行を許可された管理者およびユーザーのリストに追加します。 図125 は、FLGManageUsers 呼び出しを出すために必要な C 言語コードを示しています。

```
APIRET           rc;               // reason code for API
FLGOPTIONS       Action=0;
PFLGHEADERAREA   pInList;
FLGEXTCODE       xc=0;             // extended code
  .
  .  /*                                                  */
  .
Action= Action | FLG_ACTION_CREATE;
rc = FLGManageUsers (Action,
                     pInList,
                     NULL,
                     &xc);
```

図 125. C 言語による FLGManageUsers の呼び出しのサンプル

図126 は、205ページの図125 にある FLGManageUsers 呼び出しのための入力構造を示しています。

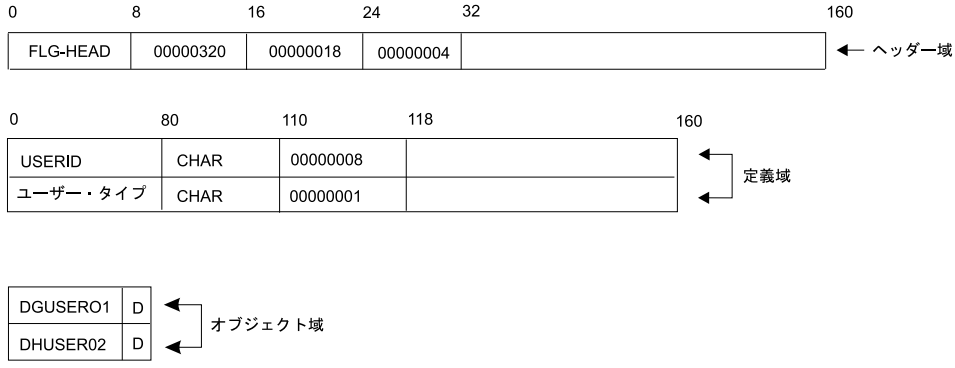


図 126. FLGManageUsers のための入力構造のサンプル

このサンプル・コードは、付加的なオブジェクト管理タスクの実行を許可されたユーザーの現在のリストを検索します。 図127 は、FLGManageUsers 呼び出しを出すために必要な C 言語コードを示しています。

```
APIRET           rc;               // reason code for API
FLGOPTIONS       Action=0;
PFLGHEADERAREA  * ppOutList;
FLGEXTCODE       xc=0;             // extended code
  .
  .  /*                                                */
  .
Action= Action | FLG_ACTION_LIST;
rc = FLGManageUsers (Action,
                     NULL,
                     ppOutList,
                     &xc);
```

図 127. C 言語による FLGManageUsers の呼び出しのサンプル

207ページの図128 は、図127 にある FLGManageUsers 呼び出しのための出力構造を示しています。

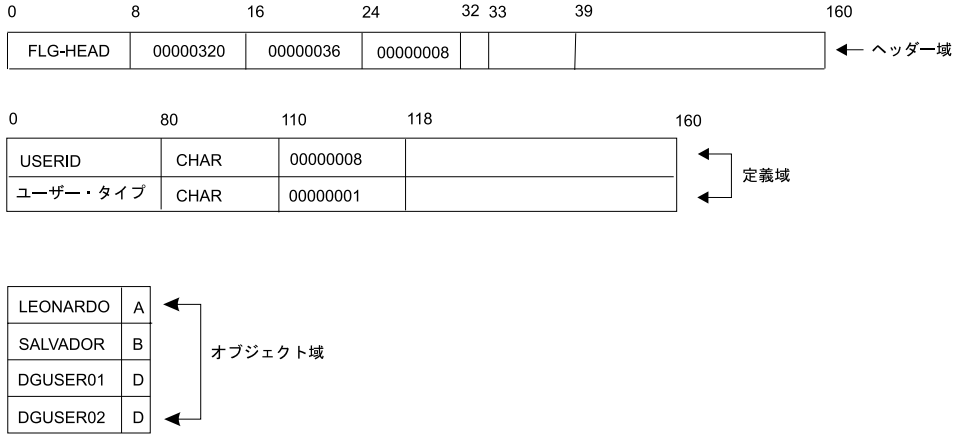


図 128. FLGManageUsers のための出力構造のサンプル

このサンプル・コードは、1 次管理者を更新します。 図129 は、この FLGManageUsers 呼び出しを出すのに必要な C 言語コードを表示しています。

```
APIRET           rc;               // reason code for API
FLGOPTIONS       Action=0;
PFLGHEADERAREA   pInList;
FLGEXTCODE       xc=0;             // extended code
  .
  .  /*                                                  */
  .
Action= Action | FLG_ACTION_UPDATE;
rc = FLGManageUsers (Action,
                     pInList,
                     NULL,
                     &xc);
```

図 129. C 言語による FLGManageUsers の呼び出しのサンプル

208ページの図130 は、図129 にある FLGManageUsers 呼び出しのための入力構造を示しています。 1 次管理者だけが更新されるので、バックアップ管理者は元のままです。

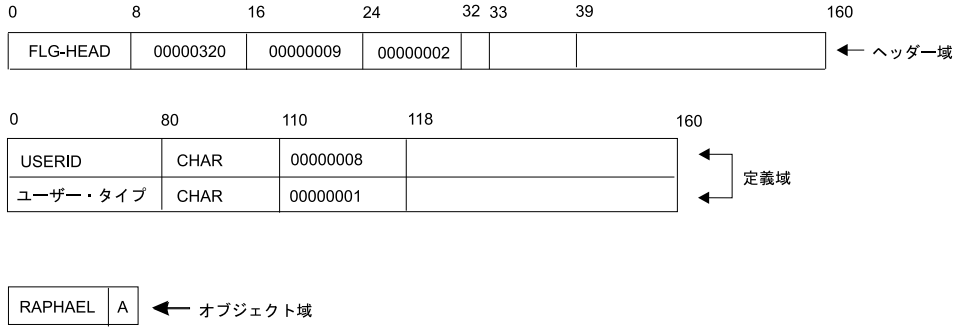


図 130. FLGManageUsers のための入力構造のサンプル

FLGMdisExport

情報カタログから MDIS 適合メタデータを検索して MDIS 適合ファイルに変換するために使用します。 MDIS メタデータをエクスポートするときの情報カタログは、 MDIS メタデータを含むものに限定されてはいませんが、FLGMdisExport は MDIS 適合メタデータだけをエクスポートします。

許可

管理者または許可ユーザー

構文

```
APIRET  APIENTRY   FLGExport( PSZ              pszTagFileName,
                              PSZ              pszLogFileName,
                              PSZ              pszObjTypeName,
                              PSZ              pszObjectName,
                                  PFLGEXTCODE  pExtCode );
```

パラメーター

pszTagFileName (PSZ) − 入力
出力タグ言語ファイルの名前。このパラメーターは必須パラメーターです。

このパラメーターにはドライブ、ディレクトリー・パス、およびファイル名が含まれており、ファイル割り振りテーブル (FAT) または HPFS ファイルとして有効な値でなければなりません。このファイルのターゲット・ドライブは、固定ディスクでなければなりません。ファイル名だけを入力すると、情報カタログ・マネージャー側は、MDIS 適合ファイルを DGWPATH 環境変数で示されるドライブおよびパスに配置します。

このターゲット MDIS 適合ファイルが存在していてはなりません。情報カタログ・マネージャーは既存のファイルへの上書きは行いません。

(ドライブおよびディレクトリーを除く) ファイル名とエクステンションは、 240 文字を超えてはなりません。 MDIS タグ・ファイル名全体が、259 文字を超過しないようにします。

pszLogFileName (PSZ) − 入力
ログ・ファイルの名前を指します。このパラメーターは必須パラメーターです。

このパラメーターにはドライブ、ディレクトリー・パス、およびファイル名が含まれており、FAT または HPFS ファイルとして有効な値でなければなりません。 ログ・ファイルのターゲット・ドライブは、固定ディスクでなければなりません。ログ・ファイル名は 259 文字を超えてはなりません。ファイル名だけを指定すると、情報カタログ・マネージャー側は、ログ・ファイルを DGWPATH 環境変数で示されるドライブおよびパスに配置します。

このパラメーターで指定されたログ・ファイルが存在していない場合には、新しいファイルが作成されます。 このパラメーターで指定されたログ・ファイルがすでに存在している場合には、 FLGMdisExport API 呼び出しはそれに対して行われます。

pszObjTypeName (PSZ) – 入力

エクスポートする以下のいずれかの MDIS オブジェクト・タイプを指定します。

- Database
- Dimension
- Subschema
- Record
- Element

オブジェクト・タイプの名前は、大文字小文字を区別しません。

pszObjectName (PSZ) – 入力

エクスポートするオブジェクトを指定します。 pszObjectName の値は、pszObjTypeName パラメーターに指定したオブジェクト・タイプに応じ、3 から 5 のプロパティー値になります。それぞれの値は、ピリオド (.) で区切ります。

pszObjTypeName	pszObjectName
Database	*ServerName.DatabaseName.OwnerName*
Dimension	*ServerName.DatabaseName.OwnerName.DimensionName*
Subschema	*ServerName.DatabaseName.OwnerName.SubschemaName*
Record	*ServerName.DatabaseName.OwnerName.RecordName*
Element	*ServerName.DatabaseName.OwnerName.RecordName.ElementName*

このリストでは、名前のパートはそれぞれの MDIS 名で表されます。対応する情報カタログ・マネージャー名を知るには、*情報カタログ・マネージャー 管理の手引き* の付録 B を参照してください。

1. エクスポートするオブジェクト・タイプの表を見つけます。
2. **「MDIS 名へのマップ (Maps to MDIS name)」**欄で MDIS 名を探します。
3. **「特性名 (Property name)」**および**「特性省略名 (Property short name)」**で、対応する情報カタログ・マネージャー名を見つけます。

それぞれのパートごとに、エクスポートするオブジェクト用に指定したプロパティーの値を入力します。アスタリスク (*) をワイルドカードとしてパートの中に含めたり、パート全体を表すものとして使うことができます。必須のパートに何も入力しないと、情報カタログ・マネージャーは、エクスポートするオブジェクトを探すときに非適用記号を使うことになります。 (非適用記号は、情報カタログの作成時に特定の記号を指定しない限り、ハイフンになります。) 任意指定のパートに何も入力しない場合、情報カタログ・マネージャーは、エクスポートするオブジェクトの検索に NULL 文字を使います。

pExtCode (PFLGEXTCODE) － 出力

理由コードに関連する拡張コードを指します。戻された理由コードに関連した意味のある拡張コードがあるかどうかについては、285ページの『付録 D. 情報カタログ・マネージャーの理由コード』を参照してください。

理由コード (APIRET)

この API 呼び出しの実行結果を表します。

戻された理由コードの説明については、285ページの『付録D. 情報カタログ・マネージャーの理由コード』を参照してください。

FLGMdisImport

メタデータを、メタデータ交換仕様 (MDIS) に適合するファイルから情報カタログ・マネージャーへインポートします。 MDIS メタデータのインポート先の情報カタログには、有効な MDIS オブジェクト・タイプ定義が含まれていなければなりませんが、それに限定されることはありません。*情報カタログ・マネージャー 管理の手引き* の付録 B には、情報カタログ・マネージャー定義済みオブジェクト・タイプとそれらを MDIS にマップする方法が説明されています。

許可

管理者

構文

```
APIRET  APIENTRY   FLGMdisImport( PSZ              pszTagFileID,
                                  PSZ              pszLogFileID,
                                  PFLGEXTCODE  pExtCode );
```

パラメーター

pszTagFileID (PSZ) – 入力

タグ言語ファイルを識別します。このパラメーターは必須パラメーターです。

このパラメーターにはドライブ、ディレクトリー・パス、およびファイル名が含まれており、FAT または HPFS ファイルとして有効な値でなければなりません。このドライブは、取り外し可能ドライブであってはなりません。 (ドライブとディレクトリーを除く) ファイル名とエクステンションは、 240 文字を超えてはなりません。ファイル名だけを入力すると、情報カタログ・マネージャー側は、タグ言語ファイルを DGWPATH 環境変数で示されるドライブおよびパスに存在するものとみなします。

pszTagFileID によって識別されるファイルには、インポートされる MDIS 適合メタデータが入っています。

pszLogFileID (PSZ) – 入力

ログ・ファイルのロケーションと名前を指定します。このパラメーターは必須パラメーターです。

このパラメーターにはドライブ、ディレクトリー・パス、およびファイル名が含まれており、FAT または HPFS ファイルとして有効な値でなければ

なりません。 このドライブは、取り外し可能ドライブであってはなりません。ファイル名だけを指定すると、情報カタログ・マネージャー側は、ログ・ファイルを DGWPATH 環境変数で示されるドライブおよびパスに配置します。

このパラメーターで指定されたログ・ファイルが存在していない場合には、新しいファイルが作成されます。このパラメーターで指定されたログ・ファイルがすでに存在している場合には、情報カタログ・マネージャーはそれに対して行われます。

pszLogFileID によって識別されたファイルには、ログ情報、および FLGMdisImport API 呼び出しの処理中に検出された警告とエラーが入ります。

pExtCode (PFLGEXTCODE) − 出力

理由コードに関連する拡張コードを指します。戻された理由コードに関連した意味のある拡張コードがあるかどうかについては、285ページの『付録 D. 情報カタログ・マネージャーの理由コード』を参照してください。

理由コード (APIRET)

この API 呼び出しの実行結果を表します。

戻された理由コードの説明については、285ページの『付録D. 情報カタログ・マネージャーの理由コード』を参照してください。

使用方法

MDIS 環境の設定

MDIS のインポートを実行する前に、 MDIS 環境変数を次のように設定します。

```
SET MDIS_PROFILE=X:¥SQLLIB¥METADATA¥PROFILES
```

X は DB2 UDB が導入されているドライブです。

他のプロダクトや Visual Warehouse 3.1 とともに MDIS を使用しているユーザーへの注: すでに MDIS 構成およびプロファイル・ファイルがある場合、DB2 UDB 導入プログラムはそれらを上書きしていません。ただし、初めて情報カタログ・マネージャーの MDIS 機能を使用する前に、情報カタログ・マネージャー MDIS プロファイルおよび構成ファイル内の情報を、既存のファイルにマージする必要があります。以下の手順を行います。

1. MDIS 環境変数の設定を調べ、既存の MDIS プロファイル・ファイル (MDISTOOL.PRO) と構成ファイル (MDISTOOL.CFG) を探し出します。

2. テキスト・エディターを使い、既存のプロファイル・ファイルに X:¥SQLLIB¥METADATA¥PROFILES¥MDISTOOL.PRO の内容を追加します。 (X は DB2 UDB を導入したドライブです。)
3. テキスト・エディターを使い、既存の構成ファイルに X:¥SQLLIB¥METADATA¥PROFILES¥MDISTOOL.CFG の内容を追加します。 (X は DB2 UDB を導入したドライブです。)

MDIS インポート・エラーのデバッグ

情報カタログ・マネージャーは、MDIS 適合ファイルをインポートするときに、ログ・ファイルを作成します。

ログ・ファイルには、インポート処理で発生した事象が記録されます。これには、インポート処理が開始および終了した日付と時刻が含まれます。また、処理中に発生した問題に関する警告やエラー・メッセージも含まれます。 pszLogFileID パラメーターによりログ・ファイルが識別されます。

FLGNavigate

特定の Grouping オブジェクトに含まれるオブジェクトのリストを検索するために使用します。

許可

管理者またはユーザー

構文

```
APIRET  APIENTRY    FLGNavigate( PSZ              pszFLGID,
                                 PFLGHEADERAREA * ppListStruct,
                                  PFLGEXTCODE  pExtCode );
```

パラメーター

pszFLGID (PSZ) − 入力
含まれているオブジェクトが検索されるオブジェクト・インスタンスを表す 16 文字の FLGID を指します。

この ID の 1 文字目から 6 文字目までは、このインスタンスのオブジェクト・タイプを識別します。

この ID の 7 文字目から 16 文字目までは、システム生成された固有のインスタンス ID です。

ppListStruct (PFLGHEADERAREA) − 出力
出力構造を指すポインターのアドレスを指します。出力構造がない場合、出力構造へのポインターは NULL にセットされます。

pExtCode (PFLGEXTCODE) − 出力
理由コードに関連する拡張コードを指します。戻された理由コードに関連した意味のある拡張コードがあるかどうかについては、285ページの『付録D. 情報カタログ・マネージャーの理由コード』を参照してください。

理由コード (APIRET)
この API 呼び出しの実行結果を表します。

戻された理由コードの説明については、285ページの『付録D. 情報カタログ・マネージャーの理由コード』を参照してください。

出力構造

FLGNavigate が作成する出力構造には、図131 に示すように、指定されたオブジェクトに含まれるオブジェクトのリストが入っています。

出力構造のオブジェクト域には、指定されたオブジェクト・インスタンスに含まれるすべてのオブジェクト・インスタンスのリストが入ります。戻されたオブジェクト・インスタンスは、FLGID の値、オブジェクト・インスタンスの外部名、および子インディケーターによって識別されます。子インディケーターは、そのオブジェクトが別のオブジェクトを含んでいるかどうかを示します。

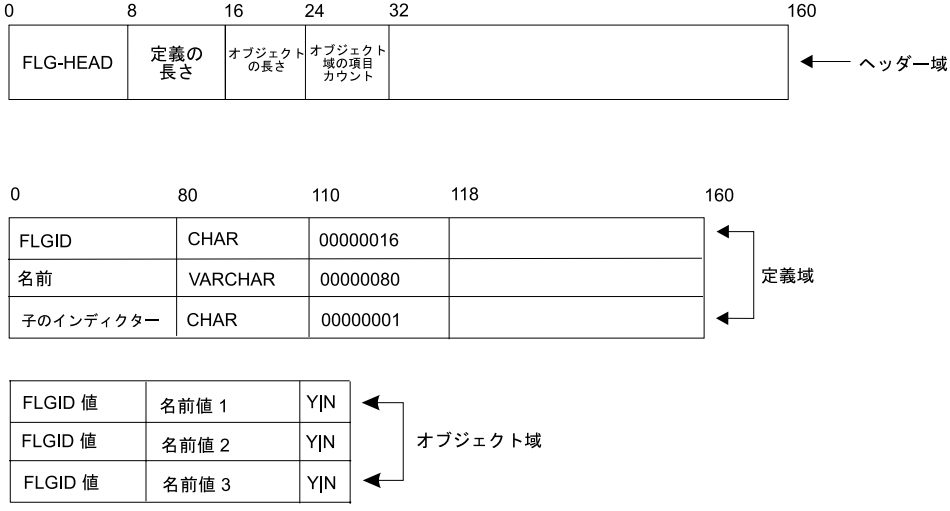


図 131. FLGNavigate 出力構造

バイト・オフセットの意味については、35ページの『第4章 情報カタログ・マネージャー入力および出力構造』を参照してください。

使用方法

この出力構造には、戻された各インスタンスに関する以下の特性値が入ります。

FLGID オブジェクト・インスタンスの 16 文字の識別子。

Name オブジェクト・インスタンスの 80 バイトの外部名。

CHILDIND
オブジェクト・インスタンスが別のオブジェクト・インスタンスを含んでいるかどうかを示す、1 文字の値。Y は含んでいることを表し、N は含んでいないことを表します。

出力リストは、使用しているデータベース管理システムで採用されている照合順序に従って、最初にオブジェクト・タイプ名によって分類され、次に 80 バイトのオブジェクト・インスタンス名によって分類されます。

FLGNavigate によって戻すことのできる、含まれているオブジェクト・インスタンスの最大数は計算機で使用可能な記憶域によって異なり、約 5000 です。

出力構造に割り振られたメモリーの解放

FLGNavigate が出力構造にあるデータを戻した場合、出力構造に戻されたデータを保管してから、FLGFreeMem を呼び出さなくてはなりません (137ページの『FLGFreeMem』参照)。メモリーを解放するのに、たとえば C 言語指示のような他の方式は使用しないでください。

例

218ページの図133 は、別のオブジェクトを含んでいるオブジェクトの構造をナビゲートします。この例の ACCOUNTING には、ACCPAYABLE、 ACCRECBLE、および GLEDGER の 3 つのオブジェクトが含まれています。この構造は図132 のようになっています。

```
ACCOUNTING
   ACCPAYABLE
      PAYABLE1
      PAYABLE2
   ACCRECBLE
   GLEDGER
```

図 132. ACCOUNTING オブジェクトの内容

218ページの図133 は、FLGNavigate API 呼び出しを出すために必要な C 言語コードを示しています。

```
APIRET          rc;               // reason code from FLGNavigate
UCHAR           pszFLGID[FLG_ID_LEN + 1];
PFLGHEADERAREA  * ppListStruct;
FLGEXTCODE     ExtCode = 0;               // Declare extended code
  .
  .  /* set pszParentID to FLGID of 'ACCOUNTING'          */
  .
rc = FLGNavigate (pszFLGID,
                  ppListStruct, // pass the address of
                                // output structure pointer
                    &ExtCode);
```

図 133. C 言語による FLGNavigate の呼び出しのサンプル

図134 は、この API 呼び出しの出力構造を示しています。

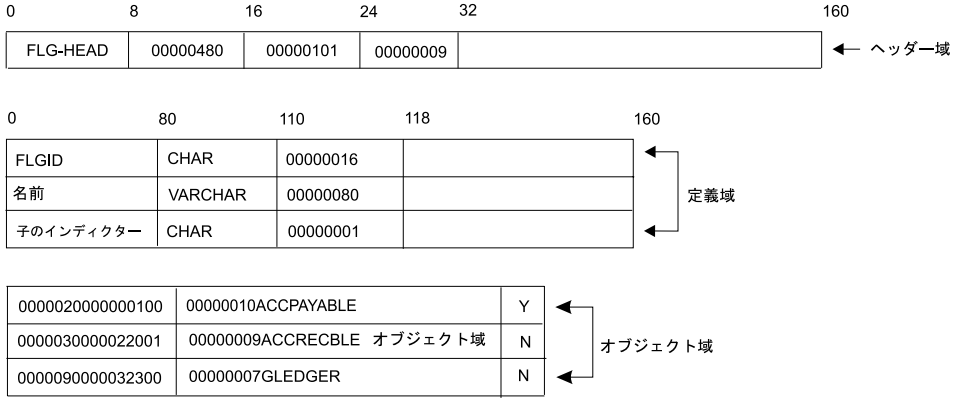


図 134. FLGNavigate のための出力構造のサンプル

FLGOpen

情報カタログ・マネージャーから外部プログラムを開始するために使用します。

許可

管理者またはユーザー

構文

```
APIRET  APIENTRY   FLGOpen( PSZ              pszPgmFLGID,
                            PSZ              pszObjFLGID,
                                  PFLGEXTCODE  pExtCode );
```

パラメーター

pszPgmFLGID (PSZ) – 入力
実行情報を含む Programs オブジェクト・インスタンスの 16 文字の FLGID を指します。この FLGID は、6 文字のオブジェクト・タイプ ID と、それに続く Programs オブジェクトの 10 文字のインスタンス ID からなります。

pszObjFLGID (PSZ) – 入力
パラメーター・リストに値を提供した非 Program 区分のオブジェクト・インスタンスの 16 文字の FLGID を指します。この FLGID は、6 文字のオブジェクト・タイプ ID とそれに続く 10 文字のインスタンス ID からなります。

pExtCode (PFLGEXTCODE) – 出力
理由コードに関連する拡張コードを指します。戻された理由コードに関連した意味のある拡張コードがあるかどうかについては、285ページの『付録D. 情報カタログ・マネージャーの理由コード』を参照してください。

理由コード (APIRET)
この API 呼び出しの実行結果を表します。

戻された理由コードの説明については、285ページの『付録D. 情報カタログ・マネージャーの理由コード』を参照してください。

使用方法

あるプログラムに関する FLGOpen 呼び出しを出す場合には、 29ページの『プログラムを開始するための Programs オブジェクトの設定』の説明に従ってそのプログラム・オブジェクトを設定しておく必要があります。

Programs オブジェクトで記述されたプログラムが開始されると、識別されたオブジェクト・インスタンスによって提供された呼び出しパラメーターが使用されます。情報カタログ・マネージャーは、呼び出しパラメーターとともに入力された形式制御文字はどれも取り外します。

例

図135 は、FLGOpen API 呼び出しを出すために必要な C 言語コードを示しています。このサンプル・コードは、REPORT1 というオブジェクト・インスタンスにより提供された呼び出しパラメーターを使用して、 PRINTRPT というプログラムを開始します。

```
.
 APIRET     rc;               // reason code from FLGOpen
 UCHAR      pszPgmFLGID[FLG_ID_LEN + 1];
 UCHAR      pszObjFLGID[FLG_ID_LEN + 1];
 FLGEXTCODE     ExtCode = 0;                    // Extended code
   .
   .  /* set pszPgmFLGID information Catalog Manager-id of 'PRINTRPT'              */
   .  /* set pszObjFLGID to information Catalog Manager-id of 'REPORT1'            */
   .
 rc = FLGOpen    (pszPgmFLGID,
                  pszObjFLGID,
                  &ExtCode);
```

図 135. C 言語による FLGOpen の呼び出しのサンプル

FLGRelation

2 つのオブジェクト・インスタンス間での以下の関係を作成または削除するために使用します。

- アタッチメント
- 包含
- 連絡先
- リンク

許可

管理者または許可ユーザー (すべての関係); ユーザー (アタッチメント関係のみ)

構文

```
APIRET  APIENTRY   FLGRelation( PSZ              pszSrcFLGID,
                                PSZ              pszTrgFLGID,
                                FLGRELTYPE       RelType,
                                FLGRELOPTION     RelOpt,
                                  PFLGEXTCODE  pExtCode );
```

パラメーター

pszSrcFLGID (PSZ) – 入力

ソース・オブジェクト・インスタンスの 16 文字のシステム生成された固有の識別子を指します。

この ID の 1 文字目から 6 文字目までは、このインスタンスのオブジェクト・タイプを識別します。

この ID の 7 文字目から 16 文字目までは、システム生成された固有のインスタンス ID です。

指定する FLGID は、作成または削除したい関係のタイプにより、異なります。

アタッチメント関係

連絡先が付加または切り離される、Attachment 区分以外のオブジェクト・インスタンスの FLGID

連絡先関係

連絡先が、定義または除去される Elemental または Grouping 区分のオブジェクト・インスタンスの FLGID

包含関係

Grouping 区分の含む側のオブジェクト・インスタンスの FLGID

リンク関係

ほかのオブジェクト・インスタンスとの対等関係が作成、または削除される、Elemental または Grouping 区分のオブジェクト・インスタンスの FLGID

pszTrgFLGID (PSZ) － 入力

ターゲット・オブジェクトを表すシステム生成された固有の 16 文字 ID を指します。 6 文字のオブジェクト・タイプ ID と 10 文字のインスタンス ID からなります。指定する FLGID は、作成または削除したい関係のタイプにより、異なります。

アタッチメント関係

付加または切り離される Attachment 区分のオブジェクト・インスタンスの FLGID

連絡先関係

定義または除去される Contact 区分のオブジェクト・インスタンスの FLGID

包含関係

Grouping ソースを含む側に追加または除去される Elemental または Grouping 区分のオブジェクト・インスタンスの FLGID

リンク関係

ほかのオブジェクト・インスタンスとの対等関係が作成、または削除される、Elemental または Grouping 区分のオブジェクト・インスタンスの FLGID

RelType (FLGRELTYPE) － 入力

作成または削除される関係のタイプを識別します。有効な値は以下のとおりです。

A アタッチメント
C 包含
L リンク
T 連絡先

RelOpt (FLGRELOPTION) － 入力

実行するアクションを指定します。有効な値は以下のとおりです。

C 関係の作成
D 関係の削除

pExtCode (PFLGEXTCODE) – 出力

理由コードに関連する拡張コードを指します。戻された理由コードに関連した意味のある拡張コードがあるかどうかについては、285ページの『付録D. 情報カタログ・マネージャーの理由コード』を参照してください。

理由コード (APIRET)

この API 呼び出しの実行結果を表します。

戻された理由コードの説明については、285ページの『付録D. 情報カタログ・マネージャーの理由コード』を参照してください。

使用方法

前提条件

オブジェクト・インスタンスを削除する前に、そのオブジェクト・インスタンスが別のオブジェクト・インスタンスを包含する関係をすべて削除しなければなりません。

情報カタログ更新の制御

プログラムを可能な限り、情報カタログと同期化させるために、FLGRelation が正常に完了したあとで、呼び出しを FLGCommit (82ページの『FLGCommit』参照) に組み込む必要があります。 FLGRelation が正常に完了しない場合は、呼び出しを FLGRollback (225ページの『FLGRollback』参照) に組み込む必要があります。

例

224ページの図136 は、あるオブジェクト・インスタンスによって別のオブジェクトが包含されることを定義する関係を作成するための、 FLGRelation API 呼び出しを出すために必要な C 言語コードを示しています。このサンプル・コードで、MYBUSGP はビジネス・グループ・オブジェクト・タイプ (Grouping オブジェクト) のインスタンスであり、IMAGE-A はイメージ・オブジェクト・タイプ (Elemental オブジェクト) のインスタンスです。

```
APIRET         rc;                    // Declare reason code
UCHAR  pszSrcFLGID[FLG_ID_LEN + 1];
UCHAR  pszTrgFLGID[FLG_ID_LEN + 1];
FLGRELTYPE  RelType=FLG_CONTAINER_RELATION;
FLGRELOPTION  RelOpt=FLG_CREATE_RELATION;
FLGEXTCODE     ExtCode=0;                // Declare extended code
  .
  .  /* set values for pszSrcFLGID and pszTrgFLGID      */
  .
rc = FLGRelation (pszSrcFLGID,
                  pszTrgFLGID,
                  RelType,
                  RelOpt,
                  &ExtCode);
```

図 136. C 言語による FLGRelation の呼び出しのサンプル

FLGRollback

最後のコミット点またはロールバック以降に情報カタログに対して行われたすべての変更を削除するために使用します。

許可

管理者およびユーザー

構文

```
APIRET APIENTRY FLGRollback (PFLGEXTCODE   pExtCode)
```

パラメーター

pExtCode (PFLGEXTCODE) － 出力

理由コードに関連する拡張コードを指します。戻された理由コードに関連した意味のある拡張コードがあるかどうかについては、285ページの『付録D. 情報カタログ・マネージャーの理由コード』を参照してください。

理由コード (APIRET)

この API 呼び出しの実行結果を表します。

戻された理由コードの説明については、285ページの『付録D. 情報カタログ・マネージャーの理由コード』を参照してください。

使用方法

プログラムでエラーが発生したために情報カタログに矛盾が生じる可能性がある場合には、 FLGRollback を出してください。

例

226ページの図137 には、FLGRollback API 呼び出しを発行するコードが示されています。

```
APIRET        rc;           // Declare reason code from FLGRollback
FLGEXTCODE     ExtCode = 0;               // Declare extended code
  .
  .
rc = FLGRollback(&ExtCode);      // pass the address of
                             // extended code
```

図 137. FLGRollback API 呼び出しを出す C コードのサンプル

FLGSearch

ユーザー定義の探索基準にもとづいて情報カタログを探索し、特定のオブジェクト・タイプのインスタンスを突き止めるために使用します。

許可

管理者またはユーザー

構文

```
APIRET  APIENTRY    FLGSearch( PSZ              pszObjTypeID,
                               PFLGHEADERAREA   pSelCriteriaStruct,
                                 PFLGHEADERAREA * ppListStruct,
                                  PFLGEXTCODE  pExtCode );
```

パラメーター

pszObjTypeID (PSZ) – 入力
探索したい情報カタログ・マネージャー・オブジェクト・タイプの 6 文字の ID を指定します。

pSelCriteriaStruct (PFLGHEADERAREA) – 入力
探索基準に関する特性の仕様と値が入っている入力構造を指します。

この値が NULL の場合、情報カタログ・マネージャーは指定されたオブジェクト・タイプのすべてのインスタンスを戻します。

ppListStruct (PFLGHEADERAREA) – 出力
検索によって選択されたオブジェクト・インスタンスのリストが入っている出力構造を指すポインターのアドレスを指します。

各インスタンスについて以下の情報が示されます。
- FLGID (16 文字)
- 名前 (80 文字)

すべてのインスタンスは、使用するデータベース管理システムの照合順序に従って、80 文字の外部名 (名前の値) の昇順で分類されます。

FLGSearch によって戻すことのできるオブジェクト・インスタンスの最大数は計算機で使用可能な記憶域によって異なり、約 5000 です。

出力構造がない場合、出力構造へのポインターは NULL にセットされます。

pExtCode (PFLGEXTCODE) − 出力

理由コードに関連する拡張コードを指します。戻された理由コードに関連した意味のある拡張コードがあるかどうかについては、285ページの『付録D. 情報カタログ・マネージャーの理由コード』を参照してください。

理由コード (APIRET)

この API 呼び出しの実行結果を表します。

戻された理由コードの説明については、285ページの『付録D. 情報カタログ・マネージャーの理由コード』を参照してください。

入力構造

FLGSearch を使用するには、図138 に示す入力構造を定義しなければなりません。この構造はヘッダー域と定義域だけからなります。

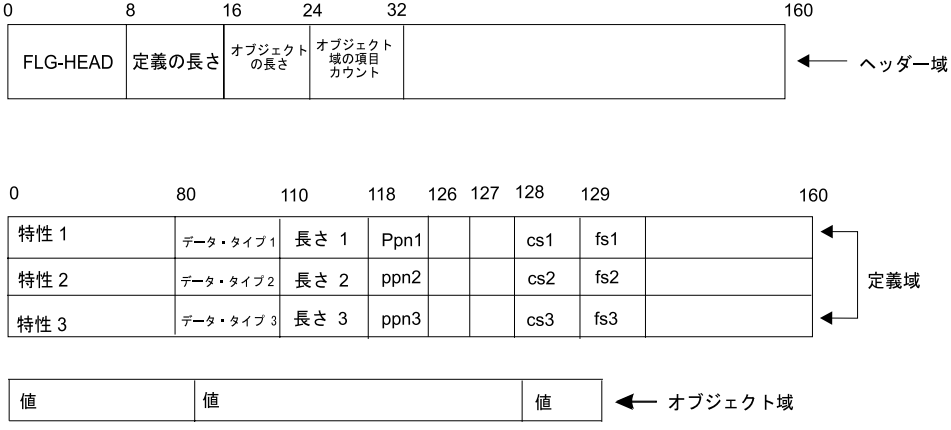


図 138. FLGSearch 入力構造

FLGSearch 入力構造の定義域は、図138 に示すとおりに指定しなければなりません。ただし、オブジェクト・タイプに関して定義された特性はどれでも指定することができます。定義域に指定されたそれぞれの特性に対応する探索基準値を、オブジェクト域に指定しなければなりません。バイト・オフセットの意味については、56ページの『情報カタログ・マネージャーの API 出力構造』を参照してください。

データベースが DB2 UDB (OS/390 版) である場合、探索基準の最大長は 254 です。

出力構造

FLGSearch が作成する出力構造には、図139 に示すように、探索基準を使用して検索されたオブジェクトのリストが入っています。

出力構造のオブジェクト域には、入力探索基準に一致したすべてのオブジェクト・インスタンスのリストが入ります。戻されたオブジェクト・インスタンスは、FLGID の値とオブジェクト・インスタンスの外部名によって識別されます。

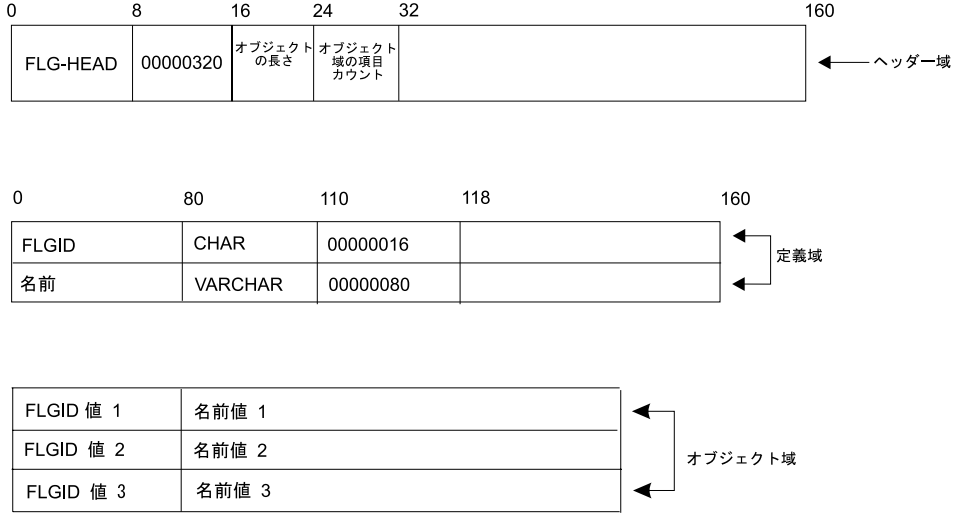


図 139. FLGSearch 出力構造

使用方法

FLGSearch は 1 つのオブジェクト・タイプのインスタンスだけを探索します。すべてのオブジェクト・タイプのインスタンスを探索するには、FLGSearchAll API 呼び出しを使用してください。

すべてのオブジェクト・タイプではなく、そのうちの複数のオブジェクト・タイプを探索するには、探索したいオブジェクト・タイプごとに 1 回ずつ FLGSearch を呼び出してください。

入力構造には、探索基準の特性の仕様と値が入ります。

– 探索基準特性としては、任意のオブジェクトの特性を指定することができます。

– 複数の特性を指定すると、それらの特性が AND 演算子に連係されて探索基準が作られます。
– nonCHAR データ・タイプ上の後書きのブランク以外のブランクは、探索基準の一部と見なされます。
– 探索基準にはワイルド・カード 文字を含めることができます。ワイルド・カード文字を使用すると、特定の特性の値の中から探し出したいパターンを指定することができます。このデータベースでは、次の 2 種類のワイルド・カード文字がサポートされます。

  % 0 個またはそれ以上の文字を表します。

  _ 1 つの文字を表します。

  ユーザー・インターフェースではこれと異なるワイルド・カード文字を使用することもできますが、 FLGSearch で使用できるのは % 文字と _ 文字だけです。

  DB2 データベースは、後書きのブランクを有効なものとして扱うので、探索基準または CHAR タイプ特性の最後にワイルドカードを組み込まなければなりません。そうしない場合、FLGSearch への呼び出しから受け取るオブジェクトが予想よりも少なくなります。

  探索基準にワイルド・カード文字を含める場合には、ファジー探索フラグ (fs) を Y にセットしなければなりません。
– 定義域で以下のフラグの値を指定する必要があります。

  cs バイト 128 の大文字小文字の区別フラグ。有効な値は、大文字小文字を区別する場合には Y、区別しない場合には N です。

  情報カタログが、DB2 UDB (OS/390 版) にある場合、
  - すべて大文字の値 (デフォルト) で作成され、大文字小文字の区別フラグは N でなければなりません。
  - 大文字小文字混合文字の値で作成され、大文字小文字の区別フラグは Y でなければなりません。

  fs バイト 129 のファジー探索フラグ。有効な値は、ファジー探索を行う場合には Y、行わない場合には N です。探索基準にワイルド・カード (%または_) が含まれている場合には、この値は Y でなければなりません。

情報カタログ更新の制御

FLGSearch はデータベースに対して変更をコミットします。 FLGSearch 呼び出しの前に行われた予期しない変更が情報カタログ・マネージャーによっ

てコミットされないようにするために、プログラムの中で、FLGSearch を出す前に FLGCommit または FLGRollback を出すようにしてください。

出力構造に割り振られたメモリーの解放

FLGSearch が出力構造にあるデータを戻した場合、出力構造に戻されたデータを保管してから、FLGFreeMem を呼び出さなくてはなりません (137ページの『FLGFreeMem』参照)。メモリーを解放するのに、たとえば C 言語指示のような他の方式は使用しないでください。

例

FLGSearch: 例 1

図140 のサンプル・コードは、用語集インスタンスの探索を行います。 232ページの図141 に示すように、定義域のバイト 129 でファジー探索フラグが N にセットされているため、この探索は*絶対探索* です。ただし、バイト 128 の大文字小文字の区別フラグは N にセットされているため、値で使用される文字の大文字と小文字の区別は意味を持ちません。 FLGListObjTypes または FLGGetType 呼び出しを使って、オブジェクト・タイプ識別子を見つけておく必要があります。

```
APIRET          rc;             // reason code from FLGSearch
UCHAR           pszObjTypeID[FLG_OBJTYPID_LEN + 1];
PFLGHEADERAREA   pSelCriteria;         // search criterion input structure pointer
PFLGHEADERAREA  * ppListStruct;  // pointer to search result pointer
FLGEXTCODE     ExtCode = 0;                // Declare extended code
  .
  .  /* provide values for input parameters */
  .
strcpy (pszObjTypeID, "000006");
rc = FLGSearch (pszObjTypeID,     // information Catalog Manager object type ID
                pSelCriteria,        // input structure pointer
                ppListStruct, // pass the address of
                                // output structure pointer
                &ExtCode);
```

図 140. C 言語による FLGSearch の呼び出しのサンプル

232ページの図141 は、この探索に関する特性と値の情報が入る探索条件入力構造 (pSelCriteria によって指し示されます) を示しています。

定義域におけるバイト 128 の大文字小文字の区別フラグとバイト 129 のファジー探索フラグは N にセットしなければなりません。これは、ユーザーが絶対探索を希望していて、しかも特性値の大文字と小文字を区別しようとしていないためです。

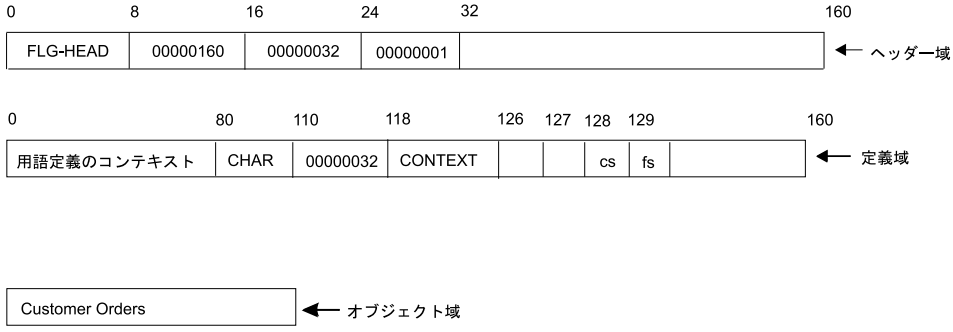


図 141. FLGSearch のための入力構造のサンプル

図142 は、探索によって得られた用語集インスタンスが入った出力構造を示しています (ppListStruct は、この出力構造を指すポインターのアドレスを指します)。

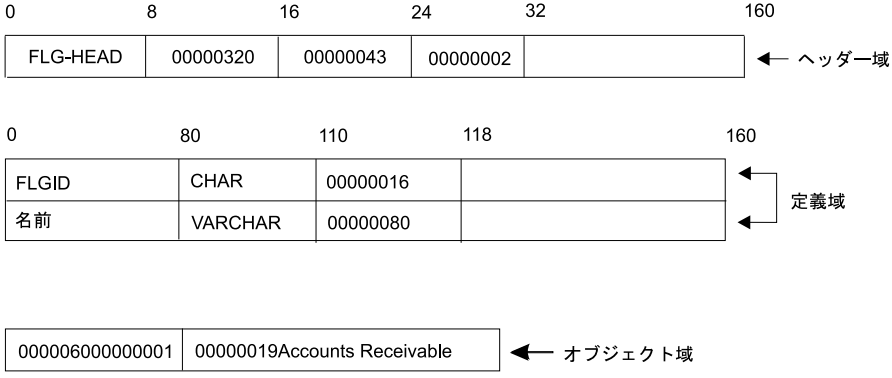


図 142. FLGSearch のための出力構造のサンプル

CONTEXT の値 `Customer Orders` が探索基準として使用されます。CONTEXT がこの値になっている用語集インスタンスが出力構造に戻されます。大文字小文字の区別フラグが `N` にセットされているため、 CONTEXT 値が `customer orders` や `CUSTOMER ORDERS` などのようになっているインスタンスも戻されます。

FLGSearch: 例 2

この例は、あるパターンに一致する値を含むインスタンスを見つけるために、ユーザーのプログラムでファジー探索を使用する方法を示しています。

233ページの図143 の入力構造に指定された値では、`metadata` という文字を含む用語集インスタンスをワイルド・カード探索することが指定されています。

複数文字ワイルド・カード (%) は、その値に任意の別の文字が含まれていても探索基準が満たされることを示しています。

図143 は、ユーザーのプログラムが FLGSearch に渡す入力構造 (pSelCriteria によって指し示されます) を示しています。

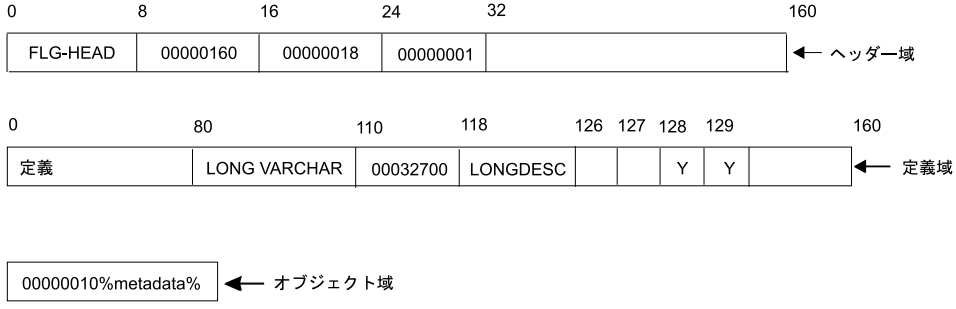


図 143. FLGSearch のための入力構造のサンプル

この探索はワイルド・カード探索であるため、バイト 129 のファジー探索フラグは Y にセットしなければなりません。ファジー探索フラグが N にセットされていると、% 文字は探索基準のリテラル部分になります。つまり指定された特性値に % が含まれているインスタンスだけが戻されます。

この例ではメタデータの大文字と小文字の区別が意味を持つため、定義域のバイト 128 にある大文字小文字の区別フラグは Y にセットされています。 234 ページの図144 は、探索によって得られた用語集インスタンスが入った出力構造を示しています (ppListStruct は、この出力構造を指すポインターのアドレスを指します)。

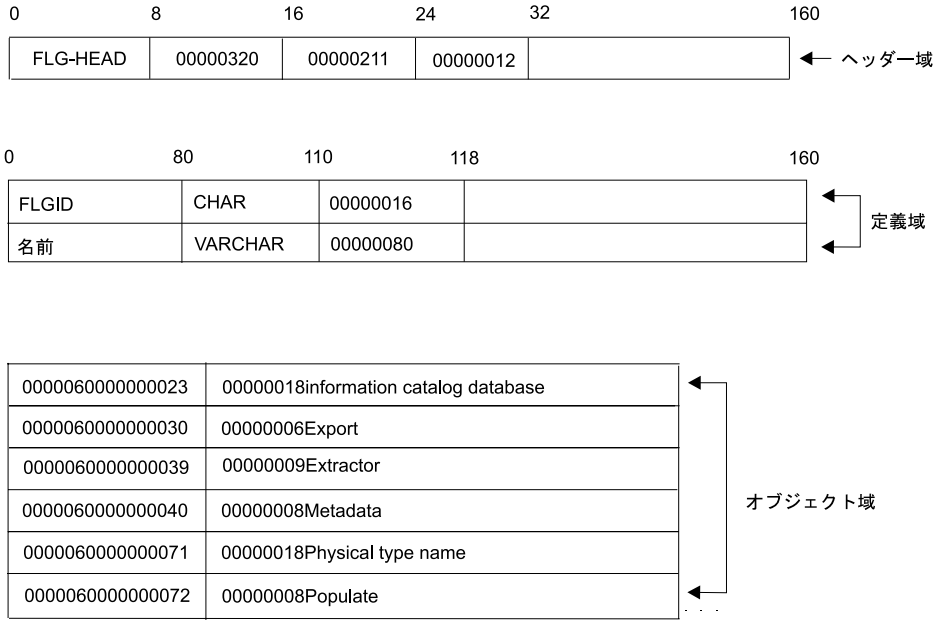


図 144. FLGSearch のための出力構造のサンプル

定義特性の値 %metadata% が探索基準として使用されます。定義特性値に metadata が含まれている用語集インスタンスが出力構造に戻されます。大文字小文字の区別フラグが Y にセットされているため、この例で検出されたすべてのインスタンスは、metadata の大文字と小文字の区別にも一致します。

FLGSearch: 例 3

この例は、あるパターンに一致する値を含むインスタンスを見つけるために、ユーザーのプログラムでファジー探索を使用する方法を示しています。

235ページの図145 の入力構造で指定された値では、指定した特性値のほかに 1 つだけの可変文字を含む用語集インスタンスを探索するために、単一文字ワイルド・カード (_) が使用されています。

235ページの図145 は、ユーザーのプログラムが FLGSearch に渡す入力構造 (pSelCriteria によって指し示されます) を示しています。

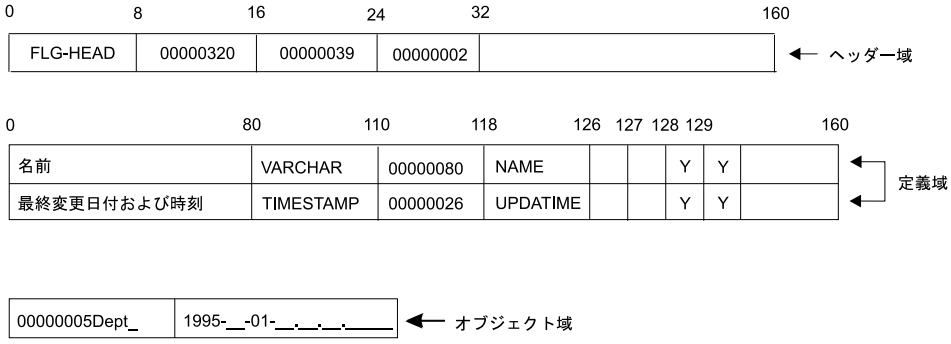


図 145. FLGSearch のための入力構造のサンプル

検索基準に単一文字ワイルドカード (_) が含まれているため、 129 バイトのファジー検索フラグを Y にセットする必要があります。ファジー探索フラグが N にセットされていると、情報カタログ・マネージャーは _ を探索基準のリテラル部分と見なし、指定された特性値の一部として _ が含まれているオブジェクト・インスタンスだけを戻します。

この例では、NAME および UPDATIME の両方の値が探索基準として使用されます。

- 指定された NAME 値 `Dept_` は、インスタンスの探索が Dept で始まり、未知の 1 文字で終わることを意味します。この値には 5 つの文字が含まれます。
- タイム・スタンプ・データ・タイプ特性 UPDATIME には、年と日付の値が指定されています。 `1995` 年および `01` 日の UPDATIME 値は、AND 演算子によって NAME 値と連係され、あるオブジェクト・インスタンスが戻されるかどうかを判別するための探索基準を構成します。情報カタログ・マネージャーは、UPDATIME 値と NAME 値の両方が探索基準を満たさなければオブジェクト・インスタンスを戻しません。

236ページの図146 は、探索によって得られた用語集インスタンスが入った出力構造を示しています (ppListStruct は、この出力構造を指すポインターのアドレスを指します) 。

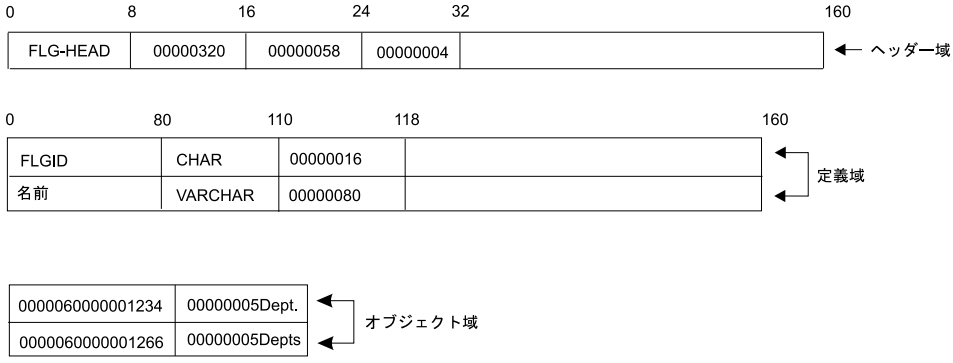


図 146. FLGSearch のための出力構造のサンプル

5 文字の名前値の接頭部が `Dept` になっていて、1995 年の各月の最初の日に更新された用語集インスタンスが出力構造に戻されます。

FLGSearchAll

情報カタログのすべてのオブジェクト・タイプを探索し、探索基準に一致するインスタンス名 (名前特性) のインスタンスを探し出すために使用します。

許可

管理者またはユーザー

構文

```
APIRET  APIENTRY   FLGSearchAll( PFLGHEADERAREA   pSelCriteriaStruct,
                                 PFLGHEADERAREA * ppListStruct,
                                  PFLGEXTCODE  pExtCode );
```

パラメーター

pSelCriteriaStruct (PFLGHEADERAREA) – 入力

入力構造を指します。

この構造には、探索基準の特性の仕様と値が含まれます。 FLGSearchAll の探索基準としては、オブジェクト・インスタンスの外部名 (Name) の値だけを使用することができます。

pSelCriteriaStruct が NULL にセットされていると、情報カタログ・マネージャーは、最大約 5000 までの、情報カタログ内のすべてのインスタンスを戻します。

ppListStruct (PFLGHEADERAREA) – 出力

検索によって選択されたオブジェクト・インスタンスのリストが入っている出力構造を指すポインターのアドレスを指します。出力構造がない場合、出力構造へのポインターは NULL にセットされます。各インスタンスについて以下の情報が示されます。

- FLGID (16 文字)
- 名前 (80 文字)

すべてのインスタンスは、使用するデータベース管理システムの照合順序の昇順に従って、最初にオブジェクト・タイプ名によって分類され、次にインスタンスの外部名 (名前の値) によって分類されます。

FLGSearchAll によって戻すことのできるオブジェクト・インスタンスの最大数は計算機で使用可能な記憶域によって異なり、約 5000 です。

pExtCode (PFLGEXTCODE) − 出力

理由コードに関連する拡張コードを指します。戻された理由コードに関連した意味のある拡張コードがあるかどうかについては、285ページの『付録D. 情報カタログ・マネージャーの理由コード』を参照してください。

理由コード (APIRET)

この API 呼び出しの実行結果を表します。

戻された理由コードの説明については、285ページの『付録D. 情報カタログ・マネージャーの理由コード』を参照してください。

入力構造

FLGSearchAll の探索基準を指定するには、以下の入力構造を定義する必要があります。この構造は、ヘッダー域、名前特性のみを含むことのできる定義域、およびオブジェクト域からなります。

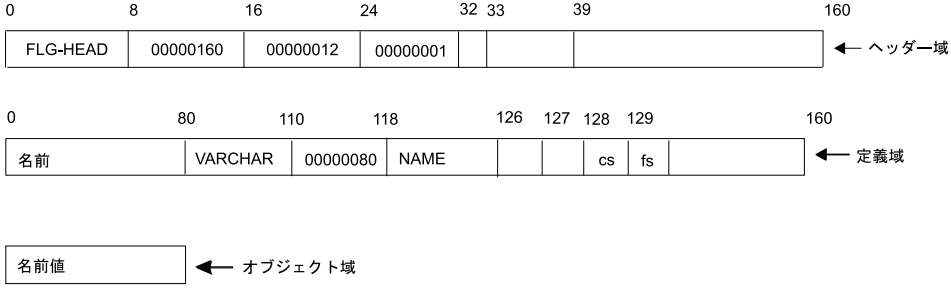


図 147. FLGSearchAll 入力構造

FLGSearchAll 入力構造の定義域は、図147 に示すとおりに指定しなければなりません。バイト・オフセットの意味については、36ページの『情報カタログ・マネージャー API 入力構造』を参照してください。

出力構造

FLGSearchAll が作成する出力構造には、239ページの図148 に示すように、探索基準を使用して検索されたオブジェクトのリストが入っています。

出力構造のオブジェクト域には、入力探索基準に一致したすべてのオブジェクト・インスタンスのリストが入ります。戻されたオブジェクト・インスタンスは、FLGID の値とオブジェクト・インスタンスの外部名によって識別されます。

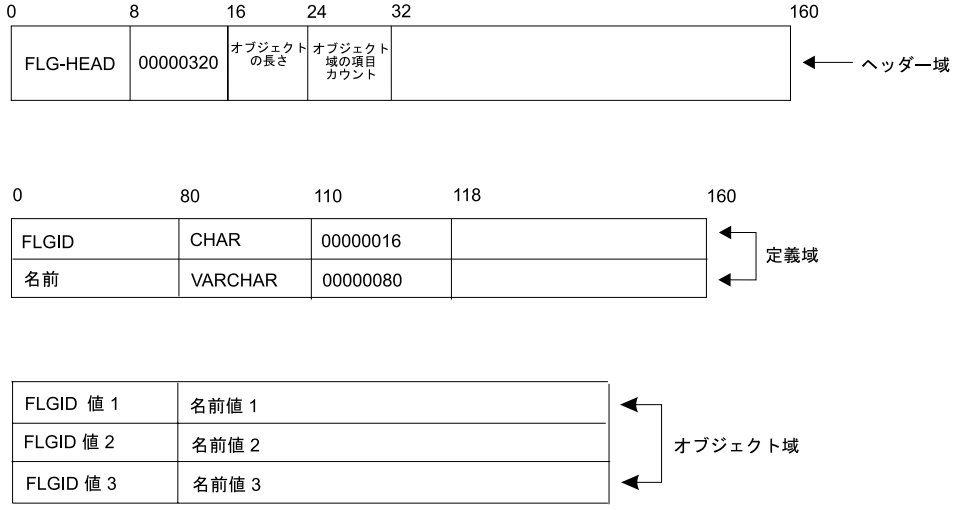


図 148. FLGSearchAll 出力構造

使用方法

探索基準としては、オブジェクト・インスタンスの外部名 (名前) の値だけを使用することができます。それ以外の特性値は、FLGSearchAll では使用することができません。探索基準に別の特性の値を使用したい場合には、FLGSearch を使用してください (227ページの『FLGSearch』を参照)。

探索基準には*ワイルド・カード* 文字を含めることができます。ワイルド・カード文字を使用すると、特定の特性の値の中から探し出したいパターンを指定することができます。このデータベースでは、次の 2 種類のワイルド・カード文字がサポートされます。

% 0 個またはそれ以上の文字を表します。

_ 1 つの文字を表します。

ユーザー・インターフェースではこれと異なるワイルド・カード文字を使用することもできますが、 FLGSearchAll で使用できるのは % 文字と _ 文字だけです。

探索基準にワイルド・カード文字を含める場合には、ファジー探索フラグ (fs) を Y にセットしなければなりません。

定義域で以下のフラグの値を指定する必要があります。

cs バイト 128 の大文字小文字の区別フラグ。有効な値は、大文字小文字を区別する場合には Y、区別しない場合には N です。

情報カタログが、DB2 UDB (OS/390 版) にある場合、
- すべて大文字の値 (デフォルト) で作成され、大文字小文字の区別フラグは N でなければなりません。
- 大文字小文字混合文字の値で作成され、大文字小文字の区別フラグは Y でなければなりません。

fs バイト 129 のファジー探索フラグ。有効な値は、ファジー探索を行う場合には Y、行わない場合には N です。ワイルド・カード (% または _) を探索基準に含めて探索を行いたい場合には、この値を Y にしなければなりません。

情報カタログ更新の制御

FLGSearchAll はデータベースに対して変更をコミットします。
FLGSearchAll 呼び出しの前に行われた予期しない変更が情報カタログ・マネージャーによってコミットされないようにするために、プログラムの中で、FLGSearchAll を出す前に FLGCommit または FLGRollback を出すようにしてください。

出力構造に割り振られたメモリーの解放

FLGSearchAll が出力構造にあるデータを戻した場合、出力構造に戻されたデータを保管してから、FLGFreeMem を呼び出さなくてはなりません (137 ページの『FLGFreeMem』参照)。メモリーを解放するのに、たとえば C 言語指示のような他の方式は使用しないでください。

例

図149 は、FLGSearchAll 呼び出しを出すために必要な C 言語コードを示しています。このサンプル・コードは、すべてのオブジェクト・タイプ・インスタンスから特定の名前を探索します。

```
APIRET           rc;                // reason code from FLGSearchAll
PFLGHEADERAREA   pSelCriteria;         // search criterion input structure pointer
PFLGHEADERAREA  * ppListStruct;  // pointer to search result pointer
FLGEXTCODE     ExtCode = 0;               // Declare extended code
  .
  .  /* provide values for input parameters */
  .
rc = FLGSearchAll ( pSelCriteria,        // input structure pointer
                ppListStruct, // pass the address of
                                // output structure pointer
                &ExtCode);
```

図 149. C 言語による FLGSearchAll の呼び出しのサンプル

図150 は、この探索に関する特性と値の情報が入る探索条件入力構造 (pSelCriteria によって指し示されます) を示しています。

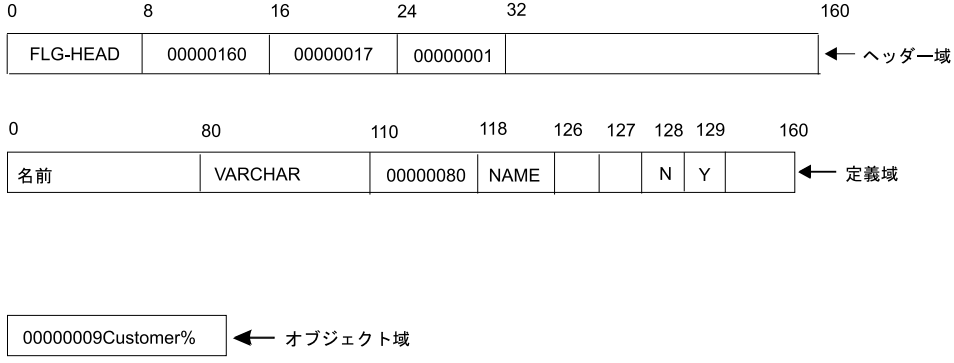


図 150. FLGSearchAll のための入力構造のサンプル

この例では、探索基準にワイルドカード文字を使用してファジー探索を行うため、定義域のバイト 129 のファジー探索フラグは Y にセットされています。

探索基準の文字との厳密な一致は必要でないため、定義域のバイト 128 の大文字小文字の区別フラグは N にセットされています。 242ページの図151 は、探索によって得られた情報カタログ・マネージャー・オブジェクトが入った出力構造を示しています (ppListStruct は、この出力構造を指すポインターのアドレスを指します)。

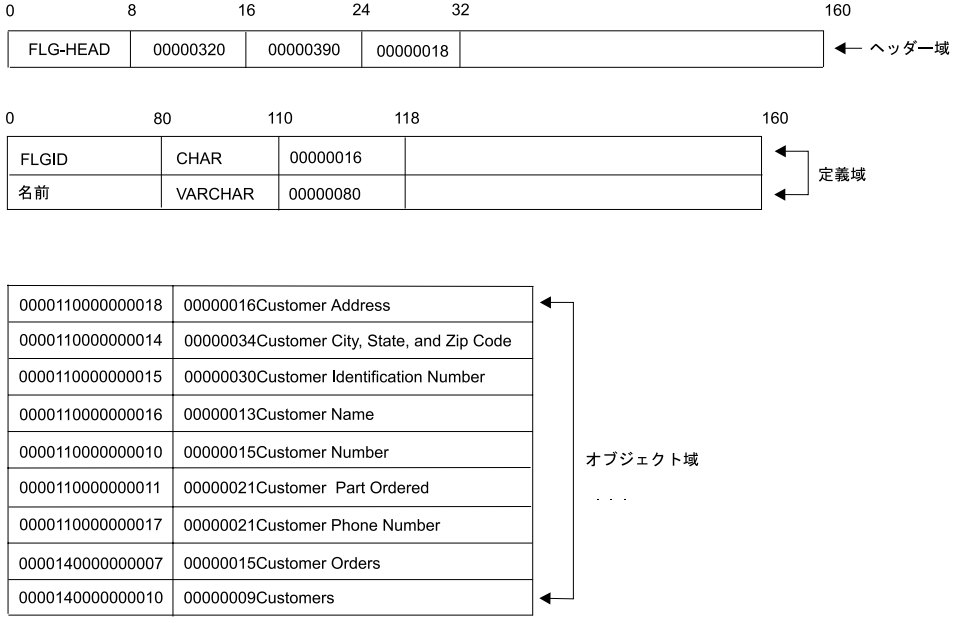


図 151. FLGSearchAll のための出力構造のサンプル

指定された部分オブジェクト・インスタンスは、2 つの異なるオブジェクト・タイプの 9 つのインスタンスによって使用されています。

FLGTerm

情報カタログ・マネージャー API DLL 環境を終了させ、データベース・マネージャーから切断し、関連するすべてのシステム資源を解放するために使用します。

許可

管理者またはユーザー

構文

```
APIRET APIENTRY FLGTerm (PFLGEXTCODE   pExtCode )
```

パラメーター

pExtCode (PFLGEXTCODE) – 出力
理由コードに関連する拡張コードを指します。戻された理由コードに関連した意味のある拡張コードがあるかどうかについては、285ページの『付録D. 情報カタログ・マネージャーの理由コード』を参照してください。

理由コード (APIRET)
この API 呼び出しの実行結果を表します。

戻された理由コードの説明については、285ページの『付録D. 情報カタログ・マネージャーの理由コード』を参照してください。

使用方法

ユーザーのプログラムが FLGTerm を呼び出すと、情報カタログ・マネージャーは、情報カタログ・データベースに対して行われたコミットされていない変更を自動的にコミットします。変更事項をロールバックする必要がある場合には、FLGTerm を呼び出して情報カタログ・マネージャーを終了する前に、プログラムで FLGRollback を呼び出してください。

情報カタログ・マネージャーがデータベースをロール・バックしようとしたときに重大なエラーが発生した場合、FLGTerm が情報カタログ・マネージャーを遮断して資源を解放しようとするとエラーが発生します。 FLGTerm 呼び出しが失敗したときに、ユーザーのプログラムを使用している人が管理者としてログオンしている場合、その人は、管理者ユーザー ID をログオフするために情報カタログ・マネージャー CLEARKA ユーティリティーを使用する必要があります。

例

図152 は、FLGTerm API 呼び出しを出すために必要な C 言語コードを示しています。このサンプル・コードは、情報カタログ・マネージャー API DLL を停止させます。

```
.
 APIRET         rc;               // Reason code
 FLGEXTCODE     ExtCode = 0;                   // Extended code
 .
 .  // FLGInit()
 .  // calls to the FLG API
 rc = FLGTerm ( &ExtCode );
```

図 152. C 言語による FLGTerm の呼び出しのサンプル

FLGTrace

トレース (.TRC) ファイルに書き込まれる情報カタログ・マネージャー機能に関する情報のレベルを設定するために使用します。

許可

管理者またはユーザー

構文

```
APIRET  APIENTRY   FLGTrace( FLGTRACEOPTION   TraceOpt,
                             PFLGEXTCODE  pExtCode );
```

パラメーター

TraceOpt (FLGTRACEOPTION) – 入力
希望するトレース・オプションを示します。有効なオプションは以下のとおりです。

- **0** デフォルトです。すべてのメッセージと警告、エラー、および重大エラー状態が含まれます。
- **1** 最高レベルの情報カタログ・マネージャー機能の入り口および出口の記録が含まれます。
- **2** 情報カタログ・マネージャー機能のきわめて詳細な入り口および出口の記録が含まれます。
- **3** 入出力パラメーターが入ります (入出力構造を除きます)。
- **4** 情報カタログ・マネージャーに渡されて使用されるすべての入出力構造が入ります。これには、使用するデータベース管理システムに渡されて使用される SQLCA 情報も含まれます。

これらの値の定数は、情報カタログ・マネージャー API ヘッダー・ファイル DG2API.H で定義されています。

pExtCode (PFLGEXTCODE) – 出力
理由コードに関連する拡張コードを指します。戻された理由コードに関連した意味のある拡張コードがあるかどうかについては、285ページの『付録 D. 情報カタログ・マネージャーの理由コード』を参照してください。

理由コード (APIRET)
この API 呼び出しの実行結果を表します。

戻された理由コードの説明については、285ページの『付録D. 情報カタログ・マネージャーの理由コード』を参照してください。

使用方法

トレース・ファイルの名前は、使用している情報カタログの名前にエクステンション .TRC を付けたものです。

プログラムをデバッグするためにトレース・ファイルを使用する場合には、おそらくレベル 0 および 4 がもっとも便利です。

レベル 0

情報カタログ・マネージャーが実行している機能を説明する情報を戻します。

情報カタログ・マネージャーは、エラーを検出すると、そのファイルに関する理由コードと拡張コードを新規理由コードおよび新規拡張コードとしてトレース・ファイルに挿入します。トレース・ファイルには、旧理由コードと旧拡張コードも含まれています。ここには、エラーが発生する前に戻された理由コードが入っています。情報カタログ・マネージャーが生成したメッセージもトレース・ファイルに収められます。

レベル 4

レベル 0 の場合と同じ情報のほかに、情報カタログ・マネージャーに関するさらに詳しい機能情報、および情報カタログ・マネージャーとの間で受け渡されるデータ構造に関する情報 (データベースから得られた入力構造、出力構造、および SQLCA 構造を含みます) が入ります。

データ・エラーの原因を判別したり、入出力構造の内容の作成または読み取りが正しく行われていることを確認したりする必要がある場合には、これらの構造の内容をトレースすると便利です。

トレース・ファイルの使い方の詳細については、*情報カタログ・マネージャー 管理の手引き* を参照してください。

例

247ページの図153 は、FLGTrace API 呼び出しを出すために必要な C 言語コードを示しています。このサンプル・コードは、情報アプリケーションから行うトレースのレベルを設定しています。

```
.
 FLGTRACEOPTION TraceOpt   = FLG_TRACELEVEL_1; // Turn on Entry/Exit Tracing
 FLGTRACEOPTION TraceReset = FLG_TRACELEVEL_0; // Reset to default level
 APIRET         rc;                          // reason code
 FLGEXTCODE     ExtCode = 0;                      // Extended code
 .
 .  // FLGInit()
 .  // calls to the FLG API
 rc = FLGTrace ( TraceOpt,
                 &ExtCode );
 .
 .  // Check rc and ExtCode
 .
 .  // More API calls
 .
 rc = FLGTrace ( TraceReset,
                 &ExtCode );
```

図 153. C 言語による FLGTrace の呼び出しのサンプル

FLGUpdateInst

特定のオブジェクト・インスタンスに関する 1 つまたは複数の特性値を変更するために使用します。

許可

管理者または許可ユーザー (すべてのオブジェクト・タイプ); ユーザー (Comments オブジェクト・タイプのみ)

構文

```
APIRET  APIENTRY    FLGUpdateInst( PFLGHEADERAREA   pObjInstStruct,
                                   PFLGEXTCODE  pExtCode );
```

パラメーター

pObjInstStruct (PFLGHEADERAREA) – 入力
更新されるデータベース・オブジェクトに関する特性の仕様と値を含む入力構造を指します。

pExtCode (PFLGEXTCODE) – 出力
理由コードに関連する拡張コードを指します。戻された理由コードに関連した意味のある拡張コードがあるかどうかについては、285ページの『付録 D. 情報カタログ・マネージャーの理由コード』を参照してください。

理由コード (APIRET)
この API 呼び出しの実行結果を表します。

戻された理由コードの説明については、285ページの『付録D. 情報カタログ・マネージャーの理由コード』を参照してください。

入力構造

FLGUpdateInst を使用するには、249ページの図154 に示す入力構造を定義しなければなりません。この構造はヘッダー域、定義域、およびオブジェクト域からなります。

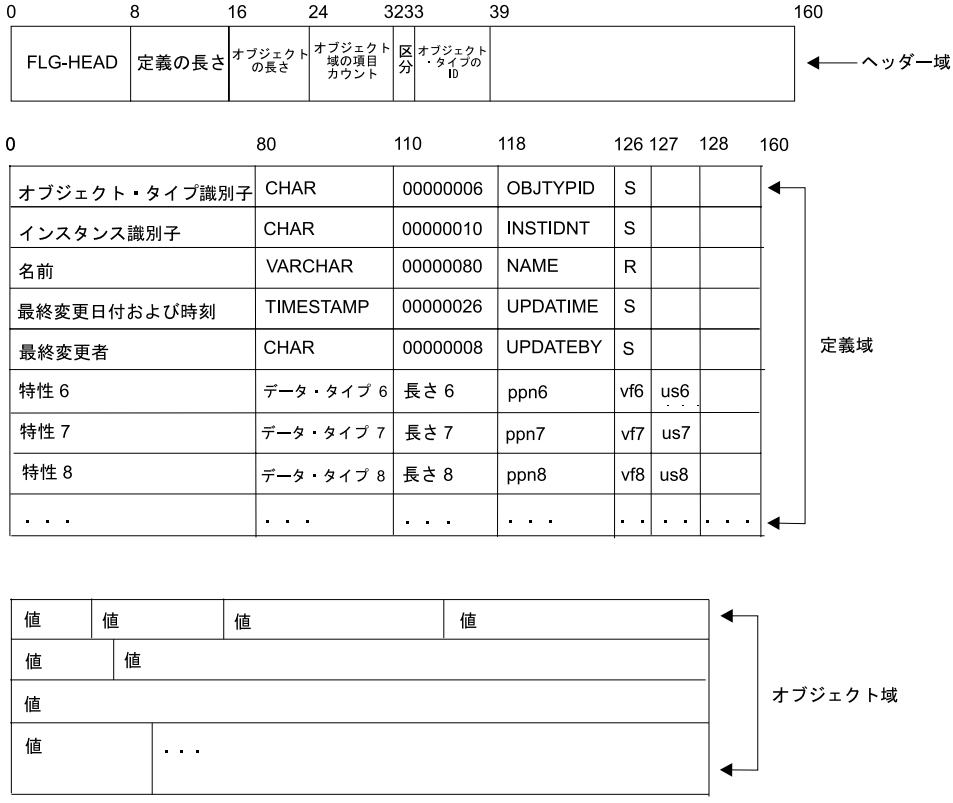


図 154. FLGUpdateInst 入力構造

バイト・オフセットの意味については、36ページの『情報カタログ・マネージャー API 入力構造』を参照してください。

使用方法

前提条件

FLGUpdateInst 呼び出しを出すまえに、FLGCreateInst 呼び出しまたは FLGGetInst 呼び出しを出して、修正するインスタンスに関する特性の仕様と値を得ておく必要があります。

入力要件

ヘッダー域

- ヘッダー・レコードに示されたすべての情報を指定する必要があります。

- ヘッダー・レコードにおけるオブジェクト・タイプ識別子の値 (33 桁目から 38 桁目まで) は、オブジェクト域における値 (オブジェクト域の最初の項目) と同じでなければなりません。

定義域

定義域は、オブジェクト・インスタンスを更新しているオブジェクト・タイプの定義済みの特性のいくつか、またはすべてを含むことができます。以下の規則が適用します。

- まずはじめに、以下の順序で 5 つの情報カタログ・マネージャーの必須特性をすべて指定します。OBJTYPID、INSTIDNT、NAME、UPDATIME、および UPDATEBY。
- すべての UUI 特性を指定します。
- 情報カタログ・マネージャーは、すべての指定された特性の値を、以下の仕様についてオブジェクト・タイプ定義と比較します。

  データ・タイプ

  データ長

  特性省略名

  値フラグ

  UUI 番号

オブジェクト域

- HANDLES 特性に指定するオブジェクト・タイプは情報カタログに存在しているものでなければならず、しかも非 Program オブジェクト・タイプでなければなりません。 PARMLIST 特性に指定する特性は、HANDLES に指定されたオブジェクト・タイプの特性でなければなりません。詳細については、29ページの『プログラムを開始するための Programs オブジェクトの設定』を参照してください。
- 定義域の 126 桁目に R が入って定義されている必須特性に、値が指定されていない場合は、オブジェクト域の該当スペースを次のように初期設定しなければなりません。

データ・タイプ	初期設定値
CHAR	非適用記号のあとに、特性の長さ全体にわたってブランクが続きます。
TIMESTAMP	最大設定可能値に設定。 9999-12-31-24.00.00.000000
VARCHAR LONG	00000001; 8 バイトで指定された長さフィールド。非適用
VARCHAR	記号が続きます。

- OBJTYPID 特性と INSTIDNT 特性の値は、更新するインスタンスを識別するための値であり、したがって必ず指定しなければなりません。
- UPDATIME 特性と UPDATEBY 特性の値はシステム生成の値であるため、ユーザーが修正してはなりません。この FLGUpdateInst 呼び出しを出す前に FLGGetInst 呼び出しを出すと、オブジェクト域にこれら 2 つのシステム生成特性の値を入れることができます。これによってエラーが起こることはありませんが、インスタンスが更新されると、システムはこれら 2 つの特性の値を置き換えます。

VARCHAR または LONG VARCHAR データ・タイプのオブジェクト域からは後続ブランクが自動的に除去され、それに応じて長さフィールドが自動的に調整されます。

情報カタログ更新の制御

プログラムを可能な限り、情報カタログと同期化させるために、FLGUpdateInst が正常に完了したあとで、呼び出しを FLGCommit (82ページの『FLGCommit』参照) に組み込む必要があります。
FLGUpdateInst が正常に完了しない場合は、呼び出しを FLGRollback (225ページの『FLGRollback』参照) に組み込む必要があります。

例

図155 は、FLGUpdateInst API 呼び出しを出すために必要な C 言語コードを示しています。

このサンプル・コードは、FLGCreateInst の例で定義された Quality Group という名前のオブジェクト・インスタンスを更新します。この更新では、短縮記述特性である簡略記述の値が修正されます。

```
APIRET         rc;                    // Declare reason code
PFLGHEADERAREA  pObjInstStruct;        // Pointer to the input structure
FLGEXTCODE     ExtCode = 0;                 // Declare extended code
  .
  .  /* updating pObjInstStruct object Instance by   */
     /*                            providing an updated input structure */
  .
rc = FLGUpdateInst (pObjInstStruct,   // Pointer to updated input structure
                    &ExtCode);             // Pass pointer to extended code
```

図 155. C 言語による FLGUpdateInst の呼び出しのサンプル

図156 は、更新されるインスタンスに関する特性と値の情報が入る入力構造 (この C コードの ″pObjInstStruct″ ポインターによって指し示されています) を示しています。

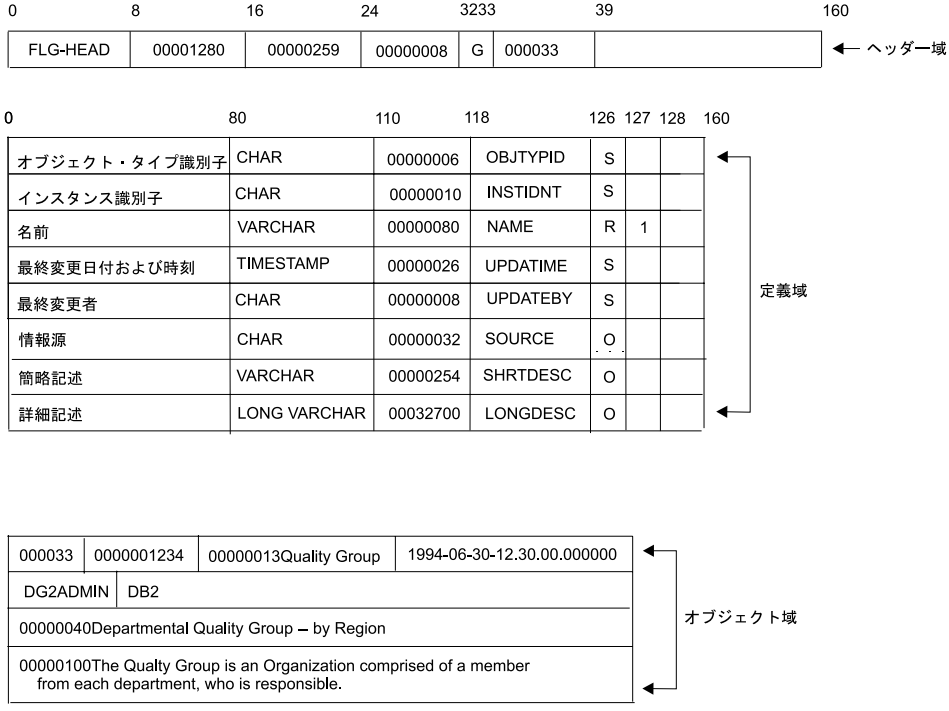


図 156. FLGUpdateInst のための入力構造のサンプル

システム生成されたものでない (すなわち、バイト 126 の値が S になっていない) 以下のオブジェクト域の値は、修正することができます。

- NAME
- SOURCE
- SHRTDESC
- LONGDESC

FLGUpdateInst を使うと、修正していない特性と値を省略することができます。

この例では、簡略記述値が更新されます。簡略記述値を修正すると、その長さも影響を受けます。したがって、オブジェクト域内の簡略記述フィールドの前の 8 文字の長さフィールドは 27 から 40 に修正されます。ヘッダー・レコード内のオブジェクトの長さ値は 246 から 259 に変更されます。

FLGUpdateInst が完了すると、UPDATEBY の値が修正されて、そのインスタンスの更新に使用されたユーザー ID が入り、また UPDATIME には更新のタイム・スタンプが入ります。

FLGUpdateReg

情報カタログ内の特定のオブジェクト・タイプに関する登録情報を修正するために使用します。

このアクションは、オブジェクト・タイプ自体を更新するものではありません。オブジェクト・タイプの登録 情報を更新するものです。

許可

管理者

構文

```
APIRET  APIENTRY   FLGUpdateReg( PFLGHEADERAREA   pObjRegStruct,
                                 PSZ              pszIconFileID,
                                  PFLGEXTCODE  pExtCode );
```

パラメーター

pObjRegStruct (PFLGHEADERAREA) − 入力
更新されるオブジェクト・タイプ登録に関する特性の仕様と値を含む入力構造を指します。

pszIconFileID (PSZ) − 入力
更新されるオブジェクト・タイプ登録済みオブジェクトの OS/2 ICON を含むファイルのドライブ、ディレクトリー・パス、およびファイル名が含まれます。このパラメーターが NULL である場合、ICON は変更されません。このパラメーターを指定すると、現在 ICON が存在してしていない場合にはこの OS/2 ICON がオブジェクト・タイプ登録に追加され、すでに存在している ICON はこの ICON によって置き換えられます。

pExtCode (PFLGEXTCODE) − 出力
理由コードに関連する拡張コードを指します。戻された理由コードに関連した意味のある拡張コードがあるかどうかについては、285ページの『付録 D. 情報カタログ・マネージャーの理由コード』を参照してください。

理由コード (APIRET)
この API 呼び出しの実行結果を表します。

戻された理由コードの説明については、285ページの『付録D. 情報カタログ・マネージャーの理由コード』を参照してください。

入力構造

FLGUpdateReg を使用するには、図157 に示す入力構造を定義しなければなりません。この構造はヘッダー域と定義域だけからなります。

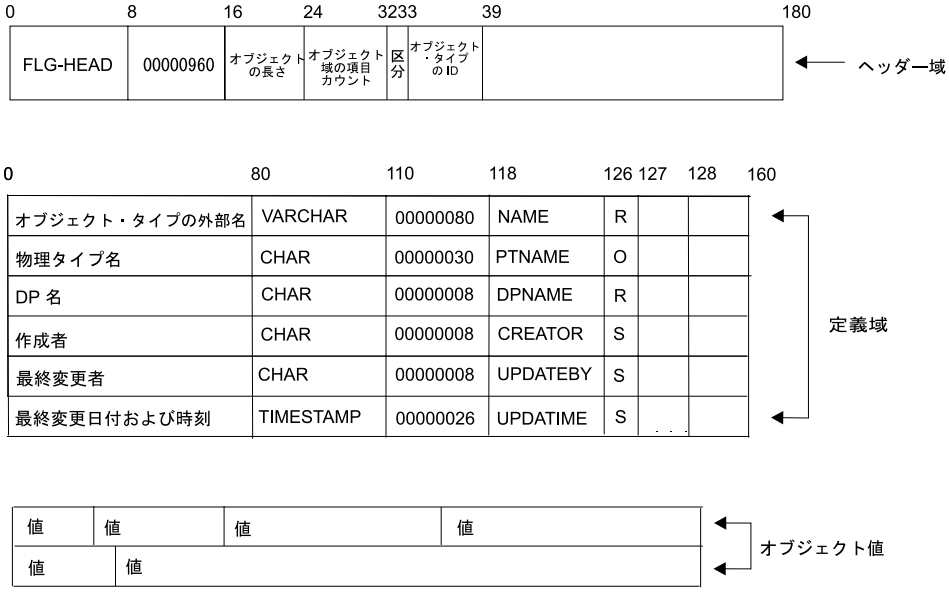


図 157. FLGUpdateReg 入力構造

使用方法

制約事項

- FLGCreateReg によって情報カタログに保管されている登録情報は、オブジェクト・タイプを記述する登録値 (DP名、物理タイプ名、外部名、アイコンなど) からなります。FLGUpdateReg では、外部名とアイコン情報しか更新することができません。
- FLGUpdateReg を使って更新できるのは、OS/2 アイコンだけです。Windows アイコンを更新するには、FLGManageIcons を使用します (198 ページの『FLGManageIcons』参照)。
- FLGCreateReg を使って、オブジェクト・タイプを定義したあとで、FLGUpdateReg または FLGManageIcons 呼び出しを発行し、オブジェクト・タイプに関連したアイコンを変更したり、もともと定義されていないアイコン関連を追加することができます。アイコンをオブジェクト・タイプから除去するのに、FLGManageIcons を使用することもできます。

前提条件

FLGUpdateReg 呼び出しを出す前に、登録情報の現行値を獲得しなければなりません。この情報をもとの FLGCreateReg 呼び出しから保管することも、修正されるオブジェクト・タイプ登録に関する FLGGetReg 呼び出しを出すこともできます。

入力要件

ヘッダー域

ヘッダー・レコードに示されたすべての情報を指定する必要があります。

定義域

– 定義域には、6 つのそれぞれの登録特性に関する定義を含める必要があります。これらの各登録特性の定義は固定されており、特性名以外のすべての仕様は 255ページの図157 に示すとおりになっていなければなりません。特性名も固定されていますが、この例で示した英語の特性名から、サポートされる言語のうちの 1 つに翻訳されていることもあります。

– 特性 (特性省略名で示します) は、以下の順序で定義域およびオブジェクト域に指定しなければなりません。

1. NAME
2. PTNAME
3. DPNAME
4. CREATOR
5. UPDATEBY
6. UPDATIME

これらの特性については、93ページの『FLGCreateReg』に説明されています。

オブジェクト域

更新することができるのは NAME (オブジェクト・タイプの外部名) の値だけです。 NAME 値はローカル情報カタログ内で固有でなければなりません。

それ以外の特性名を修正することはできません。 CREATOR、UPDATEBY、および UPDATIME はシステムで生成される値です。DPNAME および PTNAME はオブジェクト・タイプの固有の識別子であり、更新することはできません。システムで生成される特性の値は、オブジェクト・タイプ自体が作成または追加されるときに生成されます。

DPNAME の値は必ず指定する必要があり、ヘッダー・レコード内のオブジェクト・タイプ ID に関連付けられた現行オブジェクト登録の DPNAME と一致しなければなりません。

情報カタログ更新の制御

プログラムを可能な限り、情報カタログと同期化させるために、FLGUpdateReg が正常に完了したあとで、呼び出しを FLGCommit (82ページの『FLGCommit』参照) に組み込む必要があります。 FLGUpdateReg が正常に完了しない場合は、呼び出しを FLGRollback (225ページの『FLGRollback』参照) に組み込む必要があります。

例

図158 は、FLGUpdateReg API 呼び出しを出すために必要な C 言語コードを示しています。このサンプル・コードは、MYIMAGE オブジェクト・タイプに関するオブジェクト・タイプ登録を更新します。この更新では、外部名特性 NAME の値が修正されます。

```
APIRET         rc;                    // Declare reason code
PFLGHEADERAREA pObjRegStruct;         // Pointer to the input structure
UCHAR          pszIconFileID[FLG_ICON_FILE_ID_MAXLEN+1]; // Path/File name of ICON
FLGEXTCODE     ExtCode=0;                  7// Declare extended code
  .
  .  /* updating pObjRegStruct object type     */
     /*                  registration by providing an updated input structure */
  .
strcpy (pszIconFileID,"Y:¥¥FLGICON2.ICO");
rc = FLGUpdateReg (pObjRegStruct,    // Pointer to updated Input Structure
                   pszIconFileID,    // Path/File name of file containing the ICON
                   &ExtCode);             // Pass pointer to extended code
```

図 158. C 言語による FLGUpdateReg の呼び出しのサンプル

258ページの図159 は、更新されるオブジェクト・タイプ登録情報に関する特性と値の情報が入る入力構造 (この C コードの pObjRegStruct ポインターによって指し示されています) を示しています。

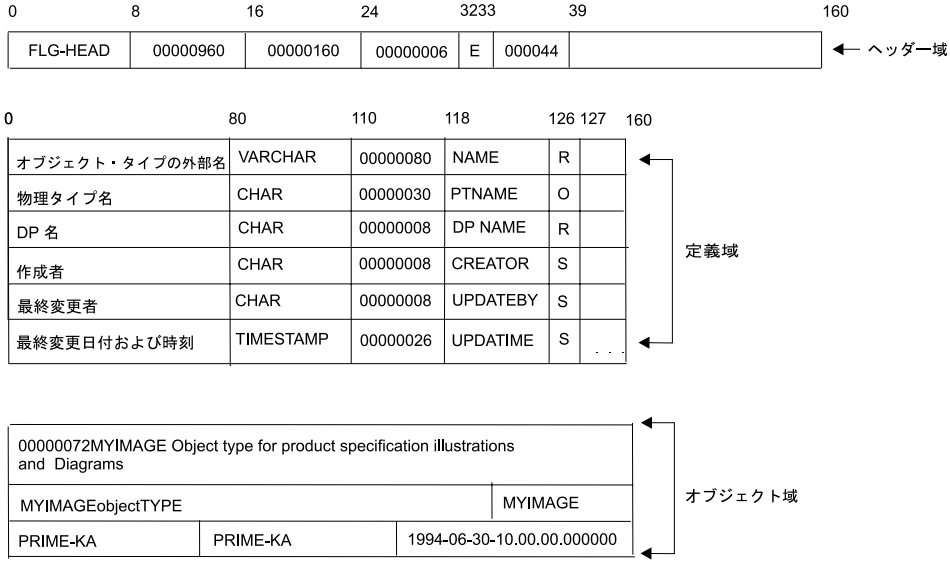


図 159. FLGUpdateReg のための入力構造のサンプル

この例では、システム生成特性 (作成者、最終変更者、および最終変更日付および時間) に対応するオブジェクト域の値は更新することができず 、FLGUpdateReg によって無視されます。入力構造を生成するための方法として、FLGGetReg を出して現行の定義と値を獲得し、その API 呼び出しで得られた出力構造をこの FLGUpdateReg 入力構造のテンプレートとして使用するやり方があることを示すためです。

ヘッダー域の 33 桁目から 38 桁目までには、更新される登録情報に関するオブジェクト・タイプのオブジェクト・タイプ ID (000044) が入ります。

FLGWhereUsed

特定のオブジェクト・インスタンスを含む Grouping オブジェクト・インスタンスのリストを検索するために使用します。

許可

管理者またはユーザー

構文

```
APIRET  APIENTRY   FLGWhereUsed( PSZ              pszFLGID,
                                 PFLGHEADERAREA * ppListStruct,
                                  PFLGEXTCODE  pExtCode );
```

パラメーター

pszFLGID (PSZ) − 入力

含まれるインスタンスを表すシステム生成された固有の ID (16 文字) を指します。

この ID の 1 文字目から 6 文字目までは、このインスタンスのオブジェクト・タイプを識別します。

この ID の 7 文字目から 16 文字目までは、システム生成された固有のインスタンス ID です。

ppListStruct (PFLGHEADERAREA) − 出力

コンテナー・オブジェクトをリストする出力構造リストのポインターのアドレスを指します。

この出力構造には、コンテナー・オブジェクトに関するいくつかの特性仕様と特性値が入ります。各収納用オブジェクトについて、以下の情報が示されます。

- FLGID (16 文字)
- 名前 (80 文字)

すべてのインスタンスは、最初にオブジェクト・タイプ名によって分類され、次に名前によって分類されます。インスタンスの実際の順序は、情報カタログで使用するデータベース管理システムの照合順序によって決まります。

FLGWhereUsed によって戻すことのできるオブジェクト・インスタンスの最大数は計算機で使用可能な記憶域によって異なり、約 5000 です。

出力構造がない場合には、構造を指すポインターは NULL にセットされます。

pExtCode (PFLGEXTCODE) − 出力

理由コードに関連する拡張コードを指します。戻された理由コードに関連した意味のある拡張コードがあるかどうかについては、285ページの『付録 D. 情報カタログ・マネージャーの理由コード』を参照してください。

理由コード (APIRET)

この API 呼び出しの実行結果を表します。

戻された理由コードの説明については、285ページの『付録D. 情報カタログ・マネージャーの理由コード』を参照してください。

出力構造

FLGWhereUsed が作成する出力構造には、図160 に示すように、指定されたオブジェクトを含むオブジェクトのリストが入っています。

出力構造のオブジェクト域には、指定されたオブジェクト・インスタンスを含むすべての Grouping オブジェクトのリストが入ります。戻されたオブジェクト・インスタンスは、FLGID の値とオブジェクト・インスタンスの外部名によって識別されます。

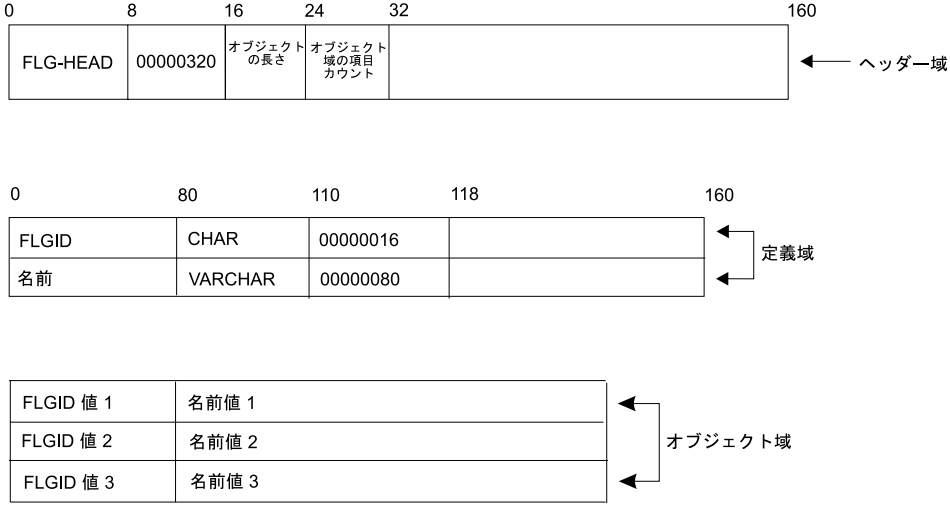


図 160. FLGWhereUsed 出力構造

使用方法

出力構造に割り振られたメモリーの解放

FLGWhereUsed が出力構造にあるデータを戻した場合、出力構造に戻されたデータを保管してから、FLGFreeMem を呼び出さなくてはなりません (137ページの『FLGFreeMem』参照)。メモリーを解放するのに、たとえば C 言語指示のような他の方式は使用しないでください。

例

図161 は、FLGWhereUsed API 呼び出しを出すために必要な C 言語コードを示しています。このサンプル・コードは、FLGWhereUsed API 呼び出しを発行します。

```
APIRET           rc;                // reason code from FLGWhereUsed
UCHAR            pszFLGID[FLG_ID_LEN + 1];    // Information Catalog Manager ID
PFLGHEADERAREA * ppListStruct;   // pointer to output structure pointer
FLGEXTCODE       ExtCode=0;              // extended code
  .
  .  /* provide values for input parameters */
  .
strcpy (pszFLGID, "0000770000003333");
rc = FLGWhereUsed (pszFLGID,
                   ppListStruct, // address of output structure pointer
                   &ExtCode);
```

図 161. C 言語による FLGWhereUsed の呼び出しのサンプル

図162 は、この呼び出しによって得られる出力構造を示しています。

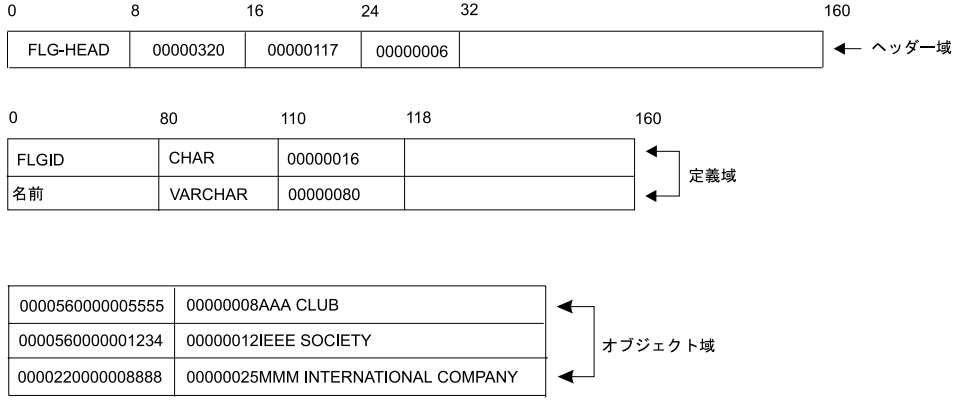


図 162. FLGWhereUsed のための出力構造のサンプル

指定されたオブジェクト・インスタンスは、2 つの異なるオブジェクト・タイプの 3 つのインスタンスに含まれています。オブジェクト・タイプ ID 000056

に対応するオブジェクト・タイプ名は、オブジェクト ID 000022 よりも英字順で小さい値であるため、最初に示されます。

FLGXferTagBuf

削除活動のログである、削除履歴をタグ・ファイルに転送し、配布環境における “シャドウ” 情報カタログのように、ほかの情報カタログ内の削除を重複するために、使用します。

許可

管理者

構文

```
APIRET  APIENTRY   FLGXferTagBuf( PSZ          pszTagFileID,
                                  FLGOPTIONS   Options,
                                  PFLGEXTCODE  pExtCode );
```

パラメーター

pszTagFileID (PSZ) − 入力
出力タグ言語ファイルの名前を指します。このパラメーターは必須パラメーターです。

OS/2 の場合、このパラメーターにはドライブ、ディレクトリー・パス、およびファイル名が含まれており、ファイル割り振りテーブル (FAT) または HPFS ファイルとして有効な値でなければなりません。 (ドライブおよびディレクトリーを除く) ファイル名とエクステンションは、 240 文字を超えてはなりません。

このファイルのターゲット・ドライブは、固定ディスクであっても、取り外し可能ディスクであっても差し支えありません。

Options (FLGOPTIONS) − 入力
削除履歴を転送したいファイルについて、以下のいずれかのオプションを選択します。

FLG_TAGOPT_NEW
新規ファイルを作成します。

FLG_TAGOPT_REPLACE
既存のファイルを置き換えます。

pExtCode (PFLGEXTCODE) − 出力
理由コードに関連する拡張コードを指します。戻された理由コードに関連した意味のある拡張コードがあるかどうかについては、285ページの『付録 D. 情報カタログ・マネージャーの理由コード』を参照してください。

理由コード (APIRET)

この API 呼び出しの実行結果を表します。

戻された理由コードの説明については、285ページの『付録D. 情報カタログ・マネージャーの理由コード』を参照してください。

使用方法

ターゲット・ディスクが満杯の場合、ディスクが取り外し可能であっても、FLGXferTagBuf は異常終了します。

特に Grouping オブジェクト・インスタンスやオブジェクト・タイプを削除している場合は、他の情報カタログで、誤って削除をしないために、削除履歴タグ・ファイルを、他の情報カタログにエクスポートするまえに内容を点検してください。

例

図163 は、FLGXferTagBuf 呼び出しを出すために必要な C 言語コードを示しています。このサンプル・コードは、ファイル c:¥sampdel.tag を作成してから、そこに削除履歴を転送します。

```
APIRET          rc;               // reason code from API
PSZ             pszTagFile = "c:¥¥sampdel.tag";
FLGEXTCODE      xc=0;             // extended code
FLGOPTIONS      Options=0;
  .
  .  /*                                                 */
  .
Options=Options | FLG_TAGOPT_NEW;
rc = FLGXferTagBuf (pszTagFile,
                    Options,
                    &xc);
```

図 163. C 言語による FLGXferTagBuf の呼び出しのサンプル

付録A. サンプル・プログラム DG2SAMP.C

情報カタログ・マネージャーには、コンパイル、連係、および実行することのできるサンプル・プログラム DG2SAMP.C が用意されています。
DG2SAMP.C は、DB2 UDB が導入されたドライブの ¥SQLLIB¥LIB ディレクトリーに入っています。このサンプル・プログラムを使用すると、ユーザーは、次のようにしてオブジェクト・インスタンスの名前を変更することができます。

1. 情報カタログ内のオブジェクト・タイプのリストを獲得します。
2. 探しているオブジェクトが存在するときには、それを検出します。
3. インスタンスに関する情報を獲得します。
4. 名前特性の値を更新します。

このプログラムは、以下の API 呼び出しを出します。

- FLGCommit
- FLGFreeMem
- FLGGetInst
- FLGInit
- FLGListObjTypes
- FLGRollback
- FLGSearch
- FLGTerm
- FLGTrace
- FLGUpdateInst

DG2SAMP.C のコンパイル

Microsoft Visual C++ コンパイラーを使用して DG2SAMP.C をコンパイルするには、 DG2SAMP.C と同じディレクトリーから次のコマンドを出す必要があります。

```
cl /c DG2SAMP.C
```

DG2SAMP.C の連係

それぞれの Microsoft Visual C++ コンパイラー・プログラムを連係するには、DG2SAMP.C と同じディレクトリーから次のコマンドを出してください。

```
link /dll dgwapi.lib dg2samp.obj
```

DG2SAMP.C の実行

この例は、情報カタログ・マネージャーで提供される DGSAMPLE 情報カタログを使用するもので、ユーザーがこの情報カタログに対する管理者権限を得ていることを想定しています。

1. コマンド DG2SAMP を入力します。
2. ユーザー ID を入力します。
3. ユーザーのパスワードを入力します。
4. 情報カタログの名前を入力します。

   このシナリオでは、DGSAMPLE と入力します。
5. 変更したいオブジェクトのオブジェクト・タイプの外部名を入力します。

   このシナリオでは、Business groupings と入力します。
6. 変更したいオブジェクトの外部名を入力します。

   このシナリオでは、Billings と入力します。
7. このオブジェクトの新しい外部名を入力します。

   このシナリオでは、Account payment histories と入力します。

付録B. 情報カタログ・マネージャー API ヘッダー・ファイル ー DG2APIH

情報カタログ・マネージャーでは、ヘッダー・ファイル DG2API.H が提供されており、情報カタログ・マネージャー API 呼び出しを使用する C 言語アプリケーションに必要な API 呼び出し、定数、およびデータ・タイプの関数原型を定義します。

DG2API.H は、情報カタログ・マネージャーが導入されたドライブの VWSLIB¥LIB ディレクトリーに導入されています。

情報カタログ・マネージャー (Windows 版) を使用して DG2API.H で定義されている定義タイプを使用するには、Microsoft Visual C++ コンパイラーに含まれている WINDOWS.H ヘッダー・ファイルをプログラムに組み込む必要があります。

DG2API.H で定義されている定数

表19 には、情報カタログ・マネージャー API 呼び出しを使用して情報カタログ・マネージャー関数にアクセスするプログラム用に定義されている変数を示しています。

表 19. DG2API.H で定義されている定数

入出力構造のヘッダー域定数	バイト	定義される長さ
FLG_H_IDENT_LEN	8	構造の識別子 (FLG-HEAD)
FLG_H_DEFAREA_LEN	8	定義の長さ
FLG_H_OBJAREA_LEN	8	オブジェクト域の長さ
FLG_H_OBJAREAENT_LEN	8	オブジェクト域の項目カウント
FLG_H_CATEGORY_LEN	1	区分
FLG_H_OBJTYPID_LEN	6	オブジェクト・タイプの ID
FLG_H_RESERVED_LEN	121	予約域
FLG_HEADER_SIZE	160	ヘッダー域
入出力構造定義域の長さ	**バイト**	**定義される長さ**
FLG_D_PROPNM_LEN	80	特性の名前
FLG_D_DATATYP_LEN	30	データ・タイプの値
FLG_D_DATA_LEN	8	データ長の値
FLG_D_PPN_LEN	8	特性省略名

表 19. DG2API.H で定義されている定数 (続き)

FLG_D_VF_LEN	1	値のフラグ
FLG_D_US_LEN	1	UUI の順序番号
FLG_D_CS_LEN	1	大文字小文字の区別フラグ
FLG_D_FS_LEN	1	ファジー探索フラグ
FLG_D_RESERVED_LEN	30	予約域
FLG_DEFINITION_SIZE	160	定義域レコード
情報カタログ・マネージャー・ストリングの長さ	**バイトの長さ**	**定義される長さ**
FLG_OBJTYPID_LEN	6	オブジェクト・タイプの ID
FLG_INSTIDNT_LEN	10	インスタンスの ID
FLG_INST_NAME_LEN	80	インスタンスの名前
FLG_UPDATIME_LEN	26	オブジェクト・タイプが作成または更新されたときのタイム・スタンプ
FLG_UPDATEBY_LEN	8	更新を実行した人のユーザー ID
FLG_ID_LEN	16	FLGID の値
FLG_EXTERNAL_NAME_LEN	80	オブジェクト・タイプの外部名
FLG_PTNAME_LEN	30	オブジェクト・タイプの物理タイプ名
FLG_DPNAME_LEN	8	オブジェクト・タイプの省略名
FLG_CREATOR_LEN	8	オブジェクト・タイプ作成者のユーザー ID
FLG_USERID_LEN	8	ログオン・ユーザー ID
FLG_PASSWORD_LEN	8	ログオン・パスワード
FLG_DATABASENAME_LEN	8	情報カタログ・マネージャー・データベースの名前
FLG_VARIABLE_DATA_LENGTH_LEN	8	VARCHAR 値および LONG VARCHAR 値の長さフィールド
データ・タイプの最大長	**バイト**	**定義される最大長**
FLG_CHAR_MAXLEN	254	CHAR データ・タイプ
FLG_VARCHAR_MAXLEN	4000	VARCHAR データ・タイプ
FLG_LONG_VARCHAR_MAXLEN	32700	LONG VARCHAR データ・タイプ
FLG_TIMESTAMP_MAXLEN	26	TIMESTAMP データ・タイプ
最大値	**値**	**定義される最大値**
FLG_REG_NUM_PROPERTIES	6	登録特性の数

表 19. DG2API.H で定義されている定数 (続き)

FLG_ICON_FILE_ID_MAXLEN	259	アイコン・ファイルのパス、ファイル名、およびエクステンションの長さ
FLG_TAG_FILE_ID_MAXLEN	259	タグ言語ファイルのパス、ファイル名、およびエクステンションの長さ
FLG_LOG_FILE_ID_MAXLEN	259	ログ・ファイルのパス、ファイル名、およびエクステンションの長さ
FLG_ECHO_FILE_ID_MAXLEN	259	エコー・ファイルのパス、ファイル名、およびエクステンションの長さ
FLG_ICON_PATH_MAXLEN	246	アイコン・パスの長さ
FLG_ICON_MAXLEN	30000	アイコン・ファイルのサイズ
FLG_UUI_MAXLEN	254	UUI 特性値の長さ (バイト数)
FLG_MAX_PROPERTIES	255	オブジェクト・タイプ内の特性の数
FLG_MAX_NUM_LONG_VARCHARS	14	LONG VARCHAR データ・タイプの特性の数
FLG_MAXLEN_SEARCH_LONGVARCHAR	3000	LONG VARCHAR データ・タイプの特性の探索基準の長さ
FLG_MAX_IMP_EXP_OBJTYPES	3500	単一のタグ言語ファイルで処理される固有のオブジェクト・タイプの数
FLG_MAX_ANCHOR_NUM	1600	FLGListAnchors により戻されるオブジェクト・インスタンスの数
FLG_MAX_ORPHAN_NUM	1600	FLGListOrphans により戻されるオブジェクト・インスタンスの数
FLG_MAX_CONTAINEE_NUM	1600	FLGFoundIn により戻されるオブジェクト・インスタンスの数

FLGConvertID の入力パラメーター	**値**	**説明**
FLG_DPNAME	'D'	オブジェクト・タイプ ID に変換されるオブジェクト・タイプの省略名

表 19. DG2API.H で定義されている定数 (続き)

FLG_FLGID	'F'	拡張オブジェクト・インスタンス名に変換するオブジェクト・インスタンス ID
FLGFoundIn、FLGListAssociates、および FLGListOrphans の入力オプション	**値**	**説明**
FLG_LIST_PROGRAM	0x00000001	指定されたオブジェクト・タイプに関連した Program オブジェクト・インスタンスを検索する
FLG_LIST_CONTAIN	0x00000002	指定されたインスタンスに含まれるオブジェクト・インスタンスを検索する
FLG_LIST_CONTACT	0x00000003	指定されたインスタンスに関連した Contact オブジェクト・インスタンスを検索する
FLG_LIST_ATTACHMENT	0x00000004	指定されたインスタンスに付加されたオブジェクト・インスタンスを検索する
FLG_LIST_COMMENTS	0x00000005	指定されたインスタンスに付加された Comments オブジェクト・インスタンスを検索する
FLG_LIST_LINK	0x00000006	指定されたインスタンスにリンクしたオブジェクト・インスタンスを検索する
FLGManageIcons のプラットフォーム・オプション	**長さ**	**説明**
FLG_PLATFORM_OS2	0x00000100	OS/2 アイコン
FLG_PLATFORM_WINDOWS	0x00000200	Windows アイコン
FLGManageIcons の出力オプション	**長さ**	**説明**
FLG_ICON_EXIST	0x00000001	指定されたアイコンが存在する
FLG_ICON_NOTEXIST	0x00000002	指定されたアイコンが存在しない
FLGRelation のオプション	**値**	**説明**
FLG_CREATE_RELATION	'C'	作成オプション
FLG_DELETE_RELATION	'D'	削除オプション
FLGRelation で定義される関係のタイプ	**値**	**説明**

表 19. DG2API.H で定義されている定数 (続き)

FLG_ATTACHMENT_RELATION	'A'	アタッチメント関係
FLG_CONTAINER_RELATION	'C'	包含関係
FLG_CONTACT_RELATION	'T'	連絡担当者関係
FLG_LINK_RELATION	'L'	リンク関係
FLGXferTagBuf のファイル・オプション	**値**	**説明**
FLG_TAGOPT_NEW	0x00000001	削除履歴を転送する新規ファイルの作成
FLG_TAGOPT_REPLACE	0x00000002	削除履歴の以前の内容を置き換え、既存のファイルに転送する
FLGManageTagBuf の入出力 (I/O) オプション	**値**	**説明**
FLG_TAGBUF_RESET	0x00000001	削除履歴ログから既存の項目を取り外す
FLG_TAGBUF_QUERY	0x00000002	削除履歴ログが項目を含むかどうか照会する
FLG_TAGBUF_EMPTY	0x00000001	削除履歴ログに項目が含まれない
FLG_TAGBUF_NOT_EMPTY	0x00000002	削除履歴ログに項目が含まれる
FLGDeleteTree の削除オプション	**値**	**説明**
FLG_DELTREE_REL	0x00000001	Grouping オブジェクト・インスタンスとその基礎となるツリー構造を削除する
FLG_DELTREE_ALL	0x00000002	Grouping オブジェクト・インスタンスと、その基礎となるオブジェクト、ツリー構造を削除する
FLGManageCommentStatus 出力構造特性省略名	**値**	**説明**
FLG_COMMENT_STATUS1_PPN	"CSTATUS1"	コメント用に第一に利用可能な現在の選択項目
FLG_COMMENT_STATUS2_PPN	"CSTATUS2"	コメント用に 2 番目に利用可能な現在の選択項目
FLG_COMMENT_STATUS3_PPN	"CSTATUS3"	コメント用に 3 番目に利用可能な現在の選択項目
FLG_COMMENT_STATUS4_PPN	"CSTATUS4"	コメント用に 4 番目に利用可能な現在の選択項目
FLG_COMMENT_STATUS5_PPN	"CSTATUS5"	コメント用に 5 番目に利用可能な現在の選択項目

表 19. DG2API.H で定義されている定数 (続き)

FLG_COMMENT_STATUS6_PPN	"CSTATUS6"	コメント用に 6 番目に利用可能な現在の選択項目
FLG_COMMENT_STATUS7_PPN	"CSTATUS7"	コメント用に 7 番目に利用可能な現在の選択項目
FLG_COMMENT_STATUS8_PPN	"CSTATUS8"	コメント用に 8 番目に利用可能な現在の選択項目
FLG_COMMENT_STATUS9_PPN	"CSTATUS9"	コメント用に 9 番目に利用可能な現在の選択項目
FLG_COMMENT_STATUSA_PPN	"CSTATUSA"	コメント用に 10 番目に利用可能な現在の選択項目

FLGManageUsers のユーザー・タイプ・オプション	**値**	**説明**
FLG_USERTYPE_PADMIN	'A'	1 次管理者
FLG_USERTYPE_BADMIN	'B'	バックアップ管理者
FLG_USERTYPE_POWERUSER	'D'	追加のオブジェクト管理タスクの実行を許可されたユーザー
FLG_USERTYPE_USER	'W'	ユーザー

FLGTrace で設定されるトレース・レベル・オプション	**値**	**説明**
FLG_TRACE_ON	1	トレースをオンにする
FLG_TRACE_OFF	0	トレースをオフにする
FLG_TRACELEVEL_0	0	デフォルトのトレース・レベル
FLG_TRACELEVEL_1	1	関数の入り口および出口のレコードを含める
FLG_TRACELEVEL_2	2	関数レベル情報を含める
FLG_TRACELEVEL_3	3	入出力パラメーターを含める
FLG_TRACELEVEL_4	4	情報カタログ・マネージャーに引き渡され、あるいは情報カタログ・マネージャーによって使用されるすべての入出力構造を含める

情報カタログ・マネージャーにより実行されるアクション	**値**	**説明**
FLG_ACTION_CREATE	0x00000001	作成。たとえば、アイコンをオブジェクト・タイプに追加したり、ユーザーを情報カタログに追加したりする

表 19. DG2API.H で定義されている定数 (続き)

FLG_ACTION_DELETE	0x00000002	削除。たとえば、アイコンをオブジェクト・タイプから削除する
FLG_ACTION_UPDATE	0x00000004	更新。たとえば、コメント用に現在利用可能な選択項目のリストの変更や、削除履歴をトグル・オンまたはオフしたりする
FLG_ACTION_GET	0x00000008	現在の設定またはリストを検索する
FLG_ACTION_QUERY	0x00000010	存在を判別する
FLG_ACTION-LIST	0x00000020	ユーザーのリストを検索する
区分タイプ	**値**	**説明**
FLG_GROUPING_OBJ	'G'	Grouping 区分
FLG_ELEMENTAL_OBJ	'E'	Elemental 区分
FLG_CONTACT_OBJ	'C'	Contact 区分
FLG_DICTIONARY_OBJ	'D'	Dictionary 区分
FLG_PROGRAM_OBJ	'P'	Program 区分
FLG_SUPPORT_OBJ	’S'	Support 区分
FLG_ATTACHMENT_OBJ	'A'	Attachment 区分
Yes および no 値	**値**	**説明**
FLG_YES	'Y'	Yes
FLG_NO	'N'	No
値のフラグ	**値**	**説明**
FLG_REQUIRED	'R'	必須特性
FLG_OPTIONAL	'O'	任意選択の特性
FLG_SYSTEM	’S'	システム生成の特性
汎用固有識別コードの順序番号	**値**	**説明**
FLG_UUI_1	'1'	UUI における最初の特性
FLG_UUI_2	'2'	UUI における 2 番目の特性
FLG_UUI_3	'3'	UUI における 3 番目の特性
FLG_UUI_4	'4'	UUI における 4 番目の特性
FLG_UUI_5	'5'	UUI における 5 番目の特性
FLG_BLANK	' '	単一のブランク文字
必須特性の特性省略名	**値**	**説明**
FLG_PPN_OBJTYPID	"OBJTYPID"	オブジェクト・タイプの ID
FLG_PPN_INSTIDNT	"INSTIDNT"	インスタンスの ID

表 19. DG2API.H で定義されている定数 (続き)

FLG_PPN_INST_NAME	"NAME"	オブジェクト・インスタンスの名前
FLG_PPN_UPDATIME	"UPDATIME"	最後に更新されたときの日付および時刻のタイム・スタンプ
FLG_PPN_UPDATEBY	"UPDATEBY"	最後に更新を実行した人のユーザー ID
FLG_PPN_EXTERNAL_NAME	"NAME"	オブジェクト・タイプの外部名
FLG_PPN_PTNAME	"PTNAME"	オブジェクト・タイプの物理タイプ名
FLG_PPN_DPNAME	"DPNAME"	オブジェクト・タイプの DP 名 (省略名)
FLG_PPN_CREATOR	"CREATOR"	オブジェクト・タイプを作成した人のユーザー ID

共通特性の省略名 ― 情報カタログ・マネージャーが定義したオブジェクト・タイプ	値	説明
FLG_PPN_UUICLASS	"UUICLASS"	オブジェクト・クラスの定義
FLG_PPN_UUIQUAL1	"UUIQUAL1"	識別子 UUI 値が UUI クラス内で固有であることを確認するのに使用される第 1 の修飾子特性
FLG_PPN_UUIQUAL2	"UUIQUAL2"	識別子 UUI 値が UUI クラス内で固有であることを確認するのに使用される第 2 の修飾子特性
FLG_PPN_UUIQUAL3	"UUIQUAL3"	識別子 UUI 値が UUI クラス内で固有であることを確認するのに使用される第 3 の修飾子特性
FLG_PPN_UUIDENT	"UUIDENT"	UUI の一部として使用される固有のオブジェクト・インスタンス・レベル識別子
FLG_PPN_HANDLES	"HANDLES"	プログラム関連によりハンドルされたオブジェクト・タイプを識別する
FLG_PPN_STARTCMD	"STARTCMD"	オブジェクト・タイプに関連したプログラムを呼び出すコマンド
FLG_PPN_PARMLIST	"PARMLIST"	呼び出し時にプログラムに渡すパラメーター・リスト

表 19. DG2API.H で定義されている定数 (続き)

FLG_PPN_SHRTDESC	"SHRTDESC"	オブジェクト・インスタンスの短い説明
FLG_PPN_CREATSTP	"CREATSTP"	Comments オブジェクトの作成日のタイム・スタンプ
FLG_PPN_STATUS	"STATUS"	Comments オブジェクト・インスタンスの状況
FLG_PPN_ACTIONS	"ACTIONS"	たとえばプログラムの開始など、オブジェクト・インスタンスに対して取られるアクション
FLG_PPN_EXTRA	"EXTRA"	予約済
FLG_PPN_LONGDESC	"LONGDESC"	オブジェクト・インスタンスの長い説明

FLGImport 再始動オプション	**値**	**説明**
FLG_RESTART_BEGIN	'B'	タグ言語ファイルを最初からインポートする
FLG_RESTART_CHECKPT	'C'	最後にコミットされたチェックポイントからタグ言語ファイルをインポートする

入力構造特性名をエクスポートする	**値**	**説明**
FLG_EXPORT_DEFAREA_FLGID	"FLGID"	インポートされるオブジェクトの FLGID を引き渡す特性
FLG_EXPORT_DEFAREA_ATTACHMENT_IND	"ATTACHMENT-IND"	関連する Attachment オブジェクトをエクスポートするかどうかを示す特性
FLG_EXPORT_DEFAREA_CONTAINEE_IND	"CONTAINEE-IND"	含まれているオブジェクトをエクスポートするかどうかを示す特性
FLG_EXPORT_DEFAREA_CONTACT_IND	"CONTACT-IND"	関連する連絡先オブジェクトをエクスポートするかどうかを示す特性
FLG_EXPORT_DEFAREA_LINK_IND	"LINK-IND"	リンク・オブジェクトをエクスポートするかどうかを示す特性

データ・タイプ	**値**	**説明**
FLG_DTYPE_CHAR	"CHAR"	固定長文字
FLG_DTYPE_VARCHAR	"VARCHAR"	可変長文字
FLG_DTYPE_LONGVARCHAR	"LONG VARCHAR"	長い可変長文字
FLG_DTYPE_TIMESTAMP	"TIMESTAMP"	タイム・スタンプ

DG2API·H で定義されている定数

表 19. DG2API.H で定義されている定数 (続き)

ヘッダー域構造の識別子	値	説明
FLG_H_IDENT	"FLG-HEAD"	情報カタログ・マネージャー入出力構造における最初の値
データベース・プラットフォームの識別子	**値**	**説明**
FLG_DG2_DB22	0	DB2 UDB (OS/2 版) を使用する情報カタログ・マネージャー
FLG_DG2_DB2	1	DB2 UDB (OS/390 版)® または DB2 (OS/390 版) を使用する情報カタログ・マネージャー
FLG_DG2MVS	2	予約済
FLG_DG2_DB400	4	DB2 UDB (AS/400 版) を使用する情報カタログ・マネージャー
FLG_DG2_DB6000	6	DB2 UDB (AIX 版) を使用する情報カタログ・マネージャー
FLG_DG2_DB26000PE	7	DB2 PE for AIX または DB2 UDB EEE を使用する情報カタログ・マネージャー
FLG_DG2_DB2NT	8	DB2 UDB (Windows NT 版) を使用する情報カタログ・マネージャー
FLG_DG2_DB295	9	DB2 UDB (Windows 95 版) を使用する情報カタログ・マネージャー

DG2API.H における構造およびデータ・タイプの定義

277ページの表20 および 277ページの表21 には、情報カタログ・マネージャー API 呼び出しで使用される構造およびデータ・タイプの定義が示されています。

表 20. 構造の定義

ヘッダー域	説明

`typedef struct _FLG_HEADER_AREA {`
`UCHAR pchHIdent [ FLG_H_IDENT_LEN ];`
`UCHAR pchHDefLength [ FLG_H_DEFAREA_LEN ];`
`UCHAR pchHObjLength [ FLG_H_OBJAREA_LEN ];`
`UCHAR pchHObjEntryCount [ FLG_H_OBJAREAENT_LEN ];`
`UCHAR pchHCategory [ FLG_H_CATEGORY_LEN ];`
`UCHAR pchHObjTypeId [ FLG_H_OBJTYPID_LEN ];`
`UCHAR pchHReserved [ FLG_H_RESERVED_LEN ];`
`} FLGHEADERAREA;`
`#ifdef WINDOWS`
`typedef FLGHEADERAREA __huge *PFLGHEADERAREA;`
`#else`
`typedef FLGHEADERAREA *PFLGHEADERAREA;`
`#endif`
	情報カタログ・マネージャー入出力構造のヘッダー域のすべての要素を含む構造を定義する

定義域	**説明**

`typedef struct _FLG_DEFINITION_AREA {`
`UCHAR pchDPropName [ FLG_D_PROPNM_LEN ];`
`UCHAR pchDDataType [ FLG_D_DATATYP_LEN ];`
`UCHAR pchDDataLength [ FLG_D_DATA_LEN ];`
`UCHAR pchDTagName [ FLG_D_PPN_LEN ];`
`UCHAR pchDVF [ FLG_D_VF_LEN ];`
`UCHAR pchDUS [ FLG_D_US_LEN ];`
`UCHAR pchDCS [ FLG_D_CS_LEN ];`
`UCHAR pchDFS [ FLG_D_FS_LEN ];`
`UCHAR pchDReserved [ FLG_D_RESERVED_LEN ];`
`} FLGDEFINITIONAREA;`
`#ifdef WINDOWS`
`typedef FLGDEFINITIONAREA __huge *PFLGDEFINITIONAREA;`
`#else`
`typedef FLGDEFINITIONAREA *PFLGDEFINITIONAREA;`
`#endif`
	情報カタログ・マネージャー入出力構造の定義域レコードのすべての要素を含む構造を定義する

表 21. データ・タイプの定義

データ・タイプの同義語	データ・タイプ
FLGRELOPTION	UCHAR－無符号文字
FLGRELTYPE	UCHAR－無符号文字
FLGTRACEOPTION	ULONG－符号なし長整数
FLGIDLENGTH	ULONG－符号なし長整数
FLGOPTIONS	ULONG－符号なし長整数
PFLGOPTIONS	* FLGOPTIONS－無符号長整数を指すポインター
FLGADMIN	UCHAR－無符号文字
FLGRESTARTOPTION	UCHAR－無符号文字
FLGEXTCODE	LONG－長整数
PFLGEXTCODE	* FLGEXTCODE－長整数を指すポインター

情報カタログ・マネージャー API 呼び出しの関数原型

表22 には、情報カタログ・マネージャー API 呼び出しの関数原型が定義されています。

表 22. API 呼び出しの関数原型

FLGAppendType

```
APIRET  APIENTRY    FLGAppendType( PFLGHEADERAREA   pObjTypeStruct,
                                   PFLGEXTCODE      pExtCode );
```

FLGCommit

```
APIRET  APIENTRY   FLGCommit( PFLGEXTCODE      pExtCode );
```

FLGConvertID

```
APIRET  APIENTRY  FLGConvertID( PSZ              pszInBuffer,
                              PSZ              pszOutBuffer,
                              FLGOPTIONS       Options,
                              PFLGEXTCODE      pExtCode );
```

FLGCreateInst

```
APIRET  APIENTRY    FLGCreateInst( PFLGHEADERAREA   pObjInstStruct,
                                   PSZ              pszFLGID,
                                   PFLGEXTCODE      pExtCode );
```

FLGCreateReg

```
APIRET  APIENTRY  FLGCreateReg( PFLGHEADERAREA   pObjRegStruct,
                              PSZ              pszIconFileID,
                              PSZ              pszObjTypeID,
                              PFLGEXTCODE      pExtCode );
```

FLGCreateType

```
APIRET  APIENTRY   FLGCreateType( PFLGHEADERAREA   pObjTypeStruct,
                                  PFLGEXTCODE      pExtCode );
```

FLGDeleteInst

```
APIRET  APIENTRY    FLGDeleteInst( PSZ              pszFLGID,
                                   PFLGEXTCODE      pExtCode );
```

FLGDeleteReg

```
APIRET  APIENTRY  FLGDeleteReg( PSZ            pszObjTypeID,
                              PFLGEXTCODE    pExtCode );
```

FLGDeleteTree

表 22. API 呼び出しの関数原型 (続き)

```
APIRET  APIENTRY   FLGDeleteTree( PSZ              pszFLGID,
                                  FLGOPTIONS       Options,
                                  PFLGHEADERAREA * ppListStruct,
                                  PFLGEXTCODE      pExtCode );
```

FLGDeleteType

```
APIRET  APIENTRY   FLGDeleteType( PSZ              pszObjTypeID,
                                  PFLGEXTCODE      pExtCode );
```

FLGDeleteTypeExt

```
APIRET  APIENTRY   FLGDeleteTypeExt( PSZ              pszObjTypeID,
                                     PFLGEXTCODE      pExtCode );
```

FLGExport

```
APIRET  APIENTRY   FLGExport( PSZ              pszTagFileID,
                              PSZ              pszLogFileID,
                              PSZ              pszIcoPath,
                              PFLGHEADERAREA   pListStruct,
                              PFLGEXTCODE      pExtCode );
```

FLGFoundIn

```
APIRET  APIENTRY   FLGFoundIn( PSZ              pszFLGID,
                               FLGOPTIONS       Options,
                               PFLGHEADERAREA * ppListStruct,
                               PFLGEXTCODE      pExtCode );
```

FLGFreeMem

```
APIRET  APIENTRY    FLGFreeMem( PFLGHEADERAREA   pFLGOutputStruct,
                                PFLGEXTCODE      pExtCode );
```

FLGGetInst

```
APIRET  APIENTRY     FLGGetInst( PSZ              pszFLGID,
                                 PFLGHEADERAREA * ppObjInstStruct,
                                 PFLGEXTCODE      pExtCode );
```

FLGGetReg

```
APIRET  APIENTRY   FLGGetReg( PSZ              pszObjTypeID,
                              PSZ              pszIconFileID,
                              PFLGHEADERAREA * ppObjRegStruct,
                              PFLGEXTCODE      pExtCode );
```

FLGGetType

```
APIRET  APIENTRY    FLGGetType( PSZ              pszObjTypeID,
                                PFLGHEADERAREA * ppObjTypeStruct,
                                PFLGEXTCODE      pExtCode );
```

FLGImport

表 22. API 呼び出しの関数原型 (続き)

```
APIRET  APIENTRY   FLGImport( PSZ              pszTagFileID,
                              PSZ              pszLogFileID,
                              PSZ              pszIcoPath,
                              FLGRESTARTOPTION RestartOpt,
                              PFLGEXTCODE      pExtCode );
```

FLGInit

```
APIRET  APIENTRY   FLGInit( PSZ              pszUserID,
                           PSZ              pszPassword,
                           PSZ              pszDatabaseName,
                           FLGADMIN         Admin,
                           PFLGHEADERAREA * ppListStruct,
                           PFLGEXTCODE      pExtCode );
```

FLGListAnchors

```
APIRET  APIENTRY   FLGListAnchors( PFLGHEADERAREA * ppListStruct,
                                  PFLGEXTCODE      pExtCode );
```

FLGListAssociates

```
APIRET  APIENTRY   FLGListAssociates( PSZ              pszInBuffer,
                                     FLGOPTIONS       Options,
                                     PFLGHEADERAREA * ppListStruct,
                                     PFLGEXTCODE      pExtCode );
```

FLGListContacts

```
APIRET  APIENTRY  FLGListContacts( PSZ              pszFLGID,
                                  PFLGHEADERAREA * ppListStruct,
                                  PFLGEXTCODE      pExtCode );
```

FLGListObjTypes

```
APIRET  APIENTRY  FLGListObjTypes( PFLGHEADERAREA * ppListStruct,
                                  PFLGEXTCODE      pExtCode );
```

FLGListOrphans

```
APIRET  APIENTRY   FLGListOrphans( PSZ              pszObjTypeID,
                                  FLGOPTIONS       Options,
                                  PFLGHEADERAREA * ppListStruct,
                                  PFLGEXTCODE      pExtCode );
```

FLGListPrograms

```
APIRET  APIENTRY  FLGListPrograms( PSZ              pszObjTypeID,
                                  PFLGHEADERAREA * ppListStruct,
                                  PFLGEXTCODE      pExtCode );
```

FLGManageCommentStatus

```
APIRET  APIENTRY   FLGManageCommentStatus( FLGOPTIONS       Action,
                                           FLGHEADERAREA  * pStatusStruct,
                                           PFLGHEADERAREA * ppStatusStruct,
                                           PFLGEXTCODE      pExtCode );
```

表 22. API 呼び出しの関数原型 (続き)

FLGManageFlags

```
APIRET  APIENTRY   FLGManageFlags( FLGOPTIONS   Action,
                                   FLGOPTIONS   FlagType,
                                   UCHAR        chValue,
                                   UCHAR      * pchValue,
                                   PFLGEXTCODE  pExtCode );
```

FLGManageIcons

```
APIRET  APIENTRY   FLGManageIcons( PSZ          pszObjTypeID,
                                   PSZ          pszIconFileID,
                                   FLGOPTIONS   InOptions,
                                   PFLGOPTIONS  pOutOptions,
                                   PFLGEXTCODE  pExtCode );
```

FLGManageTagBuf

```
APIRET  APIENTRY   FLGManageTagBuf( FLGOPTIONS       InOptions,
                                    PFLGOPTIONS      pOutOptions,
                                    PFLGEXTCODE      pExtCode );
```

FLGManageUsers

```
APIRET  APIENTRY   FLGManageUsers( FLGOPTIONS       Options,
                                   PFLGHEADERAREA   pListStruct,
                                   PFLGHEADERAREA * ppListStruct,
                                   PFLGEXTCODE      pExtCode );
```

FLGMdisExport

```
APIRET  APIENTRY   FLGMdisExport( PSZ          pszTagFileName,
                                  PSZ          pszLogFileName,
                                  PSZ          pszObjTypeName,
                                  PSZ          pszObjectName,
                                  PFLGEXTCODE  pExtCode );
```

FLGMdisImport

```
APIRET  APIENTRY   FLGMdisImport( PSZ             pszTagFileID,
                                  PSZ             pszLogFileID,
                                  PFLGEXTCODE     pExtCode );
```

FLGNavigate

```
APIRET  APIENTRY    FLGNavigate( PSZ             pszFLGID,
                                 PFLGHEADERAREA * ppListStruct,
                                 PFLGEXTCODE     pExtCode );
```

FLGOpen

```
APIRET  APIENTRY   FLGOpen( PSZ             pszPgmFLGID,
                            PSZ             pszObjFLGID,
                            PFLGEXTCODE     pExtCode );
```

FLGRelation

表 22. API 呼び出しの関数原型 (続き)

```
APIRET  APIENTRY   FLGRelation( PSZ              pszSrcFLGID,
                                PSZ              pszTrgFLGID,
                                FLGRELTYPE       RelType,
                                FLGRELOPTION     RelOpt,
                                PFLGEXTCODE      pExtCode );
```

FLGRollback

```
APIRET  APIENTRY    FLGRollback( PFLGEXTCODE      pExtCode );
```

FLGSearch

```
APIRET  APIENTRY    FLGSearch( PSZ              pszObjTypeID,
                               PFLGHEADERAREA   pSelCriteriaStruct,
                               PFLGHEADERAREA * ppListStruct,
                               PFLGEXTCODE      pExtCode );
```

FLGSearchAll

```
APIRET  APIENTRY   FLGSearchAll( PFLGHEADERAREA   pSelCriteriaStruct,
                                 PFLGHEADERAREA * ppListStruct,
                                 PFLGEXTCODE      pExtCode );
```

FLGTerm

```
APIRET  APIENTRY   FLGTerm( PFLGEXTCODE   pExtCode );
```

FLGTrace

```
APIRET  APIENTRY   FLGTrace( FLGTRACEOPTION   TraceOpt,
                             PFLGEXTCODE      pExtCode );
```

FLGUpdateInst

```
APIRET  APIENTRY    FLGUpdateInst( PFLGHEADERAREA   pObjInstStruct,
                                   PFLGEXTCODE      pExtCode );
```

FLGUpdateReg

```
APIRET  APIENTRY   FLGUpdateReg( PFLGHEADERAREA   pObjRegStruct,
                                 PSZ              pszIconFileID,
                                 PFLGEXTCODE      pExtCode );
```

FLGWhereUsed

```
APIRET  APIENTRY   FLGWhereUsed( PSZ              pszFLGID,
                                 PFLGHEADERAREA * ppListStruct,
                                 PFLGEXTCODE      pExtCode );
```

FLGXferTagBuf

```
APIRET  APIENTRY   FLGXferTagBuf( PSZ          pszTagFileID,
                                  FLGOPTIONS   Options,
                                  PFLGEXTCODE  pExtCode );
```

付録C. 情報カタログ・マネージャーの限界値

表23 は、いくつかの情報カタログ・マネージャーの限界値を示しています。

表 23. 情報カタログ・マネージャーの限界値

情報カタログ・マネージャーの値	限界値
最長の情報カタログ・データベース名	30 文字
最長の情報カタログ物理表名 (PT 名)	30 文字
DB2 UDB (OS/2 版) を使用する場合の最長の物理表名 (PT 名)	18 文字
DB2 UDB (OS/390 版) を使用する場合の最長の物理表名 (PT 名)	18 文字
UUI 特性値の最大長	254 バイト
5 つの UUI 特性値の最大合計長	1270 バイト
最長の情報カタログ・マネージャー・オブジェクト・タイプ・アイコン	30000 バイト
1 つのオブジェクト・タイプ内の最大特性数	255
1 つの情報カタログ・マネージャー・オブジェクト・タイプ内の LONG VARCHAR データ・タイプの特性の最大数	14
LONG VARCHAR 特性の探索基準の最大の長さ	3000 バイト
1 つのタグ言語ファイル内で ACTION.OBJTYPE() タグによって処理される固有オブジェクト・タイプの最大数	3500
以下の API 呼び出しで戻されるオブジェクトの最大数 FLGListAnchors FLGListOrphans	1600
以下の API 呼び出しで戻されるオブジェクトの最大数 FLGFoundIn FLGListAssociates FLGListContacts FLGListPrograms FLGNavigate FLGSearch FLGSearchAll FLGWhereUsed	5000

付録D. 情報カタログ・マネージャーの理由コード

表24には、情報カタログ・マネージャーによって作成されるすべての理由コードが示されています。各理由コードは番号順に列挙され、省略名、拡張コード、およびその理由コードが作成される条件が示されています。

理由コードの中には、拡張コードを作成して、エラー状況に関する補足情報を提供するものがあります。理由コードが拡張コードを戻す場合、その拡張コードが表す意味は次のとおりです。

表 24. 情報カタログ・マネージャーの理由コード

番号	理由コード	拡張コード	説明
0	FLG_OK	--	正常な完了。
1	FLG_WRN	--	プレースホルダー。警告の数値範囲の始まりを示します。
201	FLG_WRN_DISCONNECTED	--	データベースは切断されています。
202	FLG_WRN_DBM_ALREADY_STARTED	--	データベース・マネージャーは情報カタログ・マネージャーの初期設定以前に開始されています。
203	FLG_WRN_DB_RESTART	--	データベース・マネージャーは、情報カタログ・マネージャーの初期設定以前に再始動する必要があります。
204	FLG_WRN_DB_ACTIVE	--	指定されたデータベース・マネージャーは、情報カタログ・マネージャーの初期設定以前から活動状態になっています。

表 24. 情報カタログ・マネージャーの理由コード (続き)

番号	理由コード	拡張コード	説明
1001	FLG_WRN_INST_NOTFOUND	--	オブジェクト・インスタンスが見つかりませんでした (FLGListOrphans、FLGFoundIn、FLGListAssociates、および FLGExport でも使用)。
1002	FLG_WRN_CONTAINER_NOTFOUND	--	指定されたオブジェクト・インスタンスの収納用オブジェクトが見つかりませんでした。
1003	FLG_WRN_CONTAINEE_NOTFOUND	--	指定されたオブジェクト・インスタンスに含まれるオブジェクトが見つかりませんでした。
1004	FLG_WRN_CONTACT_NOTFOUND	--	指定されたオブジェクト・インスタンスの連絡担当者が見つかりませんでした。
1005	FLG_WRN_PROGRAM_NOTFOUND	--	このオブジェクト・タイプに関連付けられたプログラムが見つかりませんでした。
1006	FLG_WRN_ANCHOR_NOTFOUND	--	情報カタログ・マネージャー・データベースで定義されたアンカー (対象) が見つかりませんでした。
1007	FLG_WRN_PROGRAM_CHANGED	--	オブジェクト・タイプが削除されたときに、関連する 1 つまたは複数のプログラム・インスタンスが変更されました。

表 24. 情報カタログ・マネージャーの理由コード (続き)

番号	理由コード	拡張コード	説明
1008	FLG_WRN_NO_INPARM_ICON_FILE	--	検索されたアイコン・ファイルの名前を受け取るポインターが FLGGetReg API 呼び出しで指定されませんでした。情報カタログ・マネージャーはアイコンを戻しませんでした。
1009	FLG_WRN_NO_ICON	--	オブジェクト・タイプに関連するアイコンが戻されませんでした。
1010	FLG_WRN_ID_LIMIT_REACHED	--	オブジェクト・タイプの最大数に達しました。
1011	FLG_WRN_OBJECT_NOT_CHANGED	--	予約済
1012	FLG_WRN_EXCEED_MAX_ANCHORNUM	実際のアンカーの数	情報カタログ・マネージャー・データベースで定義されたすべてのアンカー (対象) を戻すことができませんでした。
1013	FLG_WRN_ICON_REPLACED	--	指定した ICOPATH にすでに存在するアイコン・ファイル。このアイコン・ファイルは置き換えられました。
1014	FLG_WRN_PROPDUP	--	追加された特性は、既に存在しています。
1015	FLG_WRN_EXCEED_MAX_ORPHANNUM	実際のオーファンの数	オーファンの最大数を超過しました。
1016	FLG_WRN_DB_ICON_REPLACED	--	オブジェクト・タイプ・アイコンがカタログ内で置換されました。

表 24. 情報カタログ・マネージャーの理由コード (続き)

番号	理由コード	拡張コード	説明
1017	FLG_WRN_LINKOBJ_NOTFOUND	--	指定されたオブジェクト・インスタンスについてリンクしているオブジェクトが見つかりませんでした。
1018	FLG_WRN_ATTACHOBJ_NOTFOUND	--	指定されたオブジェクトの付加オブジェクトが見つかりませんでした。
1019	FLG_WRN_MISSING_PROPS_IN_IOSTRUCT	--	入力構造に、オブジェクト・タイプに定義された特性よりも少ない特性が含まれています。欠落している特性はすべて任意選択です。オブジェクト・インスタンスが作成/更新されます。
2002	FLG_WRN_NO_DISKCNTL_TAG_PRESENTED	--	DISKCNTL は、取り外し可能装置の入力タグ言語ファイルにおける最初のタグではありません。インポートは続行されますが、現行ディスケットのタグ言語ファイルだけしか処理されません。
2003	FLG_WRN_NEED_NEW_TAGFILE_DISKETTE	--	タグ言語ファイルのインポートを続行するために次のディスケットを挿入してください。
2004	FLG_WRN_ICONFILE_OPENERR	--	予約済

表 24. 情報カタログ・マネージャーの理由コード (続き)

番号	理由コード	拡張コード	説明
2005	FLG_WRN_NOTHING_TO_IMPORT	--	タグ言語ファイルから、あるいはタグ言語ファイル内の最後のチェックポイント以降の部分から、インポートするデータが見つかりませんでした。ファイルまたはファイルの一部が空になっているか、あるいは COMMENT または DISKCNTL タグだけしか含まれていません。
2006	FLG_WRN_ICONFILE_RETRIEVE_ERROR	理由コード	pszIconFileID パラメーター内で指定されたアイコン・ファイルを検索中 (オープン、読み取り、またはクローズ中) に、API FLGCreateReg または FLGUpdateReg にエラーが発生しました。拡張コードに戻された理由コードがエラーを示しています。FLGCreateReg および FLGUpdateReg は、他の登録処理をすべて正常に完了しました。

表 24. 情報カタログ・マネージャーの理由コード (続き)

番号	理由コード	拡張コード	説明
2007	FLG_WRN_P_HANDLES_CLEARED	--	FLGImport は、プログラム・インスタンスの HANDLES 特性値が、ターゲット情報カタログに存在しないオブジェクト・タイプを参照しているため、その値を消去しました。
2501	FLG_WRN_CFLAG_IGNORED	--	エクスポートされるオブジェクトは Grouping 区分に属していないため、このオブジェクトの CONTAINEE-IND 値は無視されました。
2502	FLG_WRN_TFLAG_IGNORED	--	エクスポートされるオブジェクト値は Grouping または Elemental 区分に属していないため、このオブジェクトの CONTAINEE-IND 値は無視されました。
2503	FLG_WRN_NO_ICOPATH	--	アイコン・パスが指定されていません。アイコン・パスはエクスポートされませんでした。
2504	FLG_WRN_GETREG_WARNING	理由コード	エクスポート中に FLGGetReg から警告が出されました。拡張コードには FLGGetReg から戻された理由コードが示されています。

表 24. 情報カタログ・マネージャーの理由コード (続き)

番号	理由コード	拡張コード	説明
2505	FLG_WRN_GETINST_WARNING	理由コード	エクスポート中に FLGGetInst から警告が出されました。拡張コードには FLGGetInst から戻された理由コードが示されています。
2506	FLG_WRN_LISTCONTACTS_WARNING	理由コード	エクスポート中に FLGListContacts から警告が出されました。拡張コードには FLGListContacts から戻された理由コードが示されています。
2507	FLG_WRN_NAVIGATE_WARNING	理由コード	エクスポート中に FLGNavigate から警告が出されました。拡張コードには FLGNavigate から戻された理由コードが示されています。
2508	FLG_WRN_AFLAG_IGNORED	--	エクスポートされるオブジェクトが Attachment 区分にあり、関連した付加オブジェクトを持つことができないため、このオブジェクトの ATTACHMENT -IND 値が無視されました。
2509	FLG_WRN_LFLAG_IGNORED	--	エクスポートされるオブジェクトは Grouping または Elemental 区分に属していないため、このオブジェクトの LINK-IND 値は無視されました。

表 24. 情報カタログ・マネージャーの理由コード (続き)

番号	理由コード	拡張コード	説明
2601	FLG_WRN_NO_HISTORY	--	履歴バッファーに、履歴項目がありません。
2602	FLG_WRN_NO_TYPE_RELATE_TO_PROGRAM	--	プログラム・インスタンスに関連したオブジェクト・タイプがありません。
7500	FLG_WRN_VIEW_NOT_SUPPORTED	--	「ツール (Tool)」プロファイルでビュー ″T″ が指定されていますが、情報カタログ・マネージャーではこの機能はサポートされていません。
7501	FLG_WRN_LEVEL_NOT_SUPPORTED	--	「ツール (Tool)」プロファイルでレベル ″T″ が指定されていますが、情報カタログ・マネージャーではこの機能はサポートされていません。
7505	FLG_WRN_NO_BEGIN_DEFINITION_SECTION	--	タグ言語ファイルから BEGIN DEFINITION セクションが欠落しています。
7510	FLG_WRN_VALUE_TRUNCATED	--	最大許可長を超過したため、値は切り捨てられます。

表 24. 情報カタログ・マネージャーの理由コード (続き)

番号	理由コード	拡張コード	説明
7515	FLG_WRN_INV_TIMESTAMP_FORMAT	--	日付または時刻の値が正しい形式になっていません。 日付値の形式: YYYY-MM-DD。 時刻値の形式: HH.MM.SS。 最新表示日付値の形式: YYYY-MM-DD-HH.MM.SS。
30000	FLG_ERR	--	プレースホルダー。エラーの数値範囲の始まりを示します。
30001	FLG_ERR_INVALID_NUM_STR	--	入力として情報カタログ・マネージャーに渡された数値ストリングが無効です。
30002	FLG_ERR_INVALID_NUMBER	--	入力として情報カタログ・マネージャーに渡された整数値が長すぎます。
30003	FLG_ERR_BUFF_TOO_SMALL	--	情報カタログ・マネージャーの内部エラー。

表 24. 情報カタログ・マネージャーの理由コード (続き)

番号	理由コード	拡張コード	説明
30004	FLG_ERR_MSGFILE_NOTFOUND	--	情報カタログ・マネージャー・メッセージ・ファイル (DG*xy*MSG.MSG または DG*xy*STR.MSG。ただし、*x* はプラットフォーム識別子で、*y* は各国語版の識別子) が見つかりませんでした。 このファイルが情報カタログ・マネージャー作業ディレクトリーになければなりません。
30005	FLG_ERR_MSGID_NOTFOUND	--	メッセージ・ファイルからメッセージ識別子が見つかりませんでした。
30006	FLG_ERR_CANT_ACCESS_MSGFILE	--	情報カタログ・マネージャー・メッセージ・ファイルをオープンすることができませんでした。
30007	FLG_ERR_INVALID_MSGFILE_FORMAT	--	メッセージ・ファイル (DG*xy*MSG.MSG または DG*xy*STR.MSG。ただし、*x* はプラットフォーム識別子で、*y* は各国語版識別子) は破壊されたか、もしくは無効です。 影響を受けたファイルを再導入してください。

表 24. 情報カタログ・マネージャーの理由コード (続き)

番号	理由コード	拡張コード	説明
30008	FLG_ERR_MSGFILE_ERROR	--	情報カタログ・マネージャーの内部エラー。
30009	FLG_ERR_TRACE_FAIL	--	情報カタログ・マネージャー・トレース機能でエラーが発生しました。トレース・ファイルが破壊されているか、あるいは不完全です。
30010	FLG_ERR_INTERNAL_ERROR	理由コード	情報カタログ・マネージャーで内部エラーが検出されました。 拡張コードに戻された理由コードを調べ、問題の除去を試みてください。問題が除去できない場合には、IBM 技術員に連絡してください。
30011	FLG_ERR_RESDLL_NOT_LOADED	--	言語 DLL ファイルが見つかりません。
30012	FLG_ERR_DGPATH_NOT_FOUND	--	CONFIG.SYS ファイルで環境パス (DG2PATH) が設定されていません。 システム・レジストリーまたは AUTOEXEC.BAT ファイルで環境パス (DGWPATH) が設定されていません。

表 24. 情報カタログ・マネージャーの理由コード (続き)

番号	理由コード	拡張コード	説明
30013	FLG_ERR_CP_LOAD_FAILED	--	CONFIG.SYS ファイルで指定された 1 次および 2 次コード・ページは情報カタログ・マネージャーでサポートされません。
30014	FLG_ERR_DBSEM_ERROR	--	情報カタログ・マネージャーの内部エラー (データベース・セマフォーが入手不可能)
30015	FLG_ERR_STRINGFILE_ERROR	--	予約済
30016	FLG_ERR_MSG_TOO_LONG	--	情報カタログ・マネージャーの内部エラー。
30017	FLG_ERR_DG_DB_INUSE	--	ユーザーが同じ情報カタログ・マネージャー・データベースに 2 回ログオンしようとしました。
30018	FLG_ERR_DGLANG_PATH_NOT_FOUND	--	情報カタログ・マネージャー言語従属ディレクトリー・パスが見つかりません。
30019	FLG_ERR_INV_DG_CP	--	ワークステーションで指定されたコード・ページが、情報カタログ・マネージャーでサポートされていません。
30020	FLG_ERR_INV_DB_CP	--	ワークステーションで指定されたコード・ページが、データベースでサポートされていません。

表 24. 情報カタログ・マネージャーの理由コード (続き)

番号	理由コード	拡張コード	説明
30021	FLG_ERR_VWSPATH_NOT_FOUND	--	システム・レジストリーまたは AUTOEXEC.BAT ファイルで環境パス (VWSPATH) が設定されていません。
31000	FLG_ERR_DBERROR	データベース SQLCODE	予期しないデータベース・エラーが発生しました。SQLCODE の説明についてはデータベースの資料を参照してください。
31001	FLG_ERR_DBDISC_FAIL	--	データベースからの切断中にエラーが発生しました。
31002	FLG_ERR_NODBACCESS	--	指定した情報カタログ・マネージャー・データベースにアクセスすることができません。 管理者またはデータベース管理者から必要なデータベース権限を得てください。

表 24. 情報カタログ・マネージャーの理由コード (続き)

番号	理由コード	拡張コード	説明
31003	FLG_ERR_ID_LIMIT_EXCEEDED	--	システム生成の ID (オブジェクト・タイプ ID またはインスタンス ID) が情報カタログ・マネージャー・データベースで指定できる ID の最大数を超えています。 この限界値は、オブジェクト・インスタンス ID の場合は 99999999 であり、オブジェクト・タイプ ID の場合は 999999 です。
31004	FLG_ERR_PROP_LIMIT_EXCEEDED	--	オブジェクト・タイプに関して指定できる特性の最大数 (255) を超えています。
31005	FLG_ERR_LONG_VARCHAR_LIMIT_EXCEEDED	特性の順序番号	オブジェクト・タイプに関して指定できる LONG VARCHAR 特性の最大数 (14) を超えています。
31006	FLG_ERR_PTNAME_EXCEEDS_ENVSIZE	--	オブジェクト・タイプの物理タイプ名が許される最大長を超えています。この最大長は、使用しているデータベースによって異なります。

表 24. 情報カタログ・マネージャーの理由コード (続き)

番号	理由コード	拡張コード	説明
31007	FLG_ERR_DBNAME_NOT_FOUND	--	情報カタログ・マネージャー・データベースが見つかりませんでした。このデータベースがローカルである場合には、データベース名が見つかりませんでした。データベースがリモートである場合に、データベース名がローカル・データベース・ディレクトリーで定義されていませんでした。
31008	FLG_ERR_SRH_CRITERIA_TOOLONG	--	探索基準の全長が長すぎます。指定された探索基準すべての長さの合計最大長は、探索基準内の特性数によって異なりますが、約 32700 文字です。
31009	FLG_ERR_DB_TRANSLOG_FULL	--	データベース・トランザクション・ログがいっぱいです。 ただちに FLGCommit または FLGRollback を出してください。データベース・ログ・ファイルのサイズを大きくして、変更のコミットが必要になるまでに収納できる変更の数を増やしてください。

表 24. 情報カタログ・マネージャーの理由コード (続き)

番号	理由コード	拡張コード	説明
31010	FLG_ERR_INVALID_AUTHENTICATION	--	データベースが誤った確認オプションでカタログに登録されました。
31011	FLG_ERR_CHARCONV_WINTODBM	--	文字を Windows コード・ページからデータベース・コード・ページに変換している途中で、エラーが発生しました。
31012	FLG_ERR_DB_TIMEOUT	--	データベース・サーバーが使用中であるか、またはデッドロックされています。
31013	FLG_ERR_NOT_SUPPORTED_BY_DB	--	この機能は、データベース・サーバーでサポートされていません。
31014	FLG_ERR_DB_ICON_EXIST	--	InOptions パラメーターを FLG_ACTION_CREATE に設定した FLGManageIcons が呼び出されましたが、pszIconFileID で指定したアイコンはすでにデータベースに存在しています。 別のアイコン・ファイルを指定するか、FLG_ACTION_UPDATE を使用してください。
32000	FLG_ERR_REG_NOTEXIST	--	指定されたオブジェクト・タイプに関する登録情報が存在しません。

表 24. 情報カタログ・マネージャーの理由コード (続き)

番号	理由コード	拡張コード	説明
32001	FLG_ERR_TYPEID_NOTEXIST	--	指定されたオブジェクト・タイプに関する登録情報が存在しません。
32002	FLG_ERR_SRCTYPEID_NOTEXIST	--	指定されたソース・オブジェクト・タイプが存在しません。
32003	FLG_ERR_TRGTYPEID_NOTEXIST	--	指定されたターゲット・オブジェクト・タイプが存在しません。
32004	FLG_ERR_INSTID_NOTEXIST	--	指定されたオブジェクト ID (FLGID) が存在しません。
32005	FLG_ERR_SRCINSTID_NOTEXIST	--	指定されたソース・オブジェクト・タイプ ID (FLGID) が存在しません。
32006	FLG_ERR_TRGINSTID_NOTEXIST	--	指定されたターゲット・オブジェクト ID (FLGID) が存在しません。
32007	FLG_ERR_PROP_NOTEXIST	--	指定されたプログラムを開始することができません。プログラム・オブジェクトのパラメーター・リストで指定された特性はこのオブジェクト・インスタンスに関しては定義されていません。
32008	FLG_ERR_REL_NOTEXIST	--	この関係は、存在しないため、削除することができません。

表 24. 情報カタログ・マネージャーの理由コード (続き)

番号	理由コード	拡張コード	説明
32009	FLG_ERR_TYPE_NOT_CREATED	--	指定されたオブジェクト・タイプは、登録されていますが、作成されていません。
32010	FLG_ERR_SRCTYPE_NOT_CREATED	--	ソース・オブジェクト・インスタンスの FLGID に指定したオブジェクト・タイプは、登録されていますが、作成されていません。
32011	FLG_ERR_TRGTYPE_NOT_CREATED	--	ターゲット・オブジェクト・インスタンスの FLGID に指定したオブジェクト・タイプは、登録されていますが、作成されていません。
32012	FLG_ERR_INV_P_CATEGORY	--	オブジェクト・タイプを作成または削除するときには、P (プログラム) は無効な値です。 Program 区分のオブジェクト・タイプを作成または削除することはできません。
32013	FLG_ERR_INV_P_HANDLE_CAT	--	プログラム・オブジェクト・インスタンスの操作点特性値が無効です。 この値は、Program 以外のオブジェクト・タイプの名前でなければなりません。

表 24. 情報カタログ・マネージャーの理由コード (続き)

番号	理由コード	拡張コード	説明
32014	FLG_ERR_P_HANDLE_NOTEXIST	--	プログラム・オブジェクト・インスタンスの操作点特性値が無効です。指定されたオブジェクト・タイプが存在しません。
32015	FLG_ERR_P_HANDLE_NOT_CREATED	--	プログラム・オブジェクト・インスタンスの操作点特性値が無効です。指定されたオブジェクト・タイプは、登録されていますが、作成されていません。
32016	FLG_ERR_INV_A_CATEGORY	--	A (アタッチメント) は、オブジェクト・タイプの作成、削除、または追加をする際に、その区分に対して無効の値です。Attachment 区分のオブジェクト・タイプの作成、削除、または追加はできません。
32300	FLG_ERR_REG_DUP	--	オブジェクト・タイプを登録することができません。指定されたオブジェクト・タイプは、すでに登録されています。

表 24. 情報カタログ・マネージャーの理由コード (続き)

番号	理由コード	拡張コード	説明
32301	FLG_ERR_TYPE_DUP	--	指定された名前のオブジェクト・タイプを作成することができません。指定されたオブジェクト・タイプは、情報カタログ・マネージャー・データベースにすでに存在しています。
32302	FLG_ERR_INST_DUP	--	指定されたオブジェクト・インスタンスを作成することができません。情報カタログ・マネージャー・データベースには、同じ UUI 特性値のオブジェクト・インスタンスがすでに含まれています。
32303	FLG_ERR_REL_DUP	--	指定されたオブジェクト関係を作成することができません。この関係はすでに存在しています。
32304	FLG_ERR_REL_RECURSIVE	--	指定された関係を作成することができません。指定された関係を作成すると、Grouping オブジェクトがそれ自体を含むようになります。
32305	FLG_ERR_UUI_DUP	UUI 順序番号が重複している特性の順序番号。	このオブジェクト・タイプ、またはオブジェクトの定義には、同じ UUI 順序番号の特性が 2 つ以上含まれています。

表 24. 情報カタログ・マネージャーの理由コード (続き)

番号	理由コード	拡張コード	説明
32306	FLG_ERR_INVALID_LINK_RELATION	--	指定されたリンクについて、リンクする側とされる側が同じなので、このリンク関係は無効です。
32307	FLG_ERR_INVALID_ATTACHMENT_RELATION	--	ターゲット・オブジェクトが既にほかのアタッチメント以外のソース・オブジェクトと関連しているので、このアタッチメント関係は拒否されました。Attachment 区分オブジェクトは、Attachment 以外ただ 1 つの区分ソース・オブジェクトとしか関連できません。
32308	FLG_ERR_ICONFILE_RETRIEVE_ERROR	理由コード	pszIconFileID パラメーター内で指定されたアイコン・ファイルを検索中 (オープン、読み取り、またはクローズ中) に、API FLGManageIcons にエラーが発生しました。これは、入力オプションの FLG_ACTION_CREATE または FLG_ACTION_UPDATE のみに適用されます。拡張コードに戻された理由コードがエラーを示しています。処理は失敗しました。

表 24. 情報カタログ・マネージャーの理由コード (続き)

番号	理由コード	拡張コード	説明
32400	FLG_ERR_CONTAINEE_EXIST	--	この Grouping オブジェクト・インスタンスは 1 つまたは複数のオブジェクト・インスタンスを含んでいるため、削除することはできません。この関係または含まれているオブジェクトを削除してからでなければ、このオブジェクト・インスタンスを削除することはできません。
32401	FLG_ERR_INST_EXIST	--	指定されたオブジェクト・タイプのインスタンスが存在しているため、このオブジェクト・タイプを削除することはできません。このオブジェクト・タイプのすべてのインスタンスを削除してからでなければ、オブジェクト・タイプは削除できません。
32402	FLG_ERR_TYPE_EXIST	--	オブジェクト・タイプが存在しているため、このオブジェクト・タイプ登録を削除することはできません。オブジェクト・タイプを削除してからでなければ、このオブジェクト・タイプ登録は削除できません。

表 24. 情報カタログ・マネージャーの理由コード (続き)

番号	理由コード	拡張コード	説明
32403	FLG_ERR_CONTAINEE_DIFFTYPE	--	異なったオブジェクト・タイプに属する非包含が見つかったため、FLGDeleteTypeExt API が停止しました。
32500	FLG_ERR_INVALID_SRCCAT	--	指定された関係を作成することができません。ソース・オブジェクト・タイプの区分が無効です。
32501	FLG_ERR_INVALID_TRGCAT	--	指定された関係を作成することができません。ターゲット・オブジェクト・タイプの区分が無効です。
32502	FLG_ERR_INVALID_CAT	--	入力オブジェクト・タイプの区分が正しくありません。 必要な入力オブジェクト・タイプについては、呼び出した API に関する特定の資料を参照してください。
32600	FLG_ERR_KAEXIST	--	管理者としてログオンすることができません。別の管理者がすでにログオンしています。情報カタログ・マネージャーでは、一度に 1 人の管理者だけしかログオンすることができません。

表 24. 情報カタログ・マネージャーの理由コード (続き)

番号	理由コード	拡張コード	説明
32601	FLG_ERR_NOTAUTH	--	現行ユーザー ID では、この情報カタログ・マネージャー機能の使用が許可されていません。
32602	FLG_ERR_NOT_INITIALIZED	--	情報カタログ・マネージャーが初期設定されていません。 FLGInit を出してからでなければ、情報カタログ・マネージャーはほかの機能を実行できません。
32603	FLG_ERR_ALREADY_INITIALIZED	--	情報カタログ・マネージャーはすでに初期設定されています。 FLGTerm 呼び出しを出してからでなければ、2 番目の FLGInit 呼び出しを出すことはできません。
32604	FLG_ERR_NOT_CREATOR	--	自分で作成していないコメント・オブジェクトを更新する権限はありません。
32700	FLG_ERR_INVALID_TYPEID	--	指定されたオブジェクト・タイプ ID (OBJTYPID) が無効です。
32701	FLG_ERR_INVALID_TYPEID_LEN	--	指定されたオブジェクト・タイプ ID (OBJTYPID) が無効です。この値の長さは 6 バイトでなければなりません。

表 24. 情報カタログ・マネージャーの理由コード (続き)

番号	理由コード	拡張コード	説明
32702	FLG_ERR_INVALID_TYPEID_VAL	--	指定されたオブジェクト・タイプ ID (OBJTYPID) の値が無効です。
32703	FLG_ERR_INVALID_FLGID	エクスポートされたオブジェクトの数、またはパラメーターの位置	指定されたオブジェクト ID (FLGID) が無効です。
32704	FLG_ERR_INVALID_FLGID_LEN	--	オブジェクト ID (FLGID) が無効です。この値の長さは 16 バイトでなければなりません。
32705	FLG_ERR_INVALID_FLGID_VAL	--	オブジェクト ID (FLGID) に無効文字が含まれています。
32706	FLG_ERR_INVALID_TYPNM	--	オブジェクト・タイプ名が無効です。
32707	FLG_ERR_INVALID_INSTNM	--	オブジェクト・インスタンス名が無効です。
32708	FLG_ERR_INVALID_TIMESTAMP	特性の順序番号	入力値が無効です。入力値は YYYY-MM-DD-HH.MM.SS.NNNNNN の形式で 26 バイト長のタイム・スタンプでなければなりません。
32709	FLG_ERR_INVALID_SRCID	--	ソース・オブジェクト ID (FLGID) が無効です。
32710	FLG_ERR_INVALID_TRGID	--	ターゲット・オブジェクト ID (FLGID) が無効です。

表 24. 情報カタログ・マネージャーの理由コード (続き)

番号	理由コード	拡張コード	説明
32711	FLG_ERR_INVALID_RELTYPE	--	指定された関係タイプ (RelType) が無効です。有効な値は、C、T、A、または L です。
32712	FLG_ERR_INVALID_RELOPT	--	指定された関係オプション (RelOpt) が無効です。有効な値は、C または D です。
32713	FLG_ERR_INVALID_PGM_FLGID	--	プログラム・オブジェクトに指定されたオブジェクト ID (FLGID) が無効です。
32714	FLG_ERR_INVALID_OBJ_FLGID	--	FLGOpen 呼び出しのパラメーターを提供するオブジェクトに指定されたオブジェクト ID (FLGID) が無効です。
32718	FLG_ERR_INVALID_USERID	--	ユーザー ID が無効です。この長さは 1 文字から 8 文字までにする必要があります。 ユーザー ID/パスワードが無効です (AIX® 上では、パスワードには大文字小文字の区別があります)。 ユーザーがリモート・ノード (DB2 (OS/2 版) V2.1) にログオンしていません。

表 24. 情報カタログ・マネージャーの理由コード (続き)

番号	理由コード	拡張コード	説明
32719	FLG_ERR_INVALID_PASSWORD	--	指定されたパスワードが無効です。この長さは 1 文字から 8 文字までにする必要があります。
32720	FLG_ERR_INVALID_DBNAME	--	指定された情報カタログ・マネージャー・データベース名が無効です。この長さは 1 文字から 8 文字までにする必要があります。
32721	FLG_ERR_INVALID_ADMINOPT	--	指定されたユーザー・オプション (admin) が無効です。有効な値は、Y と N です。
32722	FLG_ERR_INVALID_TRACEOPT	--	トレース・オプション (TraceOpt) が無効です。有効なオプションは、0、1、2、3、および 4 です。
32723	FLG_ERR_NULL_PARAMETER	パラメーターの位置	この API 呼び出しの入力として必要なパラメーターが抜けているか、あるいは NULL になっています。拡張コードに NULL パラメーターの位置が示されています。
32724	FLG_ERR_NULL_EXTCODE	--	拡張コード・ポインター・パラメーター (pExtCode) が NULL になっています。

表 24. 情報カタログ・マネージャーの理由コード (続き)

番号	理由コード	拡張コード	説明
32725	FLG_ERR_INVALID_CONVERTOPT	--	指定された入力オプション (Options) が無効です。有効な値は、D または F です。
32726	FLG_ERR_INVALID_ICONOPT	--	指定された入力オプション (Options) が、FLGManageIcons に対して無効です。
32727	FLG_ERR_INVALID_TAGBUFOPT	--	FLGManageTagBuf API に対して指定された InOptions が無効です。DGxAPI.H ファイルで定義された、FLG_TAGBUF_QUERY または FLG_TAGBUF_RESET を使用してください。
32728	FLG_ERR_INVALID_TAGFILEOPT	--	FLGXferTagBuf API に対して指定されたオプション・パラメーターが無効です。DGxAPI.H ファイルで定義された、FLG_TAGOPT_NEW または FLG_TAGOPT_REPLACE を使用してください。

表 24. 情報カタログ・マネージャーの理由コード (続き)

番号	理由コード	拡張コード	説明
32729	FLG_ERR_INV_DGFLAG_ACTION	--	FLGManageFlags に対して指定されたアクション・パラメーターが無効です。 DGxAPI.H ファイルで定義された FLG_ACTION_GET または FLG_ACTION_UPDATE を使用してください。
32730	FLG_ERR_INV_DGFLAG_FLAGTYPE	--	FLGManageFlags API に対して指定された FlagType パラメーターが無効です。 DGxAPI.H ファイルで定義された FLG_HISTORY_TYPE_DELETE を使用してください。
32731	FLG_ERR_INV_DGFLAG_VALUE	--	FLGManageFlags に対して指定された chValue パラメーターが無効です。有効な値は、FLG_YES または FLG_NO です。
32732	FLG_ERR_INV_STATUS_ACTION	--	FLGManageCommentStatus API に対して指定されたアクション・パラメーターが無効です。 DGxAPI.H ファイルで定義された FLG_ACTION_UPDATE または FLG_ACTION_GET を使用してください。

表 24. 情報カタログ・マネージャーの理由コード (続き)

番号	理由コード	拡張コード	説明
32733	FLG_ERR_INV_STATUS_LEN	特性の順序番号	入力構造オブジェクト域に、80 文字より長い状況フィールドが含まれています。
32734	FLG_ERR_INVALID_TREEOPT	--	FLGDeleteTree API に対して指定されたオプション・パラメーターが無効です。 DGxAPI.H ファイルで定義された、FLG_DELTREE_REL または FLG_DELTREE_ALL を使用してください。
32735	FLG_ERR_INVALID_ASSOCOPT	--	FLGListAssociates API に対して指定されたオプション・パラメーターが無効です。 DGxAPI.H ファイルで定義された、FLG_LIST_PROGRAM、FLG_LIST_ATTACHMENT、FLG_LIST_COMMENTS、FLG_LIST_CONTAIN、FLG_LIST_CONTACT または FLG_LIST_LINK を使用してください。

表 24. 情報カタログ・マネージャーの理由コード (続き)

番号	理由コード	拡張コード	説明
32736	FLG_ERR_INVALID_ORPHANOPT	--	FLGListOrphans API に対して指定されたオプション・パラメーターが無効です。 DGxAPI.H ファイルで定義された、FLG_LIST_PROGRAM、FLG_LIST_CONTACT、FLG_LIST_ATTACHMENT または FLG_LIST_COMMENTS を使用してください。
32737	FLG_ERR_INVALID_FOUNDINOPT	--	FLGFoundIn API に対して指定されたオプション・パラメーターが無効です。 DGxAPI.H ファイルで定義された、FLG_LIST_PROGRAM、FLG_LIST_CONTAIN、 FLG_LIST_CONTACT または FLG_LIST_ATTACHMENT を使用してください。
33000	FLG_ERR_ICON_NOTEXIST	--	指定されたアイコン・ファイルが存在しません。
34000	FLG_ERR_INVALID_IOSTRUCT	--	入力構造が無効です。定義域の長さまたはオブジェクト域の長さが、記述される区域の長さと一致していません。
34001	FLG_ERR_NO_DEFN_AREA	--	入力構造で定義域が欠落しています。

表 24. 情報カタログ・マネージャーの理由コード (続き)

番号	理由コード	拡張コード	説明
34002	FLG_ERR_NO_OBJ_AREA	--	入力構造でオブジェクト域が欠落しています。
34003	FLG_ERR_INVALID_POSITION	--	情報カタログ・マネージャーの内部エラー。
34004	FLG_ERR_IOSTRUCT_CONVERSION	--	入力構造の読み取りまたは出力構造の書き込み中に情報カタログ・マネージャーの内部エラーが発生しました。
34005	FLG_ERR_INVALID_IOSTRUCT_NULL	バイト・オフセット	入力構造に NULL 文字が含まれています。
34006	FLG_ERR_OBJLEN_OBJCNT_MISMATCH	--	オブジェクト域の項目カウントまたは長さがゼロになっています。 一方の値がゼロよりも大きい場合には、他方の値をゼロにすることはできません。
34200	FLG_ERR_INV_HEADER_IDENT	--	入力構造ヘッダー域に指定した識別子が無効です。 識別子は FLG-HEAD でなければなりません。

表 24. 情報カタログ・マネージャーの理由コード (続き)

番号	理由コード	拡張コード	説明
34201	FLG_ERR_INV_HEADER_DEFLEN	--	入力構造ヘッダー域の定義の長さが無効です。 定義の長さは、ゼロよりも大きな 160 の倍数でなければなりません。API 呼び出しの中には、定義の長さが固定長でなければならないものがあります。必要な定義の長さについては、API 呼び出しの構文を参照してください。
34202	FLG_ERR_INV_HEADER_DEFCNT	--	ヘッダー域の定義の長さから予測される定義の数が、FLGExport 用の値としては無効です。 定義の数は、FLGExport の場合には 5 でなければなりません。したがって、定義の長さは 800 でなければなりません。
34203	FLG_ERR_INV_HEADER_OBJLEN	--	入力構造ヘッダー域のオブジェクトの長さが無効です。
34204	FLG_ERR_INV_HEADER_OBJCNT	--	入力構造ヘッダー域に指定したオブジェクト域の項目カウントが無効です。

表 24. 情報カタログ・マネージャーの理由コード (続き)

番号	理由コード	拡張コード	説明
34205	FLG_ERR_INV_HEADER_CATEGORY	--	ヘッダー域に指定された区分が無効です。 FLGCreateReg の場合、区分の値は G、E、C、D、または S のいずれかでなければなりません。 FLGCreateType、FLGCreateInst、FLGUpdateReg、FLGAppendType、および FLGUpdateInst の場合、区分の値は、関連するオブジェクト・タイプ登録の値と一致していなければなりません。
34206	FLG_ERR_INV_HEADER_OBJTYPEID	--	ヘッダー域のオブジェクト・タイプの値が無効です。 この値は、関連するオブジェクト・タイプ登録に関して生成されたオブジェクト・タイプ ID と同じでなければなりません。

表 24. 情報カタログ・マネージャーの理由コード (続き)

番号	理由コード	拡張コード	説明
34207	FLG_ERR_CONFLICTING_HEADER_FIELDS	--	定義の長さから求められた特性の数が、ヘッダー域のオブジェクト域項目カウントと矛盾しています。 特性の数は定義域の長さを 160 で割った値に等しく、また、オブジェクト域項目カウントは特性の数で割り切れなければなりません。
34208	FLG_ERR_CONFLICTING_OBJTYPID	特性の順序番号	オブジェクト域でオブジェクト・タイプ識別子 (OBJTYPID) に指定された値が、ヘッダー域のオブジェクト・タイプ ID と一致していません。
34209	FLG_ERR_HEADER_DEFLEN_EXCEEDS_MAX	--	ヘッダー域の定義の長さで、特性の数が最大数を超えます。
34210	FLG_ERR_NONBLANK_HEADER_CATEGORY	--	ヘッダー域の区分値が無効です。
34211	FLG_ERR_NONBLANK_HEADER_OBJTYPEID	--	ヘッダー域のオブジェクト・タイプ ID の値が無効です。
34222	FLG_ERR_NONBLANK_HEADER_RESERVED	--	入力構造ヘッダー域の予約域は、常にブランクでなければなりません。
34500	FLG_ERR_INV_PROPERTY_NAME	特性の順序番号	指定された特性名は、この API 呼び出しで必要な特性名の 1 つではありません。

表 24. 情報カタログ・マネージャーの理由コード (続き)

番号	理由コード	拡張コード	説明
34501	FLG_ERR_INV_PROPERTY_PPNAME	特性の順序番号	定義域で指定された特性の特性省略名が無効です。値が欠落しているか、DBCS 文字を使用しているか、あるいは API 呼び出しで必要とされる値を使用していないかのいずれかです。
34502	FLG_ERR_INV_PROPERTY_DATATYPE	特性の順序番号	定義域に指定した特性のデータ・タイプが無効です。 有効な値は、その API 呼び出しに応じて、CHAR、TIMESTAMP、VARCHAR、または LONG VARCHAR のいずれかになります。
34503	FLG_ERR_INV_PROPERTY_V_FLAG	特性の順序番号	定義域における表示された特性の値フラグが無効です。 有効な値は、R、O、または S です。
34504	FLG_ERR_INV_PROPERTY_SVALUE_V_FLAG	特性の順序番号	定義域における表示された特性の値フラグが無効です。指定された値フラグは S になっていますが、情報カタログ・マネージャーは特性省略名で示された特性を生成しません。

表 24. 情報カタログ・マネージャーの理由コード (続き)

番号	理由コード	拡張コード	説明
34505	FLG_ERR_INV_PROPERTY_CS_FLAG	特性の順序番号	定義域における表示された特性の大文字小文字の区別フラグ値が無効です。 有効な値は、Y または N です。
34506	FLG_ERR_INV_PROPERTY_FS_FLAG	特性の順序番号	定義域における表示された特性のファジー探索フラグ値が無効です。 有効な値は、Y または N です。
34507	FLG_ERR_INV_PROPERTY_UUISEQ	特性の順序番号	定義域における表示された特性の UUI 順序が無効です。 有効な値は、1、2、3、4、5、またはブランクです。
34508	FLG_ERR_INV_PROPERTY_LEN_FOR_DTYPE	特性の順序番号	定義されたデータ・タイプとの関係で、定義域における表示された特性の長さ値が無効です。
34509	FLG_ERR_INV_PROP_LEN_FIELD	特性の順序番号	定義域における表示された特性の長さが無効です。 API 呼び出しの構文を参照して必要な長さを調べてください。

表 24. 情報カタログ・マネージャーの理由コード (続き)

番号	理由コード	拡張コード	説明
34510	FLG_ERR_INV_PROP_VAL_LEN	--	オブジェクト域に指定された VARCHAR または LONG VARCHAR 特性値の長さフィールドが無効です。このフィールドには、数字を右寄せして指定しなければなりません。
34511	FLG_ERR_INV_RQDPROP_SPEC	特性の順序番号	定義域の特性定義で、必須特性の定義に必要な 1 つまたは複数のフィールドが無効です。 どの特性についても、API 呼び出しの入力構造ダイアグラムに示されたとおりに以下のフィールドを指定する必要があります。 • 特性の名前 (バイト 0-79) • データ・タイプ (バイト 80-109) • 長さ (バイト 110-117) • 特性の省略名 (バイト 118-125) • 値フラグ (バイト 126) • UUI の順序番号 (バイト 127)

表 24. 情報カタログ・マネージャーの理由コード (続き)

番号	理由コード	拡張コード	説明
34512	FLG_ERR_DUP_PROPERTY_NAME	特性の順序番号	入力構造内の別の特性ですでにこの特性名が使用されています。各特性名は入力構造内で固有でなければなりません。
34513	FLG_ERR_DUP_PROPERTY_PPNAME	特性の順序番号	指定された特性の特性省略名が、この入力構造内の別の特性の特性省略名と同じになっています。各特性省略名は入力構造内で固有でなければなりません。
34514	FLG_ERR_INV_TOT_UUI_LEN	--	予約済
34515	FLG_ERR_INV_UUI_LENGTH	UUI の順序番号	定義域で指定された UUI 特性の長さ値が、UUI 特性の最大長を超えています。
34516	FLG_ERR_MISSING_PROPERTY	--	オブジェクト・インスタンスの定義域に、オブジェクト・タイプに関して定義されているすべての特性が含まれていません。
34517	FLG_ERR_MISSING_PROPERTY_NAME	特性の順序番号	定義域に指定された特性に関して特性名を指定する必要がありますが、これが欠落しています。
34518	FLG_ERR_MISSING_PROPERTY_LENGTH	特性の順序番号	定義域に指定された特性に関して長さ値を指定する必要がありますが、これが欠落しています。

表 24. 情報カタログ・マネージャーの理由コード (続き)

番号	理由コード	拡張コード	説明
34519	FLG_ERR_MISSING_PROPERTY_PPNAME	特性の順序番号	定義域に指定された特性に関して特性省略名を指定する必要がありますが、これが欠落しています。
34520	FLG_ERR_MISSING_REG_DPNAME	--	入力構造定義域に DP 名 (DPNAME) 特性を指定する必要がありますが、これが欠落しています。
34521	FLG_ERR_MISSING_REG_PTNAME	--	入力構造定義域に物理タイプ名 (PTNAME) 特性を指定する必要がありますが、これが欠落しています。
34522	FLG_ERR_MISSING_REG_CREATOR	--	入力構造定義域に作成者特性を指定する必要がありますが、これが欠落しています。
34523	FLG_ERR_MISSING_REG_UPDATIME	--	入力構造定義域に最終変更日付および時刻 (UPDATIME) 特性を指定する必要がありますが、これが欠落しています。
34524	FLG_ERR_MISSING_REG_UPDATEBY	--	入力構造定義域に最終変更者 (UPDATEBY) 特性を指定する必要がありますが、これが欠落しています。

表 24. 情報カタログ・マネージャーの理由コード (続き)

番号	理由コード	拡張コード	説明
34525	FLG_ERR_MISSING_REG_NAME	--	入力構造定義域にオブジェクト・タイプの外部名 (NAME) 特性を指定する必要がありますが、これが欠落しています。
34526	FLG_ERR_MISSING_UUI_SEQUENCE	--	表示された UUI 順序番号が定義域で指定されましたが、先行する番号が指定されていません。 UUI 順序番号は、連続していなければなりません。たとえば、1、2、3 は有効ですが、1、3、5 は無効です。
34527	FLG_ERR_MISSING_RQD_INSTIDNT	--	入力構造定義域にインスタンス ID (INSTIDNT) 特性を指定する必要がありますが、これが欠落しています。
34528	FLG_ERR_MISSING_RQD_NAME	--	入力構造定義域に名前 (NAME) 特性を指定する必要がありますが、これが欠落しています。
34529	FLG_ERR_MISSING_RQD_OBJTYPID	--	入力構造定義域にオブジェクト・タイプ識別子 (OBJTYPID) 特性を指定する必要がありますが、これが欠落しています。

表 24. 情報カタログ・マネージャーの理由コード (続き)

番号	理由コード	拡張コード	説明
34530	FLG_ERR_MISSING_RQD_UPDATEBY	--	入力構造定義域に最終変更者 (UPDATEBY) 特性を指定する必要がありますが、これが欠落しています。
34531	FLG_ERR_MISSING_RQD_UPDATIME	--	入力構造定義域に最終変更日付および時刻 (UPDATIME) 特性を指定する必要がありますが、これが欠落しています。
34532	FLG_ERR_NOMATCH_PROPERTY_NAME	特性の順序番号	定義域における表示された入力特性は既存特性の特性省略名と一致していますが、特性名が一致していません。
34533	FLG_ERR_NOMATCH_PROPERTY_SPEC	特性の順序番号	定義域における表示された特性が既存特性の特性名および特性省略名と一致していますが、データ・タイプ、長さ、値フラグ、または UUI 順序の値が一致していません。
34534	FLG_ERR_PROPERTY_NOTEXIST	特性の順序番号	選択基準の一部として指定された特性が存在していません。

表 24. 情報カタログ・マネージャーの理由コード (続き)

番号	理由コード	拡張コード	説明
34536	FLG_ERR_UNMATCH_DEFINITION	特性の順序番号	次のいずれかが起こりました。 • オブジェクト・インスタンスの定義域に指定された、表示されている特性が、オブジェクト・タイプに関して定義されたどの特性とも一致しません。 • オブジェクト・インスタンスについて定義域で定義されている特性が、そのオブジェクト・タイプに関して定義されている特性よりも多くなっています。
34537	FLG_ERR_PROPDUP	--	定義域で特性名または特性省略名が重複して指定されています。
34538	FLG_ERR_REG_PROPS_OUT_OF_SEQUENCE	--	登録特性が正しい順序で指定されていません。
34539	FLG_ERR_RQD_PROPS_OUT_OF_SEQUENCE	--	定義域で必須特性が正しい順序で指定されていません。
34540	FLG_ERR_INV_V_FLAG_FOR_APPEND	特性の順序番号	表示された追加特性の値フラグが S または R になっています。 追加特性の値フラグは ″O″ (任意選択特性) でなければなりません。

表 24. 情報カタログ・マネージャーの理由コード (続き)

番号	理由コード	拡張コード	説明
34541	FLG_ERR_INV_UUI_FOR_APPEND	特性の順序番号	表示された追加特性は UUI 特性として指定されています。追加特性は UUI 特性にすることができません。
34542	FLG_ERR_NONBLANK_PROPERTY_V_FLAG	特性の順序番号	表示された特性の値フラグがブランクになっていません。値フラグは、この API 呼び出しでは使用されないため、ブランクのままにする必要があります。
34543	FLG_ERR_NONBLANK_PROPERTY_CS_FLAG	特性の順序番号	表示された特性に関する大文字小文字の区別フラグがブランクになっていません。大文字小文字の区別フラグは、この API 呼び出しでは使用されないため、ブランクのままにする必要があります。
34544	FLG_ERR_NONBLANK_PROPERTY_FS_FLAG	特性の順序番号	表示された特性に関するファジー探索フラグがブランクになっていません。ファジー探索フラグは、この API 呼び出しでは使用されないため、ブランクのままにする必要があります。

表 24. 情報カタログ・マネージャーの理由コード (続き)

番号	理由コード	拡張コード	説明
34545	FLG_ERR_NONBLANK_PROPERTY_UUISEQ	特性の順序番号	表示された特性に関する UUI 順序位置がブランクになっていません。 UUI 順序位置は、この API では使用されないため、ブランクのままにする必要があります。 データ・タイプは LONG VARCHAR ですが、UUI 順序位置がブランクになっていません。UUI 特性には、CHAR、VARCHAR、TIMESTAMP を使えますが、LONG VARCHAR は使えません。
34546	FLG_ERR_NONBLANK_PROPERTY_RESERVED	特性の順序番号	入力構造特性を指定する予約域は、常にブランクでなければなりません。
34547	FLG_ERR_UUI_V_FLAG_MUST_BE_R	特性の順序番号	すべての UUI 特性の値フラグは R (必須) でなければならないため、表示された特性の値フラグは無効です。

表 24. 情報カタログ・マネージャーの理由コード (続き)

番号	理由コード	拡張コード	説明
34548	FLG_ERR_AT_LEAST_ONE_UUI_PROP_RQD	--	定義域で指定された特性のうちで、UUI 特性として定義されているものがありません。 各情報カタログ・マネージャー・オブジェクト・タイプの定義では、少なくとも 1 つは UUI 特性が指定されていなければなりません。
34550	FLG_ERR_DUP_REG_DPNAME	--	定義域で指定された DP 名 (DPNAME) が、既存のオブジェクト・タイプ登録の DP 名の値と重複しています。 DPNAME 値はすべての情報カタログ・マネージャー・データベースを通じて固有でなければなりません。
34551	FLG_ERR_DUP_REG_PTNAME	--	物理タイプ名 (PTNAME) がデータベース内の既存の表の名前と重複しています。 PTNAME 値はすべての情報カタログ・マネージャー・データベースを通じて固有でなければなりません。

表 24. 情報カタログ・マネージャーの理由コード (続き)

番号	理由コード	拡張コード	説明
34552	FLG_ERR_DUP_REG_NAME	--	指定されたオブジェクト・タイプの外部名 (NAME) が既存のオブジェクト・タイプ登録の NAME 値と重複しています。 NAME はすべての情報カタログ・マネージャー・データベースを通じて固有でなければなりません。
34553	FLG_ERR_INV_DPNAME	--	指定された DPNAME 値の構文が無効です。
34554	FLG_ERR_INV_DB_PTNAME	--	指定された PTNAME 値は、データベースの構文規則との関係で無効です。
34555	FLG_ERR_INV_DB_DPNAME	--	予約済
34556	FLG_ERR_INV_DB_PROPERTY_PPNAME	--	特性省略名は、データベースの構文規則との関係で無効です。
34557	FLG_ERR_INV_TOT_PROPERTY_LEN	--	CHAR、VARCHAR、および TIMESTAMP 特性とオーバーヘッドの全長が、データベース内の物理テーブルの各行の最大長としてデータベースで許されている長さを超えています。
34558	FLG_ERR_INV_PTNAME	--	指定された PTNAME 値の構文が無効です。

表 24. 情報カタログ・マネージャーの理由コード (続き)

番号	理由コード	拡張コード	説明
34559	FLG_ERR_INV_PROPERTY_CS_FLAG_FOR_DB	特性の順序番号	大文字小文字の区別フラグの値が、このデータベースでは無効です。
34560	FLG_ERR_SRH_PROP_VAL_TOOLONG	特性の順序番号	探索基準値が長すぎます。 OS/390 上で DB2 を使用する場合の最大長は 254 バイトです。
34561	FLG_ERR_EXTRA_PROPS_IN_IOSTRUCT	--	入力構造に、オブジェクト・タイプ定義にない特性が 1 つ以上含まれています。
34562	FLG_ERR_MISSING_REQ_PROPERTY	特性の順序番号	必要な特性が FLGCreateInst または FLGUpdateInst API の入力構造から欠落しています。拡張コードが、オブジェクト・タイプの完全な定義を使用して、欠落した特性の位置を指しています。
34800	FLG_ERR_PROP_VALUE_REQUIRED	特性の順序番号	表示された特性に関する値がオブジェクト域に指定されていません。この特性の定義に従って、値を指定することが必要です。
34801	FLG_ERR_PROP_VALUE_EXCEEDED	特性の順序番号	表示された特性の値の長さが、定義域で定義された最大長を超えています。

表 24. 情報カタログ・マネージャーの理由コード (続き)

番号	理由コード	拡張コード	説明
34802	FLG_ERR_INVALID_PROPERTY_VALUE	特性の順序番号	次のいずれかの理由により、特性値が無効です。 • 値に DBCS 文字が使用されていますが、SBCS 文字を使用しなければなりません。 • FLGUpdateInst を使用する場合、オブジェクト域の INSTIDNT 値が無効です。
34803	FLG_ERR_INV_SRH_VAL_FOR_LONGVARCHAR	特性の順序番号	表示された特性に関する探索値が、LONG VARCHAR データ・タイプの探索値に許される最大長 (3000) を超えています。
34804	FLG_ERR_INV_OBJ_LENGTH	--	オブジェクト域の実際の長さが、ヘッダー域で指定されたオブジェクト長と一致していません。
34805	FLG_ERR_PARMLIST_REQUIRES_HANDLES	特性の順序番号	操作点特性が定義域で指定されていません。
34806	FLG_ERR_REG_CONFLICT	--	オブジェクト域に指定された DPNAME 値または PTNAME 値が、オブジェクト・タイプ ID で識別された登録情報の値と一致していません。

表 24. 情報カタログ・マネージャーの理由コード (続き)

番号	理由コード	拡張コード	説明
34807	FLG_ERR_ICON_EXCEEDS_LIMIT	--	アイコン・サイズが最大アイコン・サイズ (30000) を超えています。
34808	FLG_ERR_INST_VALUE_EXCEEDED	--	インスタンス値の全長がデータベース限界値を超えています。
34809	FLG_ERR_INVALID_VARCHAR_LENGTH	--	予約済
34810	FLG_ERR_INVALID_CREATOR	--	API の FLGCreateInst および FLGUpdateInst が入出力構造でエラーを検出しました。 CREATOR 値が、ログオン・ユーザー ID と違っています。これは、呼び出しユーザーがオブジェクト管理操作の実行を許可されていない場合の要件です。
35000	FLG_ERR_PRG_NOT_STARTED	--	予期しないオペレーティング・システム・エラーのため、プログラムを開始することができませんでした。
35001	FLG_ERR_PROG_PARM_TOOLONG	--	プログラム・オブジェクトのパラメーター・リスト (PARMLIST) 特性に指定されたパラメーターが、プラットフォーム固有のプログラム呼び出しには長すぎます。

表 24. 情報カタログ・マネージャーの理由コード (続き)

番号	理由コード	拡張コード	説明
35002	FLG_ERR_INV_PROG_PARM	--	プログラム・オブジェクト内のパラメーター・リストに、一致するもののないトークン識別子 (%) が含まれているか、あるいは、トークン識別子で区切られた特性が、操作点特性で識別されたオブジェクト・タイプの特性になっていません。
35003	FLG_ERR_PROGRAM_NOTEXIST	--	開始するプログラムが存在しないか、あるいはパス指定が正しくありません。
35004	FLG_ERR_INV_SYNTAX_STARTCMD	--	Program オブジェクトの STARTCMD 特性の値が無効です。
36001	FLG_ERR_ACCESS_DENIED	--	ファイルのオープンまたは読み取りを行おうとしたときに、アクセスが拒否されました。
36002	FLG_ERR_BAD_INVOCATION	--	情報カタログ・マネージャー・コマンド行呼び出しでエラーが発生しました。
36003	FLG_ERR_BROKEN_PIPE	--	指定されたファイルのオープンまたは読み取りを行うことができません。
36004	FLG_ERR_BUFFER_OVERFLOW	--	情報カタログ・マネージャーの内部エラー。

表 24. 情報カタログ・マネージャーの理由コード (続き)

番号	理由コード	拡張コード	説明
36005	FLG_ERR_CANNOT_MAKE	--	指定されたファイルを作成することができません。
36006	FLG_ERR_CLOSE_ERROR	--	ファイルをクローズすることができません。
36007	FLG_ERR_COPY_ERROR	--	ファイルをコピーすることができません。
36008	FLG_ERR_DELETE_ERROR	--	指定されたファイルを削除することができません。
36009	FLG_ERR_DEVICE_IN_USE	--	ファイルにアクセスすることができません。このファイルは現在使用中です。
36010	FLG_ERR_DIRECT_ACCESS_HANDLE	--	情報カタログ・マネージャーの内部エラー。
36011	FLG_ERR_DISK_FULL	--	ディスクがいっぱいで、ファイルを作成することができません。
36012	FLG_ERR_DRIVE_LOCKED	--	ドライブにアクセスすることができません。このドライブは現在使用中です。
36013	FLG_ERR_DUPHNDL_ERROR	--	情報カタログ・マネージャーの内部エラー。
36014	FLG_ERR_EAS_DIDNT_FIT	--	アイコン・ファイルの拡張属性が多すぎます。
36015	FLG_ERR_EA_LIST_INCONSISTENT	--	アイコン・ファイルの拡張属性の中に、無効なものがあります。

表 24. 情報カタログ・マネージャーの理由コード (続き)

番号	理由コード	拡張コード	説明
36016	FLG_ERR_EAS_NOT_SUPPORTED	--	拡張属性を備えたファイルを、拡張属性をサポートしないファイル・システムにコピーすることはできません。
36017	FLG_ERR_FILENAME_EXCED_RANGE	--	ファイル名またはパスが無効です。
36018	FLG_ERR_FILE_NOT_FOUND	--	指定されたパスとファイル名が見つかりませんでした。
36019	FLG_ERR_FINDFILE_ERROR	--	指定されたファイルが見つかりませんでした。
36020	FLG_ERR_FINDNEXT_ERROR	--	次のファイルが見つかりませんでした。
36021	FLG_ERR_INVALID_ACCESS	--	ファイルに書き込むことができません。このファイルは読み取り専用です。
36022	FLG_ERR_INVALID_DIRECTORY	--	指定されたディレクトリーが無効です。
36023	FLG_ERR_INVALID_DRIVE	--	指定されたドライブにアクセスすることができません。
36024	FLG_ERR_INVALID_EA_NAME	--	情報カタログ・マネージャーの内部エラー。
36025	FLG_ERR_INVALID_FILE_NAME	--	指定されたファイル名が無効です。
36026	FLG_ERR_INVALID_FUNCTION	--	情報カタログ・マネージャーの内部エラー。
36027	FLG_ERR_INVALID_HANDLE	--	情報カタログ・マネージャーの内部エラー。

表 24. 情報カタログ・マネージャーの理由コード (続き)

番号	理由コード	拡張コード	説明
36028	FLG_ERR_INVALID_PARAMETER	--	情報カタログ・マネージャーの内部エラー。
36029	FLG_ERR_INVALID_TARGET_HANDLE	--	情報カタログ・マネージャーの内部エラー。
36030	FLG_ERR_LOCK_VIOLATION	--	ファイルにアクセスすることができません。このファイルは別のアプリケーションによってロックされています。
36031	FLG_ERR_META_EXPANSION_TOO_LONG	--	情報カタログ・マネージャーの内部エラー。
36032	FLG_ERR_MORE_DATA	--	ファイルをオープンすることができません。ファイルが大きすぎます。
36033	FLG_ERR_NEED_EAS_FOUND	--	このファイルは、拡張属性をサポートしないドライブには移動することができません。このファイルには拡張属性が必要です。
36034	FLG_ERR_NEGATIVE_SEEK	--	情報カタログ・マネージャーの内部エラー。
36035	FLG_ERR_NOT_DOS_DISK	--	指定されたディスクは、有効なディスクではないか、あるいは存在していません。
36036	FLG_ERR_NO_MORE_FILES	--	情報カタログ・マネージャーの内部エラー。

表 24. 情報カタログ・マネージャーの理由コード (続き)

番号	理由コード	拡張コード	説明
36037	FLG_ERR_NO_MORE_SEARCH_HANDLES	--	情報カタログ・マネージャーは、操作点の最大数に達しました。 CONFIG.SYS ファイルで FILES= オプションの値を大きくしてください。
36038	FLG_ERR_OPEN_ERROR	--	アイコン・ファイル、タグ言語ファイル、エコー・ファイル、またはログ・ファイルをオープンすることができません。
36039	FLG_ERR_OPEN_FAILED	--	アイコン・ファイル、タグ言語ファイル、エコー・ファイル、またはログ・ファイルをオープンすることができません。
36040	FLG_ERR_PATH_NOT_FOUND	--	指定されたパスが見つかりませんでした。
36041	FLG_ERR_PIPE_BUSY	--	情報カタログ・マネージャーの内部エラー。
36042	FLG_ERR_READ_ERROR	--	情報カタログ・マネージャーの内部エラー。
36043	FLG_ERR_SEEK_ON_DEVICE	--	情報カタログ・マネージャーの内部エラー。
36044	FLG_ERR_SETFILEPTR_ERROR	--	情報カタログ・マネージャーの内部エラー。

表 24. 情報カタログ・マネージャーの理由コード (続き)

番号	理由コード	拡張コード	説明
36045	FLG_ERR_SHARING_BUFFER_EXCEEDED	--	バッファーのオーバーフローがあるので、このファイルは共用できません。
36046	FLG_ERR_SHARING_VIOLATION	--	このファイルにアクセスすることができません。このファイルは別のプロセスによって使用されています。
36047	FLG_ERR_TOO_MANY_OPEN_FILES	--	これ以上のファイルをオープンすることができません。 OS/2 では、FILES= オプションの値を大きくしてください。
36048	FLG_ERR_WRITE_ERROR	--	情報カタログ・マネージャーの内部エラー。
36049	FLG_ERR_WRITE_FAULT	--	ディスクに書き込むことができません。ディスクがロックされているか、あるいは読み取り不能になっている可能性があります。
36050	FLG_ERR_WRITE_PROTECT	--	ファイルに書き込むことができません。このファイルは読み取り専用です。

表 24. 情報カタログ・マネージャーの理由コード (続き)

番号	理由コード	拡張コード	説明
36200	FLG_ERR_NO_MORE_THREADS	--	これ以上のシステム論理経路を使用することができません。 続行するためには、なんらかの既存プログラムをクローズしてください。
36201	FLG_ERR_QDISK_FAIL	--	ディスク・ドライブに関する情報にアクセスすることができません。
37001	FLG_ERR_INV_RESTART_OPT	--	指定された再始動オプション (RestartOpt) が無効でした。 有効な値は、B、C、b、または c です。
37002	FLG_ERR_INV_OBJTYPE_OPT	--	ACTION.OBJTYPE タグのオプションが無効です。 有効なオプションは MERGE、ADD、UPDATE、DELETE、DELETE_EXT、および APPEND です。
37003	FLG_ERR_INV_OBJINST_OPT	--	ACTION.OBJINST タグのオプションが無効です。 有効なオプションは、ADD、UPDATE、DELETE、DELETE_TREE_REL、DELETE_TREE_ALL、および MERGE です。

表 24. 情報カタログ・マネージャーの理由コード (続き)

番号	理由コード	拡張コード	説明
37004	FLG_ERR_INV_RELATION_OPT	--	ACTION.RELATION タグのオプションが無効です。 有効なオプションは ADD および DELETE です。
37005	FLG_ERR_TAG_OUT_OF_SEQUENCE	--	タグ言語ファイルの ACTION タグのあとに続くタグが正しい順序になっていません。
37006	FLG_ERR_KEYNAME_TOO_LONG	--	INSTANCE タグの UUI 特性省略名の長さが最大長 (8) を超えています。
37007	FLG_ERR_INV_ACTION_TYPE	--	ACTION タグのキーワードが無効です。 有効なキーワードは OBJTYPE、OBJINST、または RELATION です。
37008	FLG_ERR_KEYWORD_TOO_LONG	--	タグのキーワードの長さがキーワードの最大長を超えています。
37009	FLG_ERR_PROPNAME_TOO_LONG	--	INSTANCE タグの特性省略名の長さが最大長 (8) を超えています。
37010	FLG_ERR_VALUE_TOO_LONG	--	タグ言語ファイル内の値の長さが、キーワード、特性省略名、または UUI 特性省略名の最大長を超えています。
37011	FLG_ERR_OBJTAG_DUP_KEYWORD	--	OBJECT タグで 2 回以上指定されているキーワードがあります。

表 24. 情報カタログ・マネージャーの理由コード (続き)

番号	理由コード	拡張コード	説明
37012	FLG_ERR_PROPTAG_DUP_KEYWORD	--	PROPERTY タグで 2 回以上指定されているキーワードがあります。
37013	FLG_ERR_RELTAG_DUP_KEYWORD	--	RELTYPE タグで 2 回以上指定されているキーワードがあります。
37014	FLG_ERR_INSTTAG_DUP_KEYNAME	--	INSTANCE タグで 2 回以上指定されている UUI 特性省略名があります。
37015	FLG_ERR_INSTTAG_DUP_PROPNAME	--	INSTANCE タグで 2 回以上指定されている特性省略名があります。
37016	FLG_ERR_OBJTAG_INV_KEYWORD	--	OBJECT タグのキーワードが無効です。 有効なキーワードは、TYPE、XTNAME、PHYNAME、ICOFILE および ICWFILE です。
37017	FLG_ERR_PROPTAG_INV_KEYWORD	--	PROPERTY タグのキーワードが無効です。 有効なキーワードは EXTNAME、DT、DL、SHRTNAME、NULLS、および UUISEQ です。

表 24. 情報カタログ・マネージャーの理由コード (続き)

番号	理由コード	拡張コード	説明
37018	FLG_ERR_RELTAG_INV_KEYWORD	--	RELTYPE タグのキーワードが無効です。 有効なキーワードは TYPE、SOURCETYPE、および TARGETYPE です。
37019	FLG_ERR_CMMTTAG_INV_KEYWORD	--	COMMIT タグのキーワードが無効です。 有効なキーワードは CHKPID です。
37020	FLG_ERR_INSTTAG_INV_KEYNAME	--	INSTANCE タグの UUI 特性省略名が無効です。
37021	FLG_ERR_INSTTAG_INV_PROPNAME	--	INSTANCE タグの特性省略名が無効です。 この特性省略名は、OBJECT タグで指定されたオブジェクト・タイプに存在しているものでなければなりません。
37022	FLG_ERR_INSTTAG_MISSING_SKEY	--	INSTANCE タグの最初のキーワードが SOURCEKEY になっていません。
37023	FLG_ERR_INSTTAG_MISSING_TKEY	--	関係を作成または削除する際に、INSTANCE タグの 2 番目のキーワードが TARGETKEY になっていません。

表 24. 情報カタログ・マネージャーの理由コード (続き)

番号	理由コード	拡張コード	説明
37024	FLG_ERR_TAGFILE_PREMATURE_EOF	--	タグ言語ファイルをインポートしているときに、情報カタログ・マネージャーが予期せずタグ言語ファイルの終了を検出しました。
37025	FLG_ERR_PROPTAG_INV_DT	--	PROPERTY タグの DT 値が無効です。 有効な値は、C、V、L、および T です。
37026	FLG_ERR_PROPTAG_RESERVED_SHRTNAME	--	予約済み特性の省略名が PROPERTY タグの SHRTNAME の値として指定されました。 省略名 OBJTYPID、PDATIME、および UPDATEBY は予約済みであり、SHRTNAME として指定することはできません。
37027	FLG_ERR_PROPTAG_INV_NULLS	--	PROPERTY タグの NULLS 値が無効です。 有効な値は、Y と N です。
37028	FLG_ERR_PROPTAG_INV_UUISEQ	--	PROPERTY タグの UUISEQ 値が無効です。 有効な値は、1、2、3、4、および 5 です。

表 24. 情報カタログ・マネージャーの理由コード (続き)

番号	理由コード	拡張コード	説明
37029	FLG_ERR_INSTTAG_RESERVED_PROPNAME	--	予約済み特性の特性省略名が INSTANCE タグで指定されました。 特性省略名 OBJTYPID、INSTIDNT、UPDATIME、および UPDATEBY は予約済みであり、値を割り当てることはできません。
37030	FLG_ERR_OBJTAG_MISSING_REQD_KEYWORD	--	OBJECT タグで必須キーワードが欠落しています。
37031	FLG_ERR_OBJTAG_KEYWORD_NOT_ALLOWED	--	OBJECT タグで指定されたキーワードは、現行の ACTION タグのキーワードとオプションでは許可されません。
37032	FLG_ERR_PROPTAG_MISSING_REQD_KEYWORD	--	PROPERTY タグで必須キーワードが欠落しています。 必須キーワードは、EXTNAME、DT、DL、SHRTNAME、および NULLS です。 SHRTNAME の値として NAME を指定した場合には、必須キーワードは SHRTNAME だけになります。

表 24. 情報カタログ・マネージャーの理由コード (続き)

番号	理由コード	拡張コード	説明
37033	FLG_ERR_RELTAG_MISSING_REQD_KEYWORD	--	RELTYPE タグで必須キーワードが欠落しています。 必須キーワードは TYPE、SOURCETYPE、および TARGETYPE です。
37034	FLG_ERR_INVALID_DISKCNTL_TAG	--	DISKCNTL タグの値およびキーワードが無効です。
37035	FLG_ERR_NO_VALID_INPUT_TAG	--	タグ言語ファイルに有効なタグが含まれていません。
37037	FLG_ERR_OBJTAG_INV_CATEGORY	--	OBJECT タグの CATEGORY 値が無効です。 有効な値は、GROUPING、ELEMENTAL、CONTACT、DICTIONARY、および SUPPORT です。
37038	FLG_ERR_RELTAG_INV_TYPE	--	RELTYPE タグの TYPE 値が無効です。 有効な値は、CONTAIN、CONTACT、LINK および ATTACHMENT です。
37039	FLG_ERR_MISSING_LPAREN	--	キーワード、UUI 特性省略名、または特性省略名のあとの左括弧が欠落しています。
37040	FLG_ERR_INSTTAG_NO_PROPNAME	--	INSTANCE タグに特性省略名が指定されていません。

表 24. 情報カタログ・マネージャーの理由コード (続き)

番号	理由コード	拡張コード	説明
37041	FLG_ERR_NO_VALUE	--	指定されたキーワードの値が欠落しています。
37042	FLG_ERR_NO_KEYWORD	--	キーワードを含んでいないタグがあります。 COMMENT、NL、および TAB 以外のすべてのタグでは、少なくともキーワードを 1 つ指定する必要があります。
37043	FLG_ERR_TAG_FOLLOWED_BY_GARBAGE	--	有効なタグのあとに余分な文字があります。
37044	FLG_ERR_BAD_PAREN_WITHIN_VALUE	--	この値で指定された括弧が無効です。 値の中の括弧は、単一引用符で囲まなければなりません。
37046	FLG_ERR_PROPTAG_KEYWORD_NOT_ALLOWED	--	SHRTNAME 値として NAME を指定した場合には、指定されたキーワードを PROPERTY タグで使用することはできません。 この場合の有効なキーワードは、SHRTNAME および UUISEQ です。
37047	FLG_ERR_UNEXPECTED_LPAREN	--	予想されるキーワード、UUI 特性省略名、または特性省略名の前に左括弧が指定されています。

表 24. 情報カタログ・マネージャーの理由コード (続き)

番号	理由コード	拡張コード	説明
37048	FLG_ERR_UNEXPECTED_RPAREN	--	予想される左括弧、キーワード、UUI 特性省略名、または特性省略名の前に右括弧が指定されています。
37300	FLG_ERR_CHKPT_DUP	--	情報カタログ・マネージャーの内部エラー。
37301	FLG_ERR_CHKPT_NOTEXIST	--	情報カタログ・マネージャーの内部エラー。
37302	FLG_ERR_INV_SAVEAREA_LEN	--	情報カタログ・マネージャーの内部エラー。
37303	FLG_ERR_INV_CHKPT_TOT_LEN	--	情報カタログ・マネージャーの内部エラー。
37304	FLG_ERR_MISSING_CHKPT_VALUE	--	情報カタログ・マネージャーの内部エラー。
37305	FLG_ERR_NO_MATCH_ON_CHKPTID	--	指定されたタグ言語ファイル内の COMMIT タグ・チェックポイント ID とシステムで保管されたチェックポイント ID とが一致しません。
37500	FLG_ERR_REQUEST_A_NEW_DISK_FAILED	--	ユーザーが、次の順番のタグ言語ファイル・ディスケットを順番どおりに挿入しませんでした。
37501	FLG_ERR_VERIFY_DISKETTE_SEQUENCE_FAILED	--	情報カタログ・マネージャーがディスケット順序を検査しているときに、エラーが発生しました。

表 24. 情報カタログ・マネージャーの理由コード (続き)

番号	理由コード	拡張コード	説明
37502	FLG_ERR_UNABLE_TO_FIND_REQUIRED_PROPERTY	--	ターゲット・データベースから、指定された特性省略名が見つかりませんでした。 この特性省略名は、ACTION.OBJINST(UPDATE) または ACTION.OBJINST(MERGE) を使用してオブジェクト・インスタンスの更新または組み合わせを行っているときに、INSTANCE タグで指定されたものです。
37503	FLG_ERR_UNABLE_TO_FIND_REQUIRED_OBJTYPE	--	OBJECT タグで指定されたオブジェクト・タイプ名がターゲット・データベースから見つかりませんでした。
37504	FLG_ERR_NONUNIQUE_UUI_KEY	--	指定された UUI 値が 1 つ以上のインスタンスを確認しています。

表 24. 情報カタログ・マネージャーの理由コード (続き)

番号	理由コード	拡張コード	説明
37505	FLG_ERR_MISMATCH_UUI_IN_MERGE	--	オブジェクト・タイプの組み合わせで、入力タグ言語ファイル内のオブジェクト・タイプを表す UUI 特性省略名が、情報カタログ・マネージャー・データベース内の同じオブジェクト・タイプの UUI 特性省略名と一致していません。
37506	FLG_ERR_DATA_LENGTH_CONVERSION_FAILED	--	情報カタログ・マネージャーの内部エラー。
37507	FLG_ERR_MISMATCH_DATA_LENGTH_IN_MERGE	--	入力タグ言語ファイルの ACTION.OBJTYPE(MERGE) タグのあとにある PROPERTY タグの DL (データ長) の値が、ターゲット情報カタログ・マネージャー・データベースにおける同じ特性の値と一致していません。

表 24. 情報カタログ・マネージャーの理由コード (続き)

番号	理由コード	拡張コード	説明
37508	FLG_ERR_MISMATCH_DATA_TYPE_IN_MERGE	--	入力タグ言語ファイル内の ACTION.OBJTYPE(MERGE) タグのあとにある PROPERTY タグの DT (データ・タイプ長) の値が、ターゲット情報カタログ・マネージャー・データベースにおける同一オブジェクト・タイプに関する同じ特性の値と一致していません。
37509	FLG_ERR_MISMATCH_PROPERTY_NAME_IN_MERGE	--	入力タグ言語ファイルの ACTION.OBJTYPE(MERGE) タグのあとにある PROPERTY タグの SHRTNAME(特性省略名) の値が、情報カタログ・マネージャー・データベースにおける同一オブジェクト・タイプに関するどの特性とも一致していません。

表 24. 情報カタログ・マネージャーの理由コード (続き)

番号	理由コード	拡張コード	説明
37510	FLG_ERR_MISMATCH_CATEGORY_IN_MERGE	--	入力タグ言語ファイルにおける ACTION.OBJTYPE(MERGE) タグのあとにある OBJECT タグの CATEGORY の値が、情報カタログ・マネージャー・データベースにおける同一オブジェクト・タイプに関する値と一致していません。
37511	FLG_ERR_MISSING_REQUIRED_OBJTYPE_MERGE_STATEMENT		ACTION.OBJTYPE(MERGE) を使用して対応するオブジェクト・タイプを組み合わせてからでなければ、ACTION.OBJINST(MERGE) を使用してオブジェクト・インスタンスを組み合わせることはできません。 ACTION.OBJTYPE(MERGE) は、同じオブジェクト・タイプに関する ACTION.OBJINST(MERGE) よりも前に処理しなければなりません。
37512	FLG_ERR_NONUNIQUE_SOURCE_UUI_KEY	--	予約済
37513	FLG_ERR_NONUNIQUE_TARGET_UUI_KEY	--	予約済
37514	FLG_ERR_NO_TAGFILE_ON_DISKETTE	--	提供されたディスケットから入力タグ言語ファイルが見つかりませんでした。

表 24. 情報カタログ・マネージャーの理由コード (続き)

番号	理由コード	拡張コード	説明
37515	FLG_ERR_WRONG_DISK_SEQUENCE	--	タグ言語ファイルを含むディスケットが誤った順序で挿入されました。
37516	FLG_ERR_REQ_INST_NOTFOUND	--	更新するインスタンスが見つかりませんでした。
37801	FLG_ERR_NO_UUI	--	エクスポート中に UUI のないオブジェクトが見つかり、処理することができません。
37802	FLG_ERR_CREATEREG_FAILED	--	予約済
37803	FLG_ERR_UPDATEREG_FAILED	--	予約済
37804	FLG_ERR_GETREG_FAILED	理由コード	エクスポートで FLGGetReg が呼び出され、これによってエラーが戻されました。 このエラーがエクスポートに与える影響については、ログ・ファイルを参照してください。
37805	FLG_ERR_DELETEREG_FAILED	--	予約済
37806	FLG_ERR_CREATETYPE_FAILED	--	予約済
37807	FLG_ERR_APPENDTYPE_FAILED	--	予約済
37808	FLG_ERR_GETTYPE_FAILED	--	予約済
37809	FLG_ERR_DELETETYPE_FAILED	--	予約済
37820	FLG_ERR_CREATEINST_FAILED	--	予約済
37821	FLG_ERR_UPDATEINST_FAILED	--	予約済

表 24. 情報カタログ・マネージャーの理由コード (続き)

番号	理由コード	拡張コード	説明
37822	FLG_ERR_GETINST_FAILED	理由コード	エクスポートで FLGGetInst が呼び出され、これによってエラーが戻されました。 このエラーがエクスポートに与える影響については、ログ・ファイルを参照してください。
37823	FLG_ERR_DELETEINST_FAILED	--	予約済
37824	FLG_ERR_LISTTYPE_FAILED	--	予約済
37825	FLG_ERR_SEARCH_FAILED	--	予約済
37826	FLG_ERR_RELATE_FAILED	--	予約済
37827	FLG_ERR_LISTCONTACTS_FAILED	理由コード	エクスポートで FLGListContacts が呼び出され、これによってエラーが戻されました。 このエラーがエクスポートに与える影響については、ログ・ファイルを参照してください。
37828	FLG_ERR_NAVIGATE_FAILED	理由コード	エクスポートで FLGNavigate が呼び出され、これによってエラーが戻されました。 このエラーがエクスポートに与える影響については、ログ・ファイルを参照してください。

表 24. 情報カタログ・マネージャーの理由コード (続き)

番号	理由コード	拡張コード	説明
37829	FLG_ERR_FREEMEM_FAILED	理由コード	エクスポートで FLGFreeMem が呼び出され、これによってエラーが戻されました。 このエラーがエクスポートに与える影響については、ログ・ファイルを参照してください。
37831	FLG_ERR_LISTASSOC_FAILED	理由コード	この機能で FLGListAssociates が呼び出され、これによってエラーが戻されました。
37901	FLG_ERR_NULL_LOGFILE	--	ログ・ファイルのポインター・パラメーター値が NULL になっています。 このパラメーターには値を指定する必要があります。
37902	FLG_ERR_LOGFILE_OPENERR	理由コード	インポートまたはエクスポートでログ・ファイルをオープンしようとしたときにエラーが発生しました。 拡張コードに、このエラーに関する理由コードが示されています。

表 24. 情報カタログ・マネージャーの理由コード (続き)

番号	理由コード	拡張コード	説明
37904	FLG_ERR_LOGFILE_WRITEERR	理由コード	インポートまたはエクスポートでログ・ファイルに書き込みを行おうとしたときにエラーが発生しました。 拡張コードに、このエラーに関する理由コードが示されています。
37906	FLG_ERR_LOGFILE_CLOSEERR	理由コード	インポートまたはエクスポートでログ・ファイルをクローズしようとしたときにエラーが発生しました。 拡張コードに、このエラーに関する理由コードが示されています。
37908	FLG_ERR_INV_TAGFILE_LEN	--	次のいずれかが起こりました。 • 指定されたタグ言語ファイルの名前が NULL になっています。 • (パス情報を含む) タグ言語ファイルの完全な名前の長さが、許される最大長 (259) を超えています。 • タグ言語ファイル名とエクステンションの長さが、許される最大長 (240) を超えています。

表 24. 情報カタログ・マネージャーの理由コード (続き)

番号	理由コード	拡張コード	説明
37909	FLG_ERR_INV_LOGFILE_LEN	--	次のいずれかが起こりました。 • 指定されたログ・ファイルの名前が NULL になっています。 • (パスを含む) 名前全体の長さが、許される最大長 (259) を超えています。
37910	FLG_ERR_INV_TAGFILE	--	タグ言語ファイル用に指定されたドライブは、情報カタログ・マネージャーがアクセスしようとしたときにエラーが発生したため、無効です。 タグ言語ファイルが MDIS 様式の場合、このドライブは、取り外し可能ドライブであってはなりません。
37911	FLG_ERR_INV_LOGFILE	--	ログ・ファイル用に指定されたドライブが無効です。指定されたドライブは、取り外し可能なドライブであるか、あるいは情報カタログ・マネージャーがアクセスしようとしたときにエラーが発生しています。

表 24. 情報カタログ・マネージャーの理由コード (続き)

番号	理由コード	拡張コード	説明
37912	FLG_ERR_ECHOFILE_OPENERR	理由コード	インポートでエコー・ファイルをオープンしようとしたときに、エラーが発生しました。 拡張コードに、このエラーに関する理由コードが示されています。
37913	FLG_ERR_TAGFILE_READERR	理由コード	インポートでタグ言語ファイルを読み取ろうとしたときに、エラーが発生しました。 拡張コードに、このエラーに関する理由コードが示されています。
37914	FLG_ERR_ECHOFILE_WRITEERR	理由コード	インポートでエコー・ファイルに書き込もうとしたときに、エラーが発生しました。 拡張コードに、このエラーに関する理由コードが示されています。
37915	FLG_ERR_INV_ICOPATH_LEN	--	指定されたアイコン・パスが長すぎます。 (ドライブとディレクトリーを含む) アイコン・パスの最大長は 246 です。

表 24. 情報カタログ・マネージャーの理由コード (続き)

番号	理由コード	拡張コード	説明
37919	FLG_ERR_ICOPATH_NONBLANK_EXT	--	指定されたアイコン・パス (pszIcoPath) にはエクステンションが含まれています。 この値には、パスだけを含めてください。
37920	FLG_ERR_INV_ICOPATH	--	アイコン・パスで指定されたドライブまたはエクステンションは、次のいずれかの理由により無効です。 • ドライブが指定されていないか、取り外し可能なドライブであるか、あるいは情報カタログ・マネージャーがこのドライブから読み取りを行っているときにエラーが発生しました。 • アイコン・パスでファイル・エクステンションが指定されました。

表 24. 情報カタログ・マネージャーの理由コード (続き)

番号	理由コード	拡張コード	説明
37921	FLG_ERR_TAGFILE_OPENERR	理由コード	タグ言語ファイルをオープンしたときに、インポート、エクスポート、または FLGXferTagBuf でエラーが発生しました。 拡張コードに、オープン・エラーに関する理由コードが示されています。
37922	FLG_ERR_TAGFILE_CLOSEERR	理由コード	タグ言語ファイルをクローズしたときに、インポート、エクスポート、または FLGXferTagBuf でエラーが発生しました。 拡張コードに、このエラーに関する理由コードが示されています。
37923	FLG_ERR_ECHOFILE_CLOSEERR	理由コード	インポートでエコー・ファイルをクローズしようとしたときに、エラーが発生しました。 拡張コードに、このエラーに関する理由コードが示されています。

表 24. 情報カタログ・マネージャーの理由コード (続き)

番号	理由コード	拡張コード	説明
37924	FLG_ERR_INV_ECHOFILE_LEN	--	タグ言語ファイル名と ECH エクステンションを含むログ・ファイル・パスの長さが、エコー・ファイルの完全なパスと名前に許される最大長を超えています。 この最大長は 259 文字です。
37925	FLG_ERR_MAX_OBJTYPE_EXCEEDED	--	タグ言語ファイルに含まれている個別オブジェクト・タイプの数が、インポートまたはエクスポート時に許される最大数 (3500) を超えています。
37926	FLG_ERR_TAGFILE_WRITEERR	理由コード	タグ言語ファイルに書き込もうとしたときに、エクスポートまたは FLGXferTagBuf API でエラーが発生しました。 拡張コードに、書き込みエラーに関する理由コードが示されています。
37928	FLG_ERR_INV_TAGFILE_EXT	--	タグ言語ファイルに指定したファイル名に、ECH のエクステンションが付けられています。このエクステンションは無効です。

表 24. 情報カタログ・マネージャーの理由コード (続き)

番号	理由コード	拡張コード	説明
37929	FLG_ERR_INV_LOGFILE_EXT	--	ログ・ファイルに指定したファイル名に、ECH のエクステンションが付けられています。このエクステンションは無効です。
37930	FLG_ERR_TAGFILE_LOGFILE_CONFLICT	--	指定したログ・ファイルはタグ言語ファイルと同じです。これら 2 つのファイルは、別々のファイルでなければなりません。
38000	FLG_ERR_INVALID_EXPORT_IOSTRUCT	オブジェクトの順序番号	FLGExport の入力構造が無効です。
38001	FLG_ERR_INVALID_CFLAG	オブジェクトの順序番号	FLGExport 入力構造内の含まれる側のフラグ値が無効です。 有効な値は、Y または N です。
38002	FLG_ERR_INVALID_TFLAG	オブジェクトの順序番号	FLGExport 入力構造内の連絡担当者フラグ値が無効です。 有効な値は、Y または N です。
38003	FLG_ERR_TAGFILE_EXIST	--	エクスポートの出力タグ言語ファイル (pszTagFileID) に指定された名前が、すでに存在しているファイルを指しています。 出力タグ言語ファイルの名前は、既存のものであってはなりません。

表 24. 情報カタログ・マネージャーの理由コード (続き)

番号	理由コード	拡張コード	説明
38004	FLG_ERR_GET_ICON_FAILED	理由コード	指定されたオブジェクト・タイプのアイコンをエクスポートすることができません。
38005	FLG_ERR_INVALID_AFLAG	オブジェクトの順序番号	エクスポート入力構造の付加フラグが無効です。有効な値は、'Y' または 'N' です。
38006	FLG_ERR_INVALID_LFLAG	オブジェクト・タイプの順序番号	エクスポート入力構造のリンク・フラグが無効です。有効な値は、'Y' または 'N' です。
39000	FLG_ERR_UPM_FAIL	--	ユーザー・プロファイル管理ユーティリティーが失敗しました (ログオンが失敗したか、またはログオン・ユーザー ID が接続されたユーザー ID と異なります)。
39001	FLG_ERR_INV_INPUT_PARM	--	コマンドの入力パラメーターのキーワードが無効であるか、あるいは欠落しています。
39002	FLG_ERR_MISSING_PARM_VALUE	--	コマンドの入力パラメーターの値が無効であるか、あるいは欠落しています。

表 24. 情報カタログ・マネージャーの理由コード (続き)

番号	理由コード	拡張コード	説明
39003	FLG_ERR_INIT_BIDI_ERROR	--	情報カタログ・マネージャーが、両方向環境用に初期化をしようとしたときに、エラーが発生しました。これは、情報カタログ・マネージャーが、アラビア語またはヘブライ語用の機械で実行しているときにのみ適用されます。
39201	FLG_ERR_INVALID_USERTYPE_FOR_UPDATE	--	更新を指定されたユーザー・タイプが無効です。有効なタイプは、1 次管理者またはバックアップ管理者のどちらかです。
39202	FLG_ERR_INVALID_USERTYPE_FOR_CRT_OR_DEL	--	作成または削除を指定されたユーザー・タイプが無効です。オブジェクト管理作業を実行する許可のあるユーザーだけが、作成または削除されます。
39203	FLG_ERR_INVALID_ID_BAD_CHAR	--	指定されたユーザー ID に無効文字が含まれています。有効な文字については、データベース資料を参照してください。
39204	FLG_ERR_INVALID_ID_NUM_START	--	指定されたユーザー ID は数値で始まります。これは、有効な開始文字ではありません。

表 24. 情報カタログ・マネージャーの理由コード (続き)

番号	理由コード	拡張コード	説明
39205	FLG_ERR_INVALID_ID_IMB_BLANK	--	指定されたユーザー ID に組み込みブランクが含まれています。これは認められていません。
39206	FLG_ERR_INVALID_MUU_OPT	--	FLGManageUsers API に指定されたオプションが無効です。有効なアクションは、FLG_ACTION_CREATE、 FLG_ACTION_UPDATE、FLG_ACTION_DELETE、または FLG_ACTION_LIST です。
39209	FLG_ERR_INVALID_PADMIN_USERID	--	1 次管理者に指定されたユーザー ID が無効です。データベース資料でユーザー ID 構文を調べてください。
39210	FLG_ERR_INVALID_BADMIN_USERID	--	バックアップ管理者に指定されたユーザー ID が無効です。データベース資料でユーザー ID 構文を調べてください。
39211	FLG_ERR_INVALID_POWERUSER_USERID	無効な入力構造のユーザー ID を指すインデックスを含む	指定されたユーザー ID が無効です。データベース資料でユーザー ID 構文を調べてください。
39502	FLG_ERR_CDF_ERROR	--	予約済
39504	FLG_ERR_INSTPROFILE_ERROR	--	予約済
39700	FLG_ERR_TERM_FAIL_ROLLBACK_CLOSE	--	予約済
39701	FLG_ERR_TERM_FAIL_ROLLBACK	--	予約済
39702	FLG_ERR_TERM_FAIL_COMMIT	--	予約済

表 24. 情報カタログ・マネージャーの理由コード (続き)

番号	理由コード	拡張コード	説明
40001	FLG_ERR_INVALID_CONFIG_PROFILE	--	「MDIS 構成 (MDIS Configuration)」プロファイル・ファイルに、有効な BEGIN CONFIGURATION セクションが含まれていません。
40002	FLG_ERR_CONFIGFILE_READERR	理由コード	MDIS インポートで「構成 (Configuration)」プロファイル・ファイルを読み取ろうとしたときに、エラーが発生しました。
40003	FLG_ERR_CONFIGFILE_CLOSEERR	理由コード	MDIS インポートで「構成 (Configuration)」プロファイル・ファイルをクローズしようとしたときに、エラーが発生しました。
40006	FLG_ERR_CONFIGFILE_INV_BEGIN_STMT	--	「MDIS 構成 (MDIS Configuration)」プロファイル・ファイルに、無効な BEGIN ステートメントが含まれています。有効なステートメントは BEGIN CONFIGURATION です。

表 24. 情報カタログ・マネージャーの理由コード (続き)

番号	理由コード	拡張コード	説明
40007	FLG_ERR_CONFIGFILE_INV_END_STMT	--	「MDIS 構成 (MDIS Configuration)」プロファイル・ファイルに、無効な END ステートメントが含まれています。有効なステートメントは END CONFIGURATION です。
40010	FLG_ERR_CONFIGFILE_INV_KEYWORD	--	「MDIS 構成 (MDIS Configuration)」プロファイル・ファイルに、無効なキーワードが含まれています。
40011	FLG_ERR_CONFIGFILE_INV_TEXT	--	「MDIS 構成 (MDIS Configuration)」プロファイル・ファイルに、無効なテキストが含まれています。
40012	FLG_ERR_CONFIGFILE_INV_VALUE	--	「MDIS 構成 (MDIS Configuration)」プロファイル・ファイルに、無効なキーワード値が含まれています。
40013	FLG_ERR_CONFIGFILE_VALUE_TOO_LONG	--	「MDIS 構成 (MDIS Configuration)」プロファイル・ファイルに、そのキーワードの最大許可長を超えるキーワード値が含まれています。

表 24. 情報カタログ・マネージャーの理由コード (続き)

番号	理由コード	拡張コード	説明
40015	FLG_ERR_CONFIGFILE_PREMATURE_EOF	--	MDIS インポートで、「構成 (Configuration)」プロファイル・ファイルが突然終了してしまいました。
40021	FLG_ERR_INVALID_TOOL_PROFILE	--	「MDIS ツール (MDIS Tool)」プロファイル・ファイルに、有効な BEGIN TOOL セクションが含まれていません。
40022	FLG_ERR_TOOLFILE_READERR	理由コード	MDIS インポートで「ツール (Tool)」プロファイル・ファイルを読み取ろうとしたときに、エラーが発生しました。
40023	FLG_ERR_TOOLFILE_CLOSEERR	理由コード	MDIS インポートで「ツール (Tool)」プロファイル・ファイルをクローズしようとしたときに、エラーが発生しました。
40026	FLG_ERR_TOOLFILE_INV_BEGIN_STMT	--	「MDIS ツール (MDIS Tool)」プロファイル・ファイルに、無効な BEGIN ステートメントが含まれています。有効なステートメントは BEGIN TOOL、BEGIN APPLICATION DATA です。

表 24. 情報カタログ・マネージャーの理由コード (続き)

番号	理由コード	拡張コード	説明
40027	FLG_ERR_TOOLFILE_INV_END_STMT	--	「MDIS ツール (MDIS Tool)」プロファイル・ファイルに、無効な END ステートメントが含まれています。有効なステートメントは END TOOL、END APPLICATION DATA です。
40030	FLG_ERR_TOOLFILE_INV_KEYWORD	--	「MDIS ツール (MDIS Tool)」プロファイル・ファイルに、無効なキーワードが含まれています。
40031	FLG_ERR_TOOLFILE_INV_TEXT	--	「MDIS ツール (MDIS Tool)」プロファイル・ファイルに、無効なテキストが含まれています。
40032	FLG_ERR_TOOLFILE_INV_VALUE	--	「MDIS ツール (MDIS Tool)」プロファイル・ファイルに、無効なキーワード値が含まれています。
40033	FLG_ERR_TOOLFILE_VALUE_TOO_LONG	--	「MDIS ツール (MDIS Tool)」プロファイル・ファイルに、そのキーワードの最大許可長を超えるキーワード値が含まれています。

表 24. 情報カタログ・マネージャーの理由コード (続き)

番号	理由コード	拡張コード	説明
40034	FLG_ERR_TOOLFILE_CONFLICTING_VALUES	--	「MDIS ツール (MDIS Tool)」プロファイル・ファイルに、矛盾する RECORD、DIMENSION、または ELEMENT 値が含まれています。
40050	FLG_ERR_TOOLFILE_PREMATURE_EOF	--	MDIS インポートで、「ツール (Tool)」プロファイル・ファイルが突然終了してしまいました。
40100	FLG_ERR_UNSUPPORTED_MDIS_FUNCTION	--	「構成 (Configuration)」プロファイル・ファイルに、情報カタログ・マネージャーがサポートしていない機能が指定されています。
40101	FLG_ERR_MISSING_REQ_MDIS_KEYWORD	--	タグ言語ファイルに必須の MDIS キーワードが存在しません。
40110	FLG_ERR_TAGFILE_INV_KEYWORD	--	MDIS タグ言語ファイルに、無効なキーワードが含まれています。
40111	FLG_ERR_TAGFILE_INV_TEXT	--	MDIS タグ言語ファイルに、無効なテキストが含まれています。
40112	FLG_ERR_TAGFILE_INV_VALUE	--	MDIS タグ言語ファイルに、無効なキーワード値が含まれています。

表 24. 情報カタログ・マネージャーの理由コード (続き)

番号	理由コード	拡張コード	説明
40113	FLG_ERR_TAGFILE_VALUE_TOO_LONG	--	MDIS タグ言語ファイルに、そのキーワードの最大許可長を超えるキーワード値が含まれています。
40115	FLG_ERR_MISSING_DQUOTE	--	キーワードの後の二重引用符が欠落しています。
40116	FLG_ERR_UNEXPECTED_DQUOTE	--	二重引用符が不適切な位置にありました。
40117	FLG_ERR_SPECIFIED_PROPERTY_NOT_FOUND	--	ターゲット・データベースから、指定された特性省略名が見つかりませんでした。
40118	FLG_ERR_TAGFILE_INV_END_STMT	--	MDIS タグ言語ファイルに、無効な END ステートメントが含まれています。
40119	FLG_ERR_TAGFILE_INV_BEGIN_STMT	--	MDIS タグ言語ファイルに、無効な BEGIN ステートメントが含まれています。
40130	FLG_ERR_INV_RECORD_SECTION	--	MDIS タグ言語ファイルで、BEGIN RECORD セクションのネストが不適切です。
40131	FLG_ERR_INV_DIMENSION_SECTION	--	MDIS タグ言語ファイルで、BEGIN DIMENSION セクションのネストが不適切です。
40132	FLG_ERR_INV_SUBSCHEMA_SECTION	--	MDIS タグ言語ファイルで、BEGIN SUBSCHEMA セクションのネストが不適切です。

表 24. 情報カタログ・マネージャーの理由コード (続き)

番号	理由コード	拡張コード	説明
40201	FLG_ERR_DUPLICATE_IDENTIFIER	--	MDIS タグ言語ファイルで、識別子の値が重複しています。
40202	FLG_ERR_INV_IDENTIFIER_REFERENCE	--	SourceObjectIdentifier または TargetObjectIdentifier の値が、タグ言語ファイルで以前に定義した識別子の値を示していません。
40211	FLG_ERR_INV_PART1_VALUE	--	MDIS オブジェクトの最初の部分の値が、親の値と一致していません。
40212	FLG_ERR_INV_PART2_VALUE	--	MDIS オブジェクトの 2 番目の部分の値が、親の値と一致していません。
40213	FLG_ERR_INV_PART3_VALUE	--	MDIS オブジェクトの 3 番目の部分の値が、親の値と一致していません。
40214	FLG_ERR_INV_PART4_VALUE	--	MDIS オブジェクトの 4 番目の部分の値が、親の値と一致していません。

表 24. 情報カタログ・マネージャーの理由コード (続き)

番号	理由コード	拡張コード	説明
40215	FLG_ERR_MDIS_WORK_BUFFER_OVERFLOW	--	MDIS ファイル (「構成 (Configuration)」プロファイル・ファイル、「ツール (Tool)」プロファイル・ファイル、または タグ言語ファイル) に、内部作業バッファーの最大許可サイズ (32700 バイト) を超える値が含まれています。
40216	FLG_ERR_MDIS_APPL_DATA_TOO_LONG	--	MDIS タグ言語ファイルの ApplicationData セクションが、情報カタログ・マネージャー・アプリケーション・データ・オブジェクト・タイプの限界値を超過しています。情報カタログ・マネージャー・アプリケーション・データ・オブジェクト・タイプは、それぞれ 32700 バイトの 10 特性に限定されています。
80000	FLG_SEVERR	--	プレースホルダー。重大エラーの数値範囲の始まりを示します。
80002	FLG_SEVERR_NO_MEMORY	--	情報カタログ・マネージャーはこれ以上メモリーを割り振ることができません。

表 24. 情報カタログ・マネージャーの理由コード (続き)

番号	理由コード	拡張コード	説明
80003	FLG_SEVERR_MEM_ERROR	--	次のいずれかが起こりました。 • ハードウェアのメモリー割り込みが発生しました。 • 情報カタログ・マネージャー・ヒープ内でなんらかの破壊が生じたため、情報カタログ・マネージャーがメモリーの割り振りまたは割り振り解除を行えません。
80004	FLG_SEVERR_NO_CSA	--	情報カタログ・マネージャーの内部エラー。
80005	FLG_SEVERR_APIDLL_FAILURE	--	API DLL で API 呼び出しが欠落しているか、あるいは API DLL をロードすることができませんでした。
80006	FLG_SEVERR_VIOPOPUP_FAIL	--	情報カタログ・マネージャーはビデオ入出力 (VIO) を使用して OS/2 文字ベースのエラー・メッセージを表示することができません。

表 24. 情報カタログ・マネージャーの理由コード (続き)

番号	理由コード	拡張コード	説明
80007	FLG_SEVERR_BIDIDLL_FAILURE	--	情報カタログ・マネージャーが、PMBIDI.DLL をロードしようとしたときに、エラーが発生しました。この DLL は、情報カタログ・マネージャーがアラビア語またはヘブライ語用の機械で実行されるときに必要です。
80008	FLG_SEVERR_DG2IFORDLL_FAILURE	--	必要な DG2IFOR.DLL ファイルが検出できなかったか、または無効です。情報カタログ・マネージャーは継続できません。
81000	FLG_SEVERR_STARTDBM_FAIL	--	ローカル・データベース管理システムを開始できません。 SQLCODE の説明については、データベースの資料を参照してください。
81001	FLG_SEVERR_STARTDB_FAIL	--	予約済
81002	FLG_SEVERR_DB_DISCONNECTED	--	予期しないときにデータベースが切断されました。
81003	FLG_SEVERR_DB_INCONSISTENT	--	情報カタログ・マネージャーが情報カタログ・マネージャー・データベースから矛盾を検出しました。

表 24. 情報カタログ・マネージャーの理由コード (続き)

番号	理由コード	拡張コード	説明
81004	FLG_SEVERR_COMMIT_FAIL	--	情報カタログ・マネージャー・データベースに対するコミット呼び出しが失敗しました。
81005	FLG_SEVERR_ROLLBACK_FAIL	--	情報カタログ・マネージャー・データベースに対するロールバック呼び出しが失敗しました。
81006	FLG_SEVERR_NO_DBSPACE	--	データベース・サーバーのスペースを使い切ったか、ファイル・システムが満杯です。
81007	FLG_SEVERR_DB_AUTO_ROLLBACK_COMPLETE	データベース SQLCODE	情報カタログ・マネージャーがデータベース・エラーを検出し、データベースに対するコミットされていない変更をロールバックしました。 情報カタログ・マネージャーがロールバックを行う原因となったエラー状態を記述するデータベース SQLCODE については、拡張コードを参照してください。

表 24. 情報カタログ・マネージャーの理由コード (続き)

番号	理由コード	拡張コード	説明
81008	FLG_SEVERR_DB_AUTO_ROLLBACK_FAIL	データベース SQLCODE	情報カタログ・マネージャーがデータベース・エラーを検出し、データベースに対するコミットされていない変更をロールバックしようとしましたが、このロールバックが失敗しました。 情報カタログ・マネージャーがロールバックを行う原因となったエラー状態を記述するデータベース SQLCODE については、拡張コードを参照してください。 このデータベースは矛盾していて、回復が必要になっている可能性があります。
82000	FLG_SEVERR_INIT_FAIL	--	情報カタログ・マネージャーで予期しない状態 (おそらく、OS/2 の内部メモリー・エラー) が発生し、情報カタログ・マネージャーを正常に実行できなくなりました。

表 24. 情報カタログ・マネージャーの理由コード (続き)

番号	理由コード	拡張コード	説明
82001	FLG_SEVERR_TERM_FAIL	--	情報カタログ・マネージャーで予期しない状態 (おそらく、OS/2 の内部メモリー・エラー) が発生し、情報カタログ・マネージャーで割り振り済み資源を解放できなくなりました。この資源は、呼び出し元のアプリケーション・セッションが終了したときに解放されます。
82002	FLG_SEVERR_TERM_FAIL_CLOSE	--	予約済
82200	FLG_SEVERR_GETREG_FAILED	理由コード	エクスポートで FLGGetReg が呼び出され、これによって重大エラーが戻されました。
82201	FLG_SEVERR_GETINST_FAILED	理由コード	エクスポートで FLGGetInst が呼び出され、これによって重大エラーが戻されました。
82202	FLG_SEVERR_LISTCONTACTS_FAILED	理由コード	エクスポートで FLGListContacts が呼び出され、これによって重大エラーが戻されました。
82203	FLG_SEVERR_NAVIGATE_FAILED	理由コード	エクスポートで FLGNavigate が呼び出され、これによって重大エラーが戻されました。
82204	FLG_SEVERR_FREEMEM_FAILED	理由コード	エクスポートで FLGFreeMem が呼び出され、これによって重大エラーが戻されました。

表 24. 情報カタログ・マネージャーの理由コード (続き)

番号	理由コード	拡張コード	説明
82400	FLG_SEVERR_THREAD_FAILED	--	新規の論理経路を作成するときに重大エラーが発生し、情報カタログ・マネージャーを続行することができません。
82500	FLG_SEVERR_PARMS_MISSING	--	情報カタログ・マネージャーの必要なシステム表が破壊されているか、あるいは欠落しています。
82501	FLG_SEVERR_DGEMPTY	--	情報カタログ・マネージャー・データベースに登録またはオブジェクト・タイプが含まれていません。この情報カタログ・マネージャー・データベースは破壊されています。 バックアップ用データベース・ファイルを使用してデータベースを回復してください。
82502	FLG_SEVERR_TYPE_WOUT_PROPERTY	--	指定されたオブジェクト・タイプに関する特性が存在しないか、あるいは情報カタログ・マネージャーが特性を検索できません。
82503	FLG_SEVERR_MORE_THAN_ONE_KA	--	安全保護違反が発生しました。同時に複数の管理者がログオンされました。
83000	FLG_SEVERR_SESSION_ABENDED	--	予約済
83001	FLG_SEVERR_CDF_ERROR	--	予約済

表 24. 情報カタログ・マネージャーの理由コード (続き)

番号	理由コード	拡張コード	説明
83002	FLG_SEVERR_INTERNAL_ERROR	--	予約済
84000	FLG_SEVERR_DEMO_EXPIRED	--	IBM 情報カタログ・マネージャー管理者の評価期間が終了しました。お近くのソフトウェア再販店または IBM 担当員に連絡して、製品をご注文ください。
84101	FLG_SEVERR_DB_CONNECT_FAILED	--	データベースに接続できません。SQLCODE の説明については、データベースの資料を参照してください。
84102	FLG_SEVERR_DB_BIND	--	情報カタログ・マネージャーを情報カタログにバインドできません。情報カタログ・マネージャーに予期できないデータベース・エラーが発生したか、現行ディレクトリーまたはパスでバインド・ファイルを検出できませんでした。
84103	FLG_SEVERR_INSAUTH_BIND	--	情報カタログ・マネージャーを情報カタログにバインドするには、SYSADM 権限が必要です。
84104	FLG_SEVERR_CREATETAB	--	情報カタログ・マネージャー・システム表を作成できません。

表 24. 情報カタログ・マネージャーの理由コード (続き)

番号	理由コード	拡張コード	説明
84105	FLG_SEVERR_INSAUTH_GRANT	--	情報カタログへアクセスできるようにするには、SYSADM 権限が必要です。
84106	FLG_SEVERR_CREATECOLLECTION	--	情報カタログ・マネージャーは AS/400® ライブラリー・コレクションの作成に失敗しました。
84107	FLG_SEVERR_ICON_NOT_GENERATED	--	情報カタログ・マネージャーにシステム・エラーが発生したか、情報カタログ・マネージャー・アイコン・ファイルまたは情報カタログ・マネージャー実行可能ファイルを見つけられません。 情報カタログ・マネージャー・アイコンは生成されません。
84108	FLG_SEVERR_DGCOL_NOTEXIST	--	このユーティリティーを起動する前に、AS/400 ライブラリー・コレクション、情報カタログ・マネージャーを作成する必要があります。

表 24. 情報カタログ・マネージャーの理由コード (続き)

番号	理由コード	拡張コード	説明
84109	FLG_SEVERR_DB_NOTFOUND	--	情報カタログ・マネージャーは指定したデータベースを検出できません。そのデータベースが存在していないのであれば、作成してください。その後、そのリモート・データベースをワークステーションに登録してください。

特記事項

本書において、日本では発表されていない IBM 製品 (機械およびプログラム)、プログラミングまたはサービスについて言及または説明する場合があります。しかし、このことは、弊社がこのような IBM 製品、プログラミングまたはサービスを、日本で発表する意図があることを必ずしも示すものではありません。本書で、 IBM ライセンス・プログラムまたは他の IBM 製品に言及している部分があっても、このことは当該プログラムまたは製品のみが使用可能であることを意味するものではありません。これらのプログラムまたは製品に代えて、 IBM の知的所有権を侵害することのない機能的に同等な他社のプログラム、製品またはサービスを使用することができます。ただし、IBM によって明示的に指定されたものを除き、これらのプログラムまたは製品に関連する稼働の評価および検証はお客様の責任で行っていただきます。

IBM および他社は、本書で説明する主題に関する特許権 (特許出願を含む)、商標権、または著作権を所有している場合があります。本書は、これらの特許権、商標権、および著作権について、本書で明示されている場合を除き、実施権、使用権等を許諾することを意味するものではありません。実施権、使用権等の許諾については、下記の宛先に、書面にてご照会ください。

〒106-0032 東京都港区六本木 3 丁目 2-31
AP 事業所
IBM World Trade Asia Corporation
Intellectual Property Law & Licensing

以下の保証は、国または地域の法律に沿わない場合は、適用されません。 IBM およびその直接または間接の子会社は、本書を特定物として現存するままの状態で提供し、商品性の保証、特定目的適合性の保証および法律上の瑕疵担保責任を含むすべての明示もしくは黙示の保証責任を負わないものとします。国または地域によっては、法律の強行規定により、保証責任の制限が禁じられる場合、強行規定の制限を受けるものとします。

本書に含まれる情報には、技術的に不正確なもの、または誤植が含まれる場合があります。これらに対する変更は、定期的に行われます。これらの変更は、資料の改訂版に含まれます。 IBM は、本書で説明している製品、プログラムに対して、予告なく改良、変更を加える場合があります。

本書において IBM 以外の Web サイトに言及している場合がありますが、便宜のため記載しただけであり、決してそれらの Web サイトを推奨するもので

はありません。それらの Web サイトにある資料は、この IBM 製品の資料の一部ではありません。それらの Web サイトは、お客様の責任でご使用ください。

IBM は、お客様が提供するいかなる情報も、お客様になんら義務も負わせない適切な方法で、使用もしくは配布することがあります。

本プログラムのライセンス保持者で、 (i) 独自に作成したプログラムとその他のプログラム (本プログラムを含む) との間での情報交換、および (ii) 交換された情報の相互利用を可能にすることを目的として、本プログラムに関する情報を必要とする方は、下記に連絡してください。

IBM Canada Limited
Office of the Lab Director
1150 Eglinton Ave. East
North York, Ontario
M3C 1H7
CANADA

本プログラムに関する上記の情報は、適切な条件の下で使用することができますが、有償の場合もあります。

本書で説明されているライセンス・プログラムまたはその他のライセンス資料は、 IBM 所定のプログラム契約の契約条項、IBM プログラムのご使用条件、またはそれと同等の条項に基づいて、 IBM より提供されます。

本書に含まれるパフォーマンス・データは、制御された環境下で決定されています。したがいまして、その他の稼働環境で得られる結果とは、かなり異なる可能性もあります。一部の測定値は、開発中のシステムを使用している場合があり、これらの測定値が一般的に提供可能なシステムで同様の数値になることを保証するものではありません。さらに、一部の測定値が推定されたものもあります。実測値と異なる場合があります。本書のユーザーは、使用される特定の環境での該当データを確認してください。

IBM 以外の製品については、当該製品の提供者から直接、出版されている資料または一般公開されている情報から入手しました。 IBM は、これらの製品についてはテストを行っておらず、これらの IBM 以外の製品に関する性能、互換性またはその他の主張について確認することはできません。 IBM 以外の製品の機能に対する質問は、それぞれの製品提供者にお問い合わせください。

IBM の将来の方向性または意図については、予告なしに変更または中止する場合があり、 IBM の目的および目標のみを示しているものです。

本書には、日常の業務処理で用いられるデータや報告書の例が含まれていますが、これは説明に具体性を与えるために記載されたものであり、それらの例には、個人、企業、ブランドの、あるいは製品などの名前が含まれている場合があります。それらの名前はすべて架空のものであり、また名称や住所が類似する企業が実在しても、それは偶然に過ぎません。

著作権：

本書に含まれる情報には、サンプル・アプリケーション・プログラムがソース言語の形式で含まれており、様々な、オペレーティング・プラットフォームでのプログラミング技法を示しています。お客様は、これらのサンプル・プログラムが書かれているオペレーティング・プラットフォームでアプリケーション・プログラミング・インターフェースが実行可能となるためのアプリケーション・プログラムを開発、使用、販売または配布もしくは転送する目的のためにのみ、サンプル・プログラムを、IBM に対する別途料金を支払うことなく、複製、変更、配布または転送することができます。これらのサンプルは、すべての条件下で十分にテストを行っていません。したがいまして、IBM は、これらのプログラムの信頼性、実用性または機能について、いかなる保証も負いません。

サンプル・プログラムまたはその改変版の複製物には、全部複製か部分複製かを問わず、次の著作権表示を必ず行うものとします。

商標

次のものは、IBM Corporation の米国およびその他の国における商標です。

ACF/VTAM
AISPO
AIX
AIX/6000
AIXwindows
AnyNet
APPN
AS/400
BookManager
CICS
C Set++
C/370
DATABASE 2
DataHub
DataJoiner
DataPropagator
DataRefresher
DB2
DB2 Connect
DB2 Extenders
DB2 OLAP Server
DB2 Universal Database
Distributed Relational Database Architecture
DRDA
eNetwork
Extended Services
FFST
First Failure Support Technology
IBM
IMS
IMS/ESA
LAN DistanceMVS
MVS/ESA
MVS/XA
Net.Data
OS/2
OS/390
OS/400
PowerPC
QBIC
QMF
RACF
RISC System/6000
RS/6000
S/370
SP
SQL/DS
SQL/400
System/370
System/390
SystemView
VisualAge
VM/ESA
VSE/ESA
VTAM
WebExplorer
WIN-OS/2

次のものは、他社の商標または登録商標です。

Tivoli および NetView は、米国およびその他の国における Tivoli Systems Inc. の商標です。

Microsoft、Windows、Windows NT、および Windows ロゴは、 Microsoft Corporation の米国およびその他の国における商標です。

Java およびすべての Java 関連の商標およびロゴは Sun Microsystems, Inc. の米国およびその他の国における商標または登録商標です。

UNIX は、The Open Group がライセンスしている米国およびその他の国における登録商標です。

他の会社名、製品名およびサービス名等はそれぞれ各社の商標または登録商標です。

用語集

A

管理者 (administrator). 情報カタログ・マネージャーの内容および使用について管理する責任のある人。

アンカー (anchor). 別のオブジェクトを含むが、他の Grouping オブジェクトには含まれない Grouping オブジェクト。

Attachment. 別の情報カタログ・マネージャー・オブジェクトに追加情報を付加するのに使用されるオブジェクト・タイプの区分。たとえば、ユーザーはオブジェクトにコメントを付加することができる。

B

ブラウズ (browse). 主題ごとにグループ化された情報カタログ・オブジェクトを表示すること。*探索 (search)* と対比。

C

カタログ (catalog). *情報カタログ (information catalog)* およびデータベース・カタログ *(database catalog)* を参照。

区分 (category). 情報カタログ・マネージャー・オブジェクト・タイプの種別。区分は次のことを示す。

- オブジェクト・タイプが使用することのできるアクションのタイプ
- 同一または別の区分に属するオブジェクト・タイプ間で受け入れられる関係

オブジェクト・タイプは、次の区分のいずれかに置かれる。

Attachment
Contact
Dictionary
Elemental
Grouping
Program
Support

CelDial サンプル・データ (CelDial sample data). 情報カタログ・マネージャーを導入するときに使用できるサンプルの情報カタログ (ICMSAMP) で、導入検査で使用できる。このサンプル情報カタログは、*情報カタログ・マネージャー 使用者の手引き* の練習でも使用される。

集合 (collection). オブジェクトのコンテナー。集合を使用して関心のあるオブジェクトを集め、アクセスしやすくすることができる。

Comments. オブジェクトの種別で、情報カタログ・マネージャーの別のオブジェクトに注釈をする。たとえば、図表内のデータについての注を含める場合、図表オブジェクトに Comments オブジェクトを付加することもある。

Comments オブジェクト・タイプは情報カタログ・マネージャーに付属している。特性を追加することはできない。

コミット (commit). 情報カタログ・データベースに対して行われた変更を永続的なものにすること。*ロールバック (roll back)* と対比。

連絡先 (contact). オブジェクトの詳細についての参照先。詳細には、オブジェクトが表す情報を作成した人や情報を管理する担当部門が含まれている。

Contact. Contact オブジェクト・タイプおよび連絡を識別する他のオブジェクト・タイプの区分。

Contact オブジェクト・タイプ (Contact object type). 連絡担当者を識別するオブジェクトの種別。

D

データベース・カタログ (database catalog). 表、ビュー、索引などのデータベース・オブジェクトの記述を含む、表の集合。

DBCS. 2 バイト文字セット (double-byte character set)。

意思決定支援システム (decision-support system). ユーザーの意思決定を援助するアプリケーションからなるシステム。ユーザーは、この種のシステムを使用することにより、スプレッドシート、図表、報告書などのような意味のある方法で提示された情報を利用することができる。

削除履歴 (delete history). 削除活動のログ。そのログを取り込むかどうかは、情報カタログ・マネージャー管理者によってオンにされたりオフにされたりする。ログはタグ言語ファイルに転送できる。

派生データ (derived data). オペレーショナル・データ・ソースからのコピーまたは (データの要約などにより) 強化されて情報データベースに入れられたデータ。

記述データ (descriptive data). 表の名前、スプレッドシートの場所、文書の作成者などのような、オブジェクトを識別および記述するデータ。メタデータとも呼ばれる。

記述ビュー (Description view). あるオブジェクトの特性および特性値をリストするビュー。

Dictionary. 用語の定義で使用できるオブジェクト・タイプの区分 (たとえば、サンプル情報カタログでの “Glossary entries” オブジェクト・タイプ)。

辞書機能 (dictionary facility). 情報カタログで使用するビジネス用語の定義または同義語のコレクション。これを作成したら、保存された検索アイコンとして、辞書機能が各ユーザーの「カタログ (Catalog)」ウィンドウに表示される。

2 バイト文字セット (double-byte character set (DBCS)). 各文字が 2 バイトで表現される文字のセット。日本語、中国語、および韓国語のように、256 のコード点では表現しきれない数の記号を含む言語には、2 バイト文字セットが必要となる。 *1 バイト文字セット (single-byte character set)* と対比。

DP 名 (DP NAME). インポート操作のためにオブジェクト・タイプを固有に識別する、オブジェクト・タイプの識別子。オブジェクト・タイプの省略名とも呼ばれる。

E

エコー・ファイル (echo file). タグ言語ファイルをインポートする際に情報カタログ・マネージャーが作成するファイル。このファイルには、タグ言語ファイルの先頭から、または最後の COMMIT タグが処理されたあとで処理されたすべてのタグが収められる。

Elemental. その他の情報カタログ・マネージャー・オブジェクト・タイプの構築ブロックである Grouping オブジェクト・タイプ以外の区分。Elemental オブジェクト・タイプは、オブジェクト・タイプ階層の最下位に位置する。 “Columns in relational tables”、 “Presentations {electronic and hardcopy}”、および “Graphics and Images” は、すべて Elemental オブジェクト・タイプの例。

エクスポート (export). 情報カタログ・マネージャーからメタデータをコピーし、そのメタデータをタグ言語に変換し、あとでインポート操作を行うときのためにこの出力をタグ言語ファイルに収めること。

外部名 (external name). オブジェクト・タイプを表す 80 バイトの名前。オブジェクト・タイプ名とも呼ばれる。

抽出制御ファイル (extract control file). 抽出ユーティリティー・プログラムの操作を制御するステートメントを含むファイル。

抽出プログラム (extract program). RDBMS カタログなどのメタデータ・ソースからのコピーを行い、そのメタデータをタグ言語に変換し、この出力をタグ言語ファイルに収めるユーティリティー。

F

FAT. ファイル割り振りテーブル (file allocation table)。ファイル用のスペースをディスク上で割り振るため、およびそのファイルの位置を探すために使用される表。

FLGID. *オブジェクト識別子 (object identifier)* を参照。

G

Grouping. 他のオブジェクト・タイプに含めることのできるオブジェクト・タイプの区分。情報カタログ・マネージャーに付属するサンプル情報カタログで使える Grouping オブジェクト・タイプの例は、 Elemental オブジェクト・タイプ “Columns in relational tables” を含む “Tables or views in a relational database”、そして別の Grouping オブジェクト・タイプ “Dimension” を含む “Multi-dimensional model”。

H

HPFS. ハイ・パフォーマンス・ファイル・システム (High-performance file system)。 OS/2 において、キャッシュと呼ばれる高速バッファー記憶装置を使用して大規模なディスク・ボリュームに高速でアクセスする、導入可能なファイル・システム。 HPFS では、254 文字までのファイル名を使用することができる。

I

インポート (import). タグ言語ファイルの内容を情報カタログ・マネージャーに適用して、情報カタログを初期移植したり、情報カタログの内容を変更したり、別の情報カタログの内容を情報カタログにコピーしたりすること。

情報カタログ (information catalog). 組織内でユーザーが使用することのできるデータや情報の識別および検出を行いやすくするための記述データが入った、情報カタログ・マネージャーによって管理されるデータベース。

情報カタログ・マネージャー・アプリケーション・プログラム・インターフェース (Information Catalog Manager application program interface (API)). 情報カタログ・マネージャーの一部であって、情報カタログ・マネージャーのサービスおよび機能に関するアプリケーション・プログラム要求を処理する部分。

情報源 (information source). 情報カタログ・マネージャー・オブジェクトによって表現される、表や図表などのデータまたは情報の項目。

情報アプリケーション (informational application). ユーザーがデータを検索および分析するために使用される、プログラムまたはシステム。

情報データベース (informational database). 派生データを含む、ビジネス上の意思決定に使用するためのデータベース。

入力構造 (input structure). 情報カタログ・マネージャー・アプリケーション・プログラム・インターフェースにデータを実行依頼するために使用される、自己定義データ構造。

インスタンス (instance). *オブジェクト (object)* を参照。

インスタンス識別子 (instance identifier). 情報カタログ・マネージャーによって生成される、各オブジェクトを表す 10 桁の数字による識別子。この識別子は特定のオブジェクト・タイプ内ではそのオブジェクトに固有であり (別のオブジェクト・タイプのオブジェクトと同じ識別子を使用してもよい)、また特定の情報カタログ・データベース内でも固有である (別の情報カタログ・データベース内のオブジェクトと同じ識別子を使用してもよい)。

入出力構造 (I/O structure). *入力構造 (input structure)* および*出力構造 (output structure)* を参照。

K

キーワード (keyword). 情報カタログとの間でインポートまたはエクスポートされるデータ値の意味を識別する、情報カタログ・マネージャー・タグ言語の要素。

キーワード探索 (keyword search). *探索 (search)* を参照。

L

リンク (link). リンク関係にある複数のオブジェクト間の接続。

リンク関係 (linked relationship). 情報カタログのオブジェクト間の関係。リンク関係にあるオブジェクトは、一方がもう一方の基本オブジェクトというより、お互いが対等である。

たとえば、情報カタログ・マネージャーに付属のサンプル情報カタログでは、**CelDial Sales Information** というオブジェクトは、その年の CelDial 広告を表すさまざまなオブジェクトとリンクしている。

ログ・ファイル (log file). タグ言語ファイルをインポートするとき、または情報カタログにオブジェクトをエクスポートするときに情報カタログ・マネージャーが作成するファイル。このファイルには、インポートまたはエクスポートが開始または終了した日付と時刻、およびその処理に関するエラー情報が記録される。

M

メタデータ (metadata). 情報源に関するデータ。*記述データ (descriptive data)* を参照。

複数文字のワイルドカード (multiple character wildcard). 任意の長さの一連の文字を表すために使用される文字。デフォルトでは、複数の文字のワイルドカードはアスタリスク (*) である。*ワイルドカード (wildcard)* および *1 文字のワイルドカード (single character wildcard)* も参照。

N

非適用記号 (non-applicable symbol). オブジェクトの作成時に必須特性の値が提供されなかったことを示す文字。非適用記号は、デフォルトではハイフン (-) になるが、情報カタログの作成時に別の記号を指定している。

O

オブジェクト (object). 情報の単位または個別のグループ化を表す項目。各情報カタログ・マネージャー・オブジェクトは情報を識別し、記述するが、実際の情報は含まれない。たとえば、あるオブジェクトで報告書の名前を指定し、その作成日付を示し、その目的を記述することができる。

オブジェクト識別子 (object identifier). 6 桁のオブジェクト・タイプ識別子と、10 桁のインスタンス識別子からなる、オブジェクトを表す 16 桁の識別子で、いくつかの API 呼び出しで使用

される。*オブジェクト・タイプ識別子 (object type identifier)* および*インスタンス識別子 (instance identifier)* を参照。

オブジェクト・タイプ (object type). オブジェクトの種別。オブジェクト・タイプは、表、報告書、またはイメージなどのビジネス情報のタイプを表すために使用される。

情報カタログ・マネージャーは、修正可能な一連のオブジェクト・タイプを提供する。追加のオブジェクト・タイプを作成し、ユーザーの編成要件に合わせることもできる。

オブジェクト・タイプ識別子 (object type identifier). 情報カタログ・マネージャーによって生成される、各オブジェクト・タイプを表す 6 桁の数字による識別子。この識別子は、情報カタログ内で固有である。

オブジェクト・タイプ登録 (object type registration). 情報カタログ・マネージャー・アプリケーション・プログラム・インターフェースでは、オブジェクト・タイプに関する基本的な情報のことで、そのオブジェクト・タイプの特性を定義する前に情報カタログ・マネージャー内で定義しておく必要がある。これらの情報には、区分、名前、アイコン、およびオブジェクト情報を含む表の名前が含まれる。

オペレーショナル・データ (operational data). ある組織の日常業務を遂行するために使用されるデータ。

オプション (option). 情報カタログ・マネージャー・タグ言語では、タグ言語ファイルのインポート時に情報カタログ・マネージャー・データベース内のオブジェクトまたはオブジェクト・タイプに関して実行すべきアクションを定義する、ACTION タグのパラメーター。

出力構造 (output structure). 情報カタログ・マネージャー API 呼び出しによって作成されたデータを戻す際に情報カタログ・マネージャーが作成する自己定義データ構造。

P

物理タイプ名 (physical type name). 情報カタログ・データベースにおける、特定オブジェクト・タイプのインスタンスに関するメタデータが入っている表の名前。

移植 (populate). 情報カタログ・マネージャーにオブジェクト・タイプ、オブジェクト、またはメタデータを追加すること。

Program 区分 (Program category). Programs オブジェクト・タイプの区分。

Programs オブジェクト・タイプ (Programs object type). 情報カタログ・マネージャー・オブジェクトによって書かれた実際の情報を処理できるアプリケーションを識別し、説明するオブジェクトの識別。

Programs オブジェクト・タイプは情報カタログ・マネージャーに付属している。

特性 (property). 情報の単位を記述する特徴または属性。各オブジェクト・タイプには、関連する特性のセットがある。サンプル情報カタログの “Graphics and Images” オブジェクト・タイプには、以下のような特性がある。

名前
説明
イメージ・タイプ
イメージ・ファイル名

各オブジェクトごとに、一連の値が特性に割り当てられる。

特性名 (property name). 情報カタログ・マネージャー・ユーザー・インターフェースで表示される特性を記述する 80 バイトの名前。*特性省略名 (property short name)* と対比。

特性省略名 (Property short name). オブジェクトまたはオブジェクト・タイプの特性を固有に識別するために情報カタログ・マネージャーによって使用される 8 文字の名前。

特性値 (property value). 特性の値。

PT 名 (PT NAME). *物理タイプ名 (physical type name)* を参照。

R

RDBMS. 関係データベース管理システム (relational database management system)。

RDBMS カタログ (RDBMS catalog). 表、ビュー、および索引などの、RDBMS によって維持される SQL オブジェクトの記述を含む表の集合。

関係データベース管理システム (relational database management system). 関係データを管理し保管する、DB2 UDB (OS/2 版) などのソフトウェア・システム。

登録 (registration). *オブジェクト・タイプ登録 (object type registration)* を参照。

ロールバック (roll back). 情報カタログ・データベースに対して行われたコミットされていない変更を除去すること。*コミット (commit)* と対比。

S

保管済み探索 (saved search). あとで使用するために保管された探索基準の集合。「カタログ (Catalog)」ウィンドウにアイコンとして表示される。

SBCS. 1 バイト文字セット (single-byte character set)。

探索 (search). 特定の基準を満たす情報カタログ・マネージャー・オブジェクトを表示するように要求すること。

主題で探索 (search by subject). *ブラウズ (browse)* を参照。

用語で探索 (search by term). *探索 (search)* を参照。

探索基準 (search criteria). 探索の実施方法を指定するために使用されるオプションおよび文字列。これには、オブジェクト・タイプ名、特性値、完全に一致するものを探索するのか、大文字・小文字を区別して探索するのか、などの情報を含めることができる。

1 バイト文字セット (single-byte character set (SBCS)). 各文字が 1 バイト・コードで表現される文字セット。 *2 バイト文字セット (double-byte character set)* と対比。

1 文字のワイルドカード (single character wildcard). なんらかの単一文字を表すために使用される文字。デフォルトでは、1 文字のワイルドカードは疑問符 (?) である。*ワイルドカード (wildcard)* および*複数文字のワイルドカード (multiple character wildcard)* も参照。

主題探索 (subject search). *ブラウズ (browse)* を参照。

Support. ユーザーの情報カタログまたは企業についての追加情報を提供するオブジェクト・タイプの区分 (たとえば、サンプル情報カタログの “情報カタログ・マネージャー News” オブジェクト・タイプ)。

サポート機能 (support facility). 情報カタログへの変更や更新の情報など、情報カタログのユーザーに役立つと思われる情報の集合。これを作成したら、保存された検索アイコンとして、サポート機能が各ユーザーの「カタログ (Catalog)」ウィンドウに表示される。

T

タグ (tag). タグ言語の要素の 1 つ。タグでは、タグ言語ファイルが情報カタログにインポートされるときに取られるアクションが示される。

タグ言語 (tag language). データウェアハウスセンターまたは情報カタログ内のオブジェクト・タイプおよびオブジェクト、ならびにそれらのオブジェクト・タイプおよびオブジェクトに対して取られるアクションを定義するための形式。

タグ言語ファイル (tag language file). タグ言語を含むファイルで、ファイルのインポート時に、データウェアハウスセンターまたは情報カタログに追加、更新、または削除されたオブジェクトおよびオブジェクト・タイプを記述する。タグ言語ファイルを作成するには、データウェアハウスセンターまたは情報カタログ・マネージャーからオブジェクトをエクスポートする。

情報カタログ・マネージャーでは、以下の方法でもタグ言語ファイルを作成できる。
- 削除履歴ログを転送する。
- 抽出プログラムを使用して、別のデータベース・システムから記述データを抽出する。

ツリー・ビュー (Tree view). あるオブジェクトと、それに含まれるオブジェクトを階層的に表示するビュー。

U

作業単位 (unit of work). あるアプリケーション・プロセスにおける回復可能な一連の操作。作業単位は、データベースを矛盾のない状態にするためにデータベース管理システムが使用する基本構築ブロックである。作業単位は、データベースに対する変更がコミットまたはロールバックされたときに終了する。

汎用固有識別コード (universal unique identifier (UUI)). オブジェクトのキー。このキーは最大 5 つまでの特性からなり、これらの特性が指定された順序で連結されることにより、機能のインポートおよびエクスポート時にオブジェクトが固有に識別される。

ユーザー (user). 情報カタログで使用可能な情報にアクセスする人。ただし、管理者ではない。権限を得ている情報カタログ・マネージャー・ユーザーは、通常は管理者によって実行されるオブジェクト管理タスクを実行することができる。

W

ワイルドカード (wildcard). 探索の特性値を指定するときに変数として使用される特殊文字。 *1 文字のワイルドカード (single character wildcard)* および*複数文字のワイルドカード (multiple character wildcard)* も参照。

参考文献

ここに示す資料のコピーの入手、あるいは特定ライブラリーに関する詳しい説明を希望される場合には、IBM 担当員にご連絡ください。

データウェアハウスセンターの資料

ウェアハウス・マネージャー インストールの手引き (GC88-8572)

情報カタログ・マネージャー 管理の手引き (SC88-8547)

データウェアハウスセンター 管理の手引き (SC88-8545)

情報カタログ・マネージャー 使用者の手引き (SC88-8548)

IBM DB2 ユニバーサル・データベース メッセージ解説書 (GC88-8543)

参考文献

索引

日本語, 数字, 英字, 特殊文字の順に配列されています。なお, 濁音と半濁音は清音と同等に扱われています。

［ア行］

アイコン
  管理 198
アプリケーション・サポート
  FLGFreeMem 17
  FLGInit 17
  FLGTerm 17
  FLGTrace 17
アプリケーション・プログラム 2, 4
アンカーのリスト 161
移植、情報カタログ・マネージャーの 150, 212
インクルード・ファイル 267
インスタンス、オブジェクト・タイプの 6
インポート、メタデータの 21, 150, 212
エクスポート、情報カタログ・マネージャー・メタデータの 124, 209
エクスポート、メタデータの 21
エラー回復 28, 245
オーファン、リストの 180
オブジェクト
  概要 5, 6
  追加 4
  分類 6
オブジェクト域
  サンプル・コードの定義 55
  出力構造 63
  入力構造 43, 52
オブジェクト区分 5
オブジェクトの関係 19
オブジェクト・インスタンス 6
オブジェクト・インスタンス 6 *(続き)*
  管理
    FLGCreateInst 19
    FLGDeleteInst 19
    FLGDeleteTree 19
    FLGGetInst 19
    FLGUpdateInst 19
  検索 227, 237
  このオブジェクトを含むオブジェクトのリスト 259
  コピー
    FLGExport 21
    FLGImport 21
  削除 108
    グループ 113
  作成 87
  情報の検索 139
  情報の更新 248
  他のインスタンスの検出 131
  探索
    FLGSearch 20
    FLGSearchAll 20
  リスト
    FLGFoundIn 20
    FLGListAnchors 20
    FLGListAssociates 20
    FLGListContacts 20
    FLGListOrphans 20
    FLGListPrograms 20
    FLGNavigate 20
    FLGWhereUsed 20
オブジェクト・タイプ
  概要 7
  関係 6
  管理
    FLGAppendType 18
    FLGCreateType 18
    FLGDeleteType 18
    FLGDeleteTypeExt 18
    FLGGetType 18
オブジェクト・タイプ *(続き)*
  区分の指定 11
  コピー
    FLGExport 21
    FLGImport 21
  削除 118
  削除、インスタンスの 121
  作成 101
  情報の検索 147
  すべてのリスト 177
  定義 8
  登録 7
  特性の追加 77
  必須特性の定義 11
  用語 14
  リスト
    FLGListObjTypes 20
  API 呼び出しによる作成 30
オブジェクト・タイプ登録
  管理
    FLGCreateReg 18
    FLGDeleteReg 18
    FLGGetReg 18
    FLGManageIcons 18
    FLGUpdateReg 18
  削除 111
  作成 93
  情報の検索 143
  情報の更新 254
  必須特性 9

［カ行］

開始
  情報カタログ・マネージャー 154
  プログラム
    FLGOpen 22, 219
    FLGOpen による 30
    HPFS ファイルの考慮事項 29

開始 *(続き)*
STARTCMD 特性 29
HPFS ファイル名の 29
開始、プログラムの
外部 219
ワークステーション 22
Programs オブジェクトの設定 29
回復、エラーからの 28
解放、出力構造の記憶域の 137
概要 1
獲得、情報の
オブジェクト・インスタンスに関する 139
オブジェクト・タイプ登録に関する 143
オブジェクト・タイプに関する 147
各国語の考慮事項 32
関係 221
オブジェクト・タイプ間の 6
関数原型、DG2API.H における 278
管理
アイコン 198
オブジェクトの関係 19
オブジェクト・インスタンス 19
オブジェクト・タイプ 18
オブジェクト・タイプ登録 18
コメント状況 190
削除履歴ログ 201
情報カタログ・マネージャー識別子
FLGConvertID 19
情報カタログ・マネージャー・ユーザー 203
データベース、企業
FLGManageCommentStatus 23
FLGManageFlags 23
FLGManageTagBuf 23
FLGManageUsers 23
FLGXferTagBuf 23
管理者 2
関連、リストの 164
関連資料 399
記述データ 1
記録、エラー状態の 245
区分
Attachment
他の区分との関係 6
定義 6
Contact
他の区分との関係 6
定義 5
Dictionary
他の区分との関係 6
定義 5
Elemental
他の区分との関係 6
定義 5
Grouping
他の区分との関係 6
定義 5
Program
他の区分との関係 6
定義 5
Support
他の区分との関係 6
定義 6
区分、オブジェクトの 5
区分、メタデータの 5
限界値 283
検索
オブジェクト・インスタンスに関する情報 139
オブジェクト・タイプ登録に関する情報 143
オブジェクト・タイプに関する情報 147
検索、オブジェクト・インスタンスの 227, 237
検索、含まれるオブジェクトのリストの 215
検出
他のインスタンス内のオブジェクト・インスタンス 131
コード、理由 285
更新、オブジェクト・インスタンスに関するメタデータの 248
更新、オブジェクト・タイプ登録情報の 254
構造
一般的な特性 35
構造 *(続き)*
出力形式 56
入力形式 36
構文、API 呼び出しの 75
構文図、読み方 75
コピー、オブジェクト・インスタンスの 21
コピー、オブジェクト・タイプの 21
コミット、情報カタログ・マネージャー・情報カタログに対する変更内容の 82
コミット、データベースに対する変更の
FLGCommit 22
コメント
現況の選択項目
リストの設定 190
コンパイル、C 言語プログラムの
Windows で 27
コンパイルと連係、サンプル・プログラムの 265

［サ行］

最大値、情報カタログ・マネージャーの 283
削除
オブジェクト・インスタンス 108
グループ 113
オブジェクト・タイプ 118
オブジェクト・タイプとインスタンスの 121
オブジェクト・タイプ登録 111
削除活動
ロギング
使用可能にする 195
使用不能にする 195
ログ
照会 201
タグ・ファイルへの転送 263
リセット 201
削除履歴
ロギング
使用可能にする 195
使用不能にする 195

削除履歴 *(続き)*
ログ
照会 201
タグ・ファイルへの転送 263
リセット 201
作成
オブジェクト・インスタンス 87
オブジェクト・タイプ 101
オブジェクト・タイプ登録 93
作成、API 呼び出しを使用するプログラムの 17
作成、C 言語のプログラムの 26
サポート、アプリケーションの
FLGFreeMem 17
FLGInit 17
FLGTerm 17
FLGTrace 17
サンプル・プログラム
コンパイルと連係 265
実行 266
入力構造の定義 53
DG2SAMP.C 33
識別名 14
実行、サンプル・プログラムの 266
終了、情報カタログ・マネージャーの 243
主題のリスト 161
出力構造
一般的な特性 35
オブジェクト域 63
概要 24
形式 56
定義域 60
特性の数の計算 66
ヘッダー域 58
戻された値の集合の数の計算 66
読み取り 65, 75
読み取りのためのサンプル・コード 69
API 呼び出しからの検索 25
DG2API.H で定義されている定数 267
DG2API.H の定義域 276
DG2API.H のヘッダー域 276
出力データ構造 56
初期設定、情報カタログ・マネージャーの 154
使用、情報カタログ・マネージャー API 呼び出しの 28
情報カタログ・マネージャー
オブジェクト 5
概要 1
限界値 283
設定、トレース・レベルの 245

［タ行］

タグ・ファイル
削除履歴の転送 201, 263
探索、オブジェクト・インスタンスの
オブジェクト名の使用 237
他のインスタンス内の 131
特性の使用 20
任意のオブジェクト・タイプの 237
1 つのオブジェクト・タイプでの 227
1 つまたは複数の特性の使用 227
追加
オブジェクト 4
オブジェクト・インスタンス 87
オブジェクト・タイプ 101
オブジェクト・タイプ登録 93
追加、オブジェクト・タイプへの特性の 77
データ
構造 36, 56
タイプ 31, 277
API 呼び出しでの受け渡し 24
データベースの保守 4
定義域
サンプル・コードの定義 54
出力構造 60
入力構造 39, 50
DG2API.H のデータ構造 276
停止、情報カタログ・マネージャーの 243
テンプレート、オブジェクトの 6
登録 7
特性、オブジェクト・タイプの 11
トレース (.TRC) ファイル 245
トレース、情報カタログ・マネージャー機能の 245

［ナ行］

名前、情報カタログ・マネージャーで使用されている 14
入力構造
一般的な特性 35
オブジェクト域 43
概要 24
形式 36
サイズの計算 46
作成 44
サンプル・コード
オブジェクト域の定義 55
定義域の定義 54
ヘッダー域の定義 53
定義
オブジェクト域 52
定義域 50
ヘッダー域 48
定義域 39
定義の例 53
ヘッダー域 37
API 呼び出しへの受け渡し 25
DG2API.H で定義されている定数 267
DG2API.H の定義域 276
DG2API.H のヘッダー域 276
入力データ構造 36

［ハ行］

パラメーター、API 呼び出しの 24
必須特性、オブジェクト・タイプの 11
プログラマー 2
プログラム
開始
FLGOpen 22
リスト 187
API 呼び出しによる作成 17
Programs オブジェクトの設定 29
プログラム、C 言語の 26
ヘッダー域
サンプル・コードの定義 53

ヘッダー域 *(続き)*
出力構造 58
入力構造 37, 48
DG2API.H のデータ構造 276
ヘッダー・ファイル 26, 267
変換
DP 名をオブジェクト・タイプ ID に 85
FLGID をオブジェクト・インスタンス名に 85
包含 221
保守、データベースの 4
翻訳された必須特性名 32, 157

［マ行］

メタデータ
区分 5
定義 1
有効なデータ・タイプ 31
API 呼び出しによる削除 30

［ヤ行］

ユーザー 2
用語、オブジェクト・タイプに関する 14
読み取り、出力構造の 65, 69

［ラ行］

リスト
アンカー・オブジェクト 161
オーファン・オブジェクト 180
オブジェクト・インスタンス 20
オブジェクト・タイプ 20, 177
関連オブジェクト 164
このオブジェクトを含む Grouping オブジェクト 259
主題オブジェクト 161
プログラム 187
Contact オブジェクト 173
例
読み取りのためのサンプル・コード 69
FLGAppendType API 呼び出し 80
例 *(続き)*
FLGCommit 83
FLGConvertID 86
FLGCreateInst 91
FLGCreateReg 98
FLGCreateType 105
FLGDeleteInst 110
FLGDeleteReg 112
FLGDeleteTree 115
FLGDeleteType 119
FLGDeleteTypeExt 122
FLGExport 129
FLGFoundIn 133
FLGFreeMem 138
FLGGetInst 141
FLGGetReg 145
FLGGetType 149
FLGImport 153
FLGInit 159
FLGListAnchors 162
FLGListAssociates 166
FLGListContacts 175
FLGListObjTypes 179
FLGListOrphans 182
FLGListPrograms 189
FLGManageCommentStatus 191
FLGManageFlags 196
FLGManageIcons 200
FLGManageTagBuf 202
FLGManageUsers 205
FLGNavigate 217
FLGOpen 220
FLGRelation 223
FLGRollback 225
FLGSearch 231
FLGSearchAll 240
FLGTerm 244
FLGTrace 246
FLGUpdateInst 251
FLGUpdateReg 257
FLGWhereUsed 261
FLGXferTagBuf 264
連係、C 言語プログラムの Windows で 27
ロールバック、情報カタログ・マネージャー情報カタログに対する変更の 225
ロールバック、データベースに対する変更の
FLGRollback 22
ロギング
削除活動
使用可能にする 195
使用不能にする 195
ログ
削除活動
照会 201
タグ・ファイルへの転送 263
リセット 201

A

API の構文 75
API 呼び出し
構文規則 75
呼び出しの構造 23
理由コード 285
DG2API.H における関数原型 278
FLGAppendType 77
FLGCommit 82
FLGConvertID 85
FLGCreateInst 87
FLGCreateReg 93
FLGCreateType 101
FLGDeleteInst 108
FLGDeleteReg 111
FLGDeleteTree 113
FLGDeleteType 118
FLGDeleteTypeExt 121
FLGExport 124
FLGFoundIn 131
FLGFreeMem 137
FLGGetInst 139
FLGGetReg 143
FLGGetType 147
FLGImport 150
FLGInit 154
FLGListAnchors 161
FLGListAssociates 164

API 呼び出し *(続き)*
FLGListContacts 173
FLGListObjTypes 177
FLGListOrphans 180
FLGListPrograms 187
FLGManageCommentStatus 190
FLGManageFlags 195
FLGManageIcons 198
FLGManageTagBuf 201
FLGManageUsers 203
FLGMdisExport 209
FLGMdisImport 212
FLGNavigate 215
FLGOpen 219
FLGRelation 221
FLGRollback 225
FLGSearch 227
FLGSearchAll 237
FLGTerm 243
FLGTrace 245
FLGUpdateInst 248
FLGUpdateReg 254
FLGWhereUsed 259
FLGXferTagBuf 263
Attachment 区分
関係
要約 6
定義 6

C

C 言語 26
CHAR データ・タイプ 31
Contact
オブジェクトの追加と除去 221
関係の作成と削除 221
リスト 173
Contact 区分
関係
要約 6
定義 5
CREATOR 特性 9

D

DBCS 文字、値における 33
DG2API.H
出力構造を読み取るための 66
定義 267
ヘッダー・ファイル 44
DG2SAMP.C 33, 265, 266
Dictionary 区分
関係
要約 6
定義 5
DOS バッチ・ファイル 29
DOS 文字ベース・プログラム 29
DPNAME 特性 9
OBJTYPID 特性への変換 85

E

Elemental 区分
定義 5

F

FLGAppendType
概要 7
API 呼び出し 77
FLGCommit 82
FLGConvertID 85
FLGCreateInst 87
FLGCreateReg
概要 7
API 呼び出し 93
FLGCreateType
概要 7
API 呼び出し 101
FLGDeleteInst 108
FLGDeleteReg
概要 7
API 呼び出し 111
FLGDeleteTree 113
FLGDeleteType
概要 7
API 呼び出し 118
FLGDeleteTypeExt 121
FLGExport 124
FLGFoundIn 131
FLGFreeMem 137
FLGGetInst 139
FLGGetReg
概要 7
API 呼び出し 143
FLGGetType
概要 7
API 呼び出し 147
FLGID
オブジェクト・インスタンス名への変換 85
FLGImport 150
FLGInit
プログラムの開始 28
API 呼び出し 154
FLGListAnchors 161
FLGListAssociates 164
FLGListContacts 173
FLGListObjTypes 177
FLGListOrphans 180
FLGListPrograms 187
FLGManageCommentStatus 190
FLGManageFlags 195
FLGManageIcons 198
FLGManageTagBuf 201
FLGManageUsers 203
FLGMdisExport 209
FLGMdisImport 212
FLGNavigate 215
FLGOpen
プログラムの開始 30
API 呼び出し 219
FLGRelation 221
FLGRollback 225
FLGSearch 227
FLGSearchAll 237
FLGTerm
プログラムの終了 28
API 呼び出し 243
FLGTrace 245
FLGUpdateInst 248
FLGUpdateReg
概要 7
API 呼び出し 254
FLGWhereUsed 259
FLGXferTagBuf 263

G

Grouping オブジェクト 215, 221
Grouping 区分
  関係
    要約 6
  定義 5

H

HANDLES 特性 29

I

INSTIDNT 特性 11

L

LIBPATH 27
LONG VARCHAR データ・タイプ 31

M

Microsoft Windows プログラム 29

N

NAME 特性 9, 11

O

OBJTYPID 特性 11
  DP 名の変換 85
OS/2 29

P

PARMLIST 特性 29
Program 区分
  関係
    要約 6
  定義 5
PTNAME 特性 9

S

SBCS 文字、値における 33
SET INCLUDE 27
SET LIB 27
STARTCMD 特性 29
Support 区分
  関係
    要約 6
  定義 6

T

TIMESTAMP データ・タイプ 31

U

UPDATEBY 特性 9, 11
UPDATIME 特性 9, 11

V

VARCHAR データ・タイプ 31

W

Windows ヘッダー・ファイル 267

［特殊文字］

#define ステートメント 26
#define ステートメント、DG2API.H の 267
#include ステートメント 26

プログラム番号: 5648-D35

Printed in Japan

SC88-8549-00

日本アイ・ビー・エム株式会社
〒106-8711　東京都港区六本木3-2-12

Spine information:

DB2® ウェアハウス・マネージャー

情報カタログ・マネージャー プログラミングの手引きおよび解説書

バージョン 7