

Adabas for Linux, UNIX and Windows

Adabas Basics

バージョン 6.6

2017 年 10 月

このマニュアルは Adabas for Linux, UNIX and Windows バージョン 6.6 およびそれ以降のすべてのリリースに適用されます。

このマニュアルに記載される仕様は変更される可能性があります。変更は以降のリリースノートまたは新しいマニュアルに記述されます。

Copyright © 1987-2017 Software AG, Darmstadt, Germany and/or Software AG USA, Inc., Reston, VA, United States of America, and/or their licensors.

The name Software AG, webMethods and all Software AG product names are either trademarks or registered trademarks of Software AG and/or Software AG USA, Inc. and/or their licensors. Other company and product names mentioned herein may be trademarks of their respective owners.

Software AG およびその子会社が所有する登録商標および特許の詳細については、<http://documentation.softwareag.com/legal/> を確認してください。

本ソフトウェアの一部にはサードパーティ製製品が含まれています。サードパーティの著作権表示およびライセンス規約については『License Texts, Copyright Notices and Disclaimers of Third-Party Products』を参照してください。このドキュメントは製品ドキュメントセットの一部であり、<http://documentation.softwareag.com/legal/> 上、またはライセンス製品のルートインストールディレクトリ内にあります。

本ソフトウェアの利用は、Software AG のライセンス規約に則って行われるものとします。ライセンス規約は製品ドキュメントセット内、<http://documentation.softwareag.com/legal/> 上、またはライセンス製品のルートインストールディレクトリ内にあります。

ドキュメント IDは: **ADAOS-BASICS-66-20200619JA**

目次

Adabas Basics	v
1	1
表記規則	2
オンライン情報	2
データ保護	3
2 データベース設計	5
Adabas の技術の概要	6
システム設計におけるパフォーマンス制御	8
Unicode のサポート	11
ファイルおよびレコード設計	12
Adabas システムファイル	17
データアクセス手法	19
ディスクスペースの使用法	27
セキュリティ計画	30
Adabas のセキュリティ機能の概要	31
トランザクションのコンセプト	32
リカバリ／再スタート設計	34
3 コンテナファイル	41
一般的な知識	42
Adabas 論理エクステンツ	43
Adabas 物理エクステンツ	44
Adabas 物理エクステンツ	44
コンテナファイルのアクセスメソッド	46
Adabas ブロックサイズ	47
データベース自動拡張	49
インデックスブロックサイズ	50
SORT データセットの格納先	50
TEMP データセットの格納先	51
ファイルシステムまたは RAW デバイス内のコンテナファイル	51
4 一時ワークスペース	53
5 FDT のレコード構造	57
データ定義構文	58
定義オプション	61
サブディスクリプタ	78
スーパーディスクリプタ	81
フォネティックディスクリプタ	89
ハイパーディスクリプタ	90
照合ディスクリプタ	93
参照制約	96
6 ディスクリプタの定義	99
ADAINV の処理に関する考慮事項	100
7 ユーティリティの使用	103
入出力デバイスの割り当て	104

ユーティリティの実行 (UNIX)	104
ユーティリティの実行 (Windows)	108
ユーティリティのリモート実行	111
ユーティリティ構文	111
単一機能ユーティリティと多機能ユーティリティ	114
ユーティリティの終了	114
エラー処理	115
Adabas シーケンシャルファイル	115
ADAMUP および ADAINV の実行の最適化	124
ニュークリアスとユーティリティの間の同期	126
8 データのロードとアンロード	129
はじめに	130
他のハードウェアアーキテクチャへのデータのコピー	131
非圧縮データフォーマット	132
ADACMP の入力データ要件	136
ADACMP の処理に関する考慮	143
ADAMUP の処理に関する考慮	144
ADABCK の処理に関する考慮	148
ADAORD の処理に関する考慮	151
ファイルのスペースの見積り	154
9 ユーザー出口とハイパー出口	163
ユーザー出口の概要	164
ユーザー出口の説明	165
ハイパー出口の概要	191
ハイパー出口コントロールブロックとバッファ	193
ハイパー出口インターフェイス	202
ユーザー出口およびハイパー出口の作成と定義	206
10 読み取り専用デバイス上の Adabas	213
Adabas ニュークリアス使用時の制限	214
Adabas ユーティリティ使用時の制限	215

Adabas Basics

このドキュメントには、Adabas データベースに関する基本情報が含まれています。これには、データベース設計、Adabas データベースを構成するコンポーネント、フィールド定義テーブル、使用可能な定義オプション、データベースへのデータのロードとデータベースからのデータのアンロード、および Adabas ユーティリティの使用に関する情報が含まれます。

次のトピックについて説明します。

- **データベース設計**：データベース設計について説明します。ここでは、Adabas ファイルの構造、マルチプルバリューフィールドとピリオディックグループ、レコード設計、キー（ディスクリプタ）の使用法、ディスクスペースの利用法（圧縮、空値省略、パディングファクタ）、セキュリティプランニング（データのサイファリングを含む）、および再スタートとリカバリプランニングについて説明します。
- **コンテナファイル**：Adabas コンテナファイルについて説明します。
- **一時ワークスペース**：Adabas ニュークリアスやユーティリティが必要とする一時ワークスペースの割り当てや削除について説明します。
- **FDT のレコード構造**：データベース内でのファイルレコード構造の定義方法について説明します。
- **ディスクリプタの定義**：ファイルディスクリプタの作成と削除の方法について説明します。
- **データのロードとアンロード**：データベースへのデータのロードおよびデータベースからのデータのアンロードに関するさまざまな方法について説明します。
- **ユーティリティの使用**：Adabas ユーティリティを使った作業方法に関する一般的な情報について説明します。
- **ユーザー出口とハイパー出口**：Adabas がサポートするユーザー出口とハイパー出口について説明します。
- **読み取り専用デバイス上の Adabas**：読み取り専用デバイスで Adabas を実行する方法、またそのような環境での制限について説明します。

1

■ 表記規則	2
■ オンライン情報	2
■ データ保護	3

表記規則

規則	説明
太字	画面上の要素を表します。
モノスペースフォント	<code>folder.subfolder:service</code> という規則を使用して webMethods Integration Server 上のサービスの保存場所を表します。
大文字	キーボードのキーを表します。同時に押す必要があるキーは、プラス記号 (+) で結んで表記されます。
斜体	独自の状況または環境に固有の値を指定する必要がある変数を表します。本文で最初に出現する新しい用語を表します。
モノスペースフォント	入力する必要があるテキストまたはシステムから表示されるメッセージを表します。Program code.
{ }	選択肢のセットを表します。ここから1つ選択する必要があります。中カッコの内側にある情報のみを入力します。{} 記号は入力しません。
	構文行で相互排他的な2つの選択肢を区切ります。いずれかの選択肢を入力します。 記号は入力しません。
[]	1つ以上のオプションを表します。大カッコの内側にある情報のみを入力します。[] 記号は入力しません。
...	同じ種類の情報を複数回入力できることを示します。情報だけを入力してください。実際のコードに繰り返し記号 (...) を入力しないでください。

オンライン情報

Software AG マニュアルの Web サイト

マニュアルは、Software AG マニュアルの Web サイト (<http://documentation.softwareag.com>) で入手できます。このサイトでは Empower クレデンシャルが必要です。Empower クレデンシャルがない場合は、TECHcommunity Web サイトを使用する必要があります。

Software AG Empower 製品のサポート Web サイト

もしまだ Empower のアカウントをお持ちでないのなら、こちらへ empower@softwareag.com 電子メールにて あなたのお名前、会社名、会社の電子メールアドレスをお書きの上、アカウントを請求してください。

いったんアカウントをお持ちになれば、Empower <https://empower.softwareag.com/> の eService セクションにてサポートインシデントをオンラインで開くことができます。

製品情報は、Software AG Empower 製品のサポート Web サイト (<https://empower.softwareag.com>) で入手できます。

機能および拡張機能に関するリクエストの送信、製品の可用性に関する情報の取得、製品ダウンロードを実行するには、Products に移動します。

修正に関する情報を取得し、早期警告、技術論文、Knowledge Base の記事を読むには、[Knowledge Center](#) に移動します。

もしご質問があれば、こちらのhttps://empower.softwareag.com/public_directory.asp グローバルサポート連絡一覧の、あなたの国の電話番号を選んで、わたくし共へご連絡ください。

Software AG TECHcommunity

マニュアルおよびその他の技術情報は、Software AG TECHcommunity Web サイト (<http://techcommunity.softwareag.com>) で入手できます。以下の操作を実行できます。

- TECHcommunity クレデンシアルを持っている場合は、製品マニュアルにアクセスできます。TECHcommunity クレデンシアルがない場合は、登録し、関心事の領域として [マニュアル] を指定する必要があります。
- 記事、コードサンプル、デモ、チュートリアルにアクセスする。
- Software AG の専門家によって承認されたオンライン掲示板フォーラムを使用して、質問したり、ベストプラクティスを話し合ったり、他の顧客が Software AG のテクノロジーをどのように使用しているかを学んだりすることが可能です。
- オープンスタンダードや Web テクノロジーを取り扱う外部 Web サイトにリンクできます。

データ保護

Software AG 製品は、EU 一般データ保護規則 (GDPR) を尊重した個人データの処理機能を提供します。該当する場合、適切な手順がそれぞれの管理ドキュメントに記載されています。

2 データベース設計

■ Adabas の技術の概要	6
■ システム設計におけるパフォーマンス制御	8
■ Unicode のサポート	11
■ ファイルおよびレコード設計	12
■ Adabas システムファイル	17
■ データアクセス手法	19
■ ディスクスペースの使用法	27
■ セキュリティ計画	30
■ Adabas のセキュリティ機能の概要	31
■ トランザクションのコンセプト	32
■ リカバリ／再スタート設計	34

Adabas の技術の概要

Adabas データベース内のデータにアクセスするプログラムは、Adabas コマンドを発行する必要があります（詳細については、『コマンドリファレンスセクション』を参照してください）。Adabas ダイレクトコールコマンドインターフェイスは、下位レベルのインターフェイスです。そのため、Software AG は、次のような Adabas への上位レベルのインターフェイスもいくつか提供しています。

- 開発環境 Natural：このプログラミング言語に、Adabas へのアクセスが統合されています。
- Adabas SQL Gateway：Adabas 用の SQL インターフェイスです。
- Adabas SOA Gateway：サービス指向アーキテクチャ（SOA）への Adabas インターフェイスです。

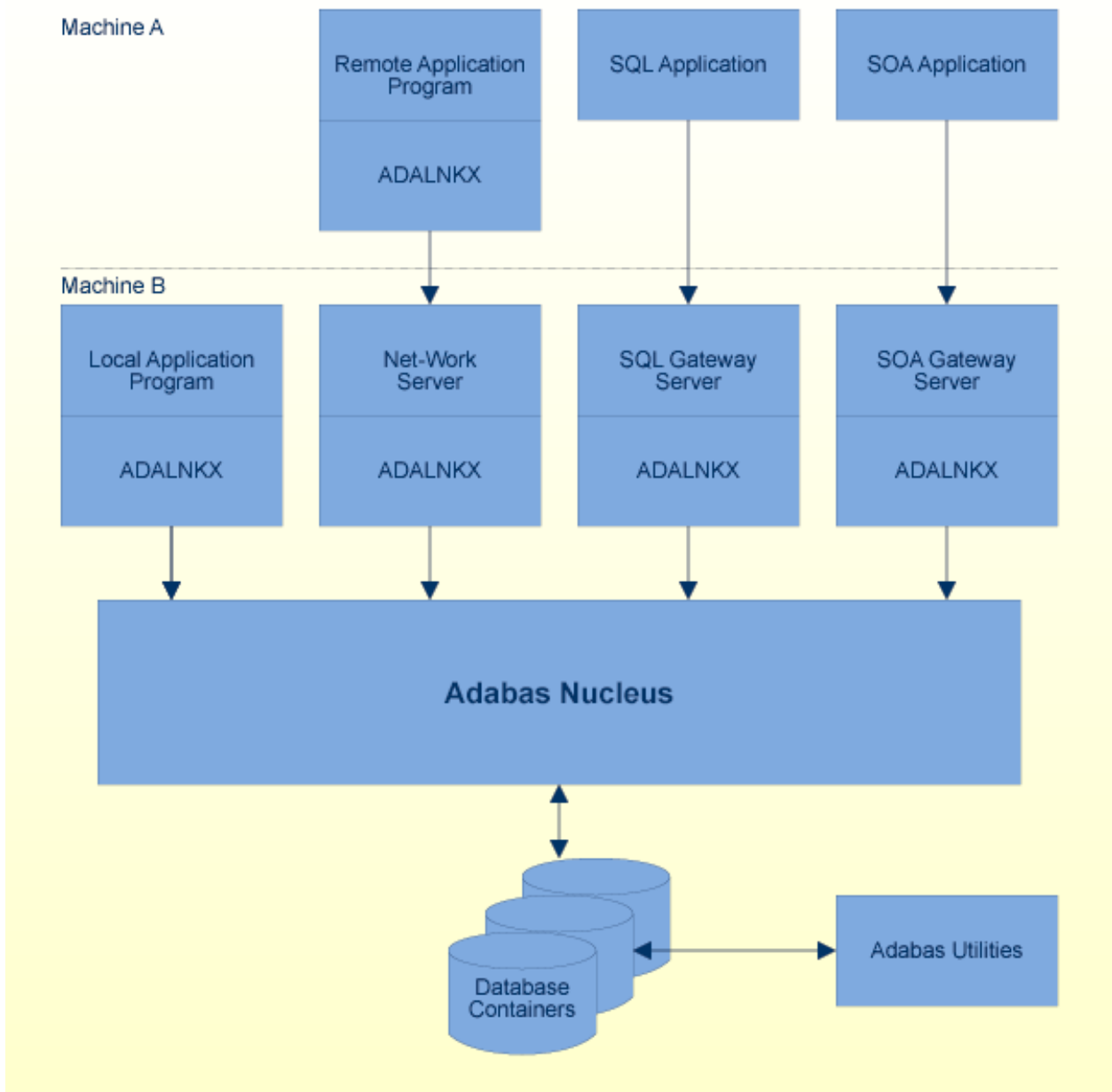
Adabas コマンドを発行するプログラムは、Adabas クライアントと呼ばれます。Adabas コマンドは、Adabas ニュークリアスと呼ばれるデータベースサーバーによって実行されます。

データベースにアクセスするには、Adabas クライアントを Adabas インターフェイスとリンクする必要があります。Adabas インターフェイスとして、例えば、Adabas クライアントパッケージに含まれている Adabas インターフェイスがあります。詳細については、『コマンドリファレンス』の「アプリケーションプログラムのリンク」を参照してください。

Adabas クライアントは、Adabas ニュークリアスと同じ（物理または仮想）マシンでローカルで実行することも、別のマシンでリモートから実行することもできます。リモートアクセスには、追加の製品 Entire Net-Work が必要です。

Adabas ニュークリアスに加えて、Adabas データベースにアクセスする、データベース管理用の Adabas ユーティリティも多数用意されています。

次の図は、プログラムが Adabas にアクセスする方法を示しています。



 **注意:** データベースコンテナは、別のストレージサーバーに保存することもできます。

 **注意:** データベースコンテナには、別のマシンからアクセスできます。ただし、データベースの一貫性は、データベースのデータベースコンテナに同時にアクセスしているすべてのユーティリティおよびニュークリアスプロセスが、同じマシン上で実行されている場合にのみ保証されます。これは、ユーティリティプロセスとニュークリアスプロセスの同期に、同じマシン上でのみ表示できる IPC リソースが使用されていることが理由です。例外は、データベースコンテナにはアクセスせず、特別なリモート Adabas コールを実行するリモートユーティリティです。データベースのすべてのユーティリティおよびニュー

クリアスプロセスが終了したら、別のマシンでデータベースプロセスを再起動できます。ただし、データベースのデータベースプロセスが古いマシン上でアクティブであるときに、新しいマシンでデータベースプロセスを開始しないようにするのは、データベース管理者の責任です。これは、Adabas によってはチェックされません。

システム設計におけるパフォーマンス制御

システムのパフォーマンスの判断基準は、実行に必要な時間とコンピュータリソースです。これらは次のような理由から重要です。

- 指定時間内で完了しなければならないシステム機能がある場合
- 厳しい時間制約のある他のシステムと、コンピュータリソースの競合が起こる場合

ただし、パフォーマンスが最も重要な目標であるとは限りません。パフォーマンスと次の項目のどちらを優先するかについてよく考える必要があります。

- 柔軟性
- データ独立性
- 情報のアクセスし易さ
- セキュリティに関する考慮
- 情報が最新であること
- スケジュールのしやすさ、データベースの同時使用ユーザー数の影響
- ディスクスペース

パフォーマンスが、最適化すべき目標ではなく制約になる場合もあります。システムは時間やボリュームの問題が生じたとき、パフォーマンスよりは他の項目に注意を向けるべき場合もあり得ます。

システム設計におけるパフォーマンス制御方法

十分なパフォーマンスを実現できるかどうかは、次に掲げる事項の完成度にかかっていると考えられます。

- ハードウェアの設計
- データベースの設計
- データベースにデータをロードするときに使用するオプション
- アプリケーション機能のロジック（例えば、ダイレクトアクセスを使用するか、ソートと順次処理を組み合わせるか）
- オペレーション手順とスケジューリング

どの程度のパフォーマンスが必要かはシステム設計段階の早い時期に考慮しなければなりません。パフォーマンスを制御するうえでの基本要件として、次の事項を確認します。

1. 主なシステム機能について、ユーザーに時間の制約を聞きます。大半の場合、この要求事項は絶対的なものです。すなわち、この制約が満たされないと、システムは使用することができません。
2. 他のデータベースで試したり、予期される負荷をシミュレーションしたりして、利用可能なハードウェアリソースが十分であるかどうか評価します。新しいハードウェアが必要な場合、次の事項を考慮します。
 - 処理速度の速い高価な数台程度のハードウェアを組み合わせる使用するか、処理速度は比較的低速でも安価なハードウェアを何台も組み合わせる使用するかを決定する必要があります。通常、ハードウェアを何台も組み合わせると、低予算で大規模な計算能力を得ることができますが、これは、並列処理を使用する意味がある場合に限られます。同時使用ユーザー数がある程度以上であれば、このようなハードウェアを使って並列処理する意味がありますが、主にバッチプログラムを順次実行し、並列処理の必要性がない場合には、Adabas を並列処理する意味はあまりありません。
 - ハードウェアスレッドを複数のソケットに分散できます。さらに、ソケットを複数のボードに分散できます。このようにした場合、異なるソケットに接続されているハードウェアスレッド間で同期にかかる時間は、同じソケットに接続されている場合よりも大幅に長くなる可能性があります。データベースブロックがバッファプールから読み取られる場合は、各データベースコマンドが複数のデータベースブロック（インデックスとデータブロック、アドレスコンバータ、ファイルコントロールブロック、フィールド記述テーブル）を読み込む必要があるため、常に同期が必要です。スレッドを追加することによる処理能力向上の利点は、追加のスレッドが別のソケットまたはボードにある場合には同期時間が長くなり、打ち消される可能性があります。

もう1つの問題は、メモリが通常複数のソケットに分散されていることです。ソケット自体のメモリへのアクセスは比較的高速ですが、別のソケットに属するメモリへのアクセスは比較的低速です。特に、別のボードに配置されている場合はさらに低速になります。ハードウェアにもよりますが、リモートメモリへのアクセスには、ローカルメモリへのアクセスよりも、時間が約3倍かかります（NUMA アーキテクチャ：不均一メモリアクセス）。

したがって、ADANUC は、同じソケットまたはボード上のハードウェアスレッドに制限する必要があります。これは、特殊なオペレーティングシステムコマンドを使用するか、オペレーティングシステム仮想化のコンセプトを使用して実現できます。通常、この方法でパフォーマンスが向上します。同期コストの削減とローカルメモリへのメモリアクセスの高速化は、並列処理を増やすことによる利点よりも大きくなります。

このような最新のアーキテクチャでパフォーマンスを向上させるために、Adabas は Adabas 処理ユニット（ADANUC パラメータ APU）のコンセプトを使用しています。ここでは、1つの Adabas 処理ユニット（APU）は、コマンドキューとコマンドが処理される専用のスレッドで構成されます。オペレーティングシステムが、1つの APU に属するスレッドがグループになっていることを認識し、それらを同じソケット／ボード上にスケジュールすると、完全なニュークリアスプロセスを1つのソケット／ボードにバインドするよりも、パフォーマンスが向上する場合があります。1つのソケットで完全なニュークリアスプロセス

が実行されている場合は、APU コンセプトを使用すると、パフォーマンスが向上する場合があります。これは、複数のコマンドキューを使用すると、1つの大きなコマンドキューを使用した場合と比較して、ロックの競合数が減少することが理由です。

ただし、Software AG では、CPU バインディングと APU 使用率の最適な構成を推奨できません。これは、オペレーティングシステムによる差異や、ハードウェア実装による差異が大きすぎるからです。また、APU を使用してもまったくメリットがない場合もあります。お使いのハードウェア環境で異なる設定を試し、データベースのロードで最善の結果が得られる設定を選択することをお勧めします。

- 最新のストレージシステムの使用を考慮します。最新のストレージシステムを使用することにより、ディスクの故障によるダウンタイムを回避でき、良好なパフォーマンスを得ることができます。
- また、「[データベース設計](#)」、「[リカバリ／再スタート設計](#)」、「[データベースコンテナ、バックアップファイル、およびプロテクションログの場所](#)」の各セクションも検討してください。
- Adabas コンテナは、高可用性のクラスタ内に収容することができます。ただし、Adabas はこのクラスタを直接サポートしません。ユーザーが独自のスクリプトを記述する必要があります。クラスタ内のロードバランシングはサポートされません。

3. 論理設計モデルの観点から、次の事項を明確化します。

- 各レコードタイプの処理方法
 - レコードを検出する場合のシーケンスとアクセスパス
 - 実行頻度とボリューム
 - 稼動時間
4. 最も重要なパフォーマンスを持つプログラムを決定します。このとき、他のシステムのスケジューリングやパフォーマンスに関する影響、ボリューム、頻度、期限などについて考慮します。他のプログラムも最低限のパフォーマンス要求を持つことがあり、重要な機能の最適化の幅も狭まることがあります。
 5. 重要な機能それぞれについて、アクセスパスの短縮化、ロジックの最適化、オーバーヘッドを増加させるようなデータベース機能の削除などで、パフォーマンスを最適化します。第1段階では、柔軟性、情報へのアクセスのしやすさ、またはシステムの他の機能要件のレベルを下げずに、パフォーマンスがどうやって最適化するかを試行錯誤する必要があります。
 6. 重要な各機能のパフォーマンスを推定します。この推定値が満足のいくものでないときは、時間の制約か機能に対する要求を少しゆるめるよう調整するか、ハードウェアをアップグレードする必要があります。
 7. その他のシステム機能のパフォーマンスを推定します。総費用を計算し、金銭上の制約と、費用やリソース要求のピーク期間とを比較します。この推定値が制約を満たさないと、ユーザー、オペレーション部門またはシニアマネージメントと調整を行い、解決しなければなりません。
 8. 可能な場合、テストデータベースをロードし各種機能の時間を計測して、推定値をチェックします。テストデータベースのファイル内のレコード数やディスクリプタの値の数は、実際

に運用するときのデータベースと同程度にする必要があります。各レコード長は、順次処理のテスト以外ではあまり重要ではありません。これは、レコードを物理順に近い形で処理するときのみ重要になります。

Unicode のサポート

Adabas は International Components for Unicode (ICU) ライブラリ (V3.2) に基づいて Unicode をサポートします。ICU の詳細については、<https://www.ibm.com/software/globalization/icu> にアクセスして ICU ホームページを参照してください。ワイド文字フィールドフォーマット (W) は Unicode フィールドのために導入されています。Adabas ユーザーは、Adabas コールまたは圧縮と圧縮解除のユーティリティ ADACMP と ADADCU を使用する際にどの外部エンコードを使用するかを指定できますが、内部的には、すべてのデータは UTF-8 で格納されます。

外部エンコードは次で指定することができます。

- Adabas OP コマンド (Adabas セッションのデフォルトエンコードを指定することができます)
- フォーマットバッファとサーチバッファ (フィールドレベルでエンコードを指定することができます)
- ユーティリティ ADACMP および ADADCU (ユーティリティの実行中に使用されるデフォルトエンコードを指定することができます)

検索やソート処理の場合、Unicode バイトの順序は通常それほど重要というわけではありませんが、言語固有の照合は重要です。この理由のために、Adabas は照合ディスクリプタをサポートします。Unicode 照合アルゴリズムと言語固有の規則を適用することによって、照合ディスクリプタはオリジナルの文字列からバイナリの文字列を生成します。

メインフレームとの互換性に関する考慮事項

Unicode 文字セットを使ってアプリケーションを作成し、メインフレームと UNIX/Windows の両方のプラットフォームでアプリケーションを実行するつもりであれば、次の点を考慮する必要があります。

- テキストフィールドに W フォーマットを使うことを推奨します。A フォーマットを使用した方がよいケースは、値を構成している文字がどれも ASCII と EBCDIC に該当する場合、ASCII と EBCDIC の照合に関連性がない場合、照合ディスクリプタが必要ない場合に限定されます。
- メインフレームでは、ユーザーが内部エンコードを選択することができますが、UNIX/Windows プラットフォームでは、すべての W フォーマットフィールドは UTF-8 として内部的に格納されます。文字列の長さはそのデコードに依存するので、メインフレームの内部エンコードにも UTF-8 を使用するか、他のプラットフォームの内部エンコードでも、アプリケーションが格納する値がオーバーフローする可能性がないことを確認する必要があります。

- メインフレームでは、W フォーマットフィールドは ECS に基づきますが、UNIX/Windows プラットフォームでは、それらは ICU に基づきます。したがって、すべてのプラットフォームで利用可能なエンコードだけを使用する必要があります。
- UNIX/Windows プラットフォームでは、照合ディスクリプタは ICU に基づきますが、メインフレームでは、ユーザーは、照合ディスクリプタを生成するためのユーザー出口を用意しておく必要があります。これらのユーザー出口は、UNIX/Windows プラットフォームで使用される ICU 照合と互換性のある照合順序を生成しなければなりません。UNIX/Windows のプラットフォームでは、メインフレーム上と同じ動作を実現するために HE オプションを使用する必要があります。
- メインフレームと UNIX/Windows プラットフォームでの Adabas ダイレクトコマンドは、W フォーマットフィールドの取り扱いについて完全に互換性があるわけではありません。互換性のあるコマンドだけがクロスプラットフォームアプリケーションに使用されることを保証する必要があります。

ファイルおよびレコード設計

概念設計段階で分けたレコードタイプ1つずつに、ファイルを1つずつ対応付けながら、Adabas データベースを設計することも可能です。このような構成にしておくで、さまざまなアプリケーション機能要件に対応可能になり、予想外の要求がでてきても取り扱いが簡単になる反面、次のような理由から、パフォーマンスという観点からみると必ずしも最良とはいえません。

- Adabas の呼び出し回数が増えます。Adabas の呼び出しごとに、解釈、整合チェックおよびオーバーヘッドのキューイングが必要になります。
- それぞれのファイルから、最低、1回は、インデックス、アドレスコンバータおよびデータストレージブロックにアクセスします。この処理に必要な入出力に加えて、バッファプールのスペースが必要であり、後で発生する要求に必要なブロックが上書きされる可能性が高くなります。

したがって、重要なプログラムが使用する Adabas ファイルの数を減らした方がよい場合があります。それには、次のような手法を使用します。

- マルチプルバリューフィールドとピリオディックグループを使用します。
- 単一の Adabas ファイルで複数のレコードタイプを使用します。
- データの重複を制御します。

これらの各手法については、後述します。

マルチプルバリューフィールドとピリオディックグループ

次の例では、注文項目は注文ファイルのピリオディックグループとして定義されています。注文項目の繰り返し数は、各レコードによって異なります。

Order Number	Order Date	Date Required	Customer Number	Item Code	Quantity	
A1234E	29MAR	10JUN	UK432M	24801K	200	1ST OCCUR.
				30419T	100	2ND OCCUR.
				273952	300	3RD OCCUR.

マルチプルバリューフィールドまたはピリオディックグループは、メインレコードを呼び出しと入出力を行う際に、同時に検索／更新の対象にすることができます。その結果、CPU タイムが短縮され、入出力回数も少なくて済みます。

マルチプルバリューフィールドやピリオディックグループの使用法に関して、注意が必要な制約を次に示します。

- ピリオディックグループの中に別のピリオディックグループを含むことはできません。
- 1オカレンスの圧縮後のサイズによっては、レコードサイズが極端に大きくなることがあり、Adabas がサポートする最大レコード長より大きなレコード長になることもあります。

ピリオディックグループ内のディスクリプタや、ピリオディックグループ内のフィールドに基づいているディスクリプタは、FIND や SORT コマンドのソートキーとしては使用できません。さらに、マルチプルバリューフィールドとして定義されているディスクリプタや、ピリオディックグループ内に含まれているディスクリプタをいくつか対象とする検索要求には、決まった規則が適用されます。この規則については、『コマンドリファレンスマニュアル』に記述されています。

単一 Adabas ファイル内の複数レコードタイプ

ファイル数を減らすもう 1 つの方法に、同一 Adabas ファイルに 2 つの論理レコードタイプのデータを格納するという方法があります。例えば、下図の「複数のレコードタイプ (1)」に示す顧客ファイルと注文ファイルとを組み合わせる方法です。この方法は、Adabas の空値省略機能の利点を活用しています。

組み合わせたファイルについてのフィールド定義テーブル内のフィールドには、次のものがあります。

Key, Record Type, Order Data, Order Item Data

格納されるレコードは、次の形式です。

Key Type Order Data *

Key Type * Order Item Data

* は空値省略を示しています。

注文項目レコードのキーは、注文番号とこの注文内のシーケンス番号から構成されます。

この手法では顧客レコードと注文レコードタイプがコントロールブロックと上位レベルのインデックスブロックを共用するので、入出力回数を減らすことができます。したがって、ファイルの処理を開始するために読み込むブロック数も減り、空きスペースが増えるため、バッファプールには別タイプのブロックを読み込むことができます。

顧客レコードと注文レコードは、データストレージにいっしょにグルーピングされるので、ある顧客に対するすべての注文を検索するとき読むブロックを減らすことができます。顧客と同時に全注文を追加するときも、必要な入出力回数は減ります。

顧客データと注文データのどちらにも効率良くアクセスできるように、キーの設計は注意して行わなければなりません。顧客レコードのキーには、通常、空値の接尾辞を入れて、そのレコードに付加される同じ顧客の注文をそれぞれ区別できるようにします（下図「複数のレコードタイプ (2)」参照）。

A00231	000	Order header for order A00231
A00231	001	Order item 1
A00231	002	Order item 2
A00231	003	Order item 3
A00232	000	Order header for order A00232
A00232	001	Order item 1

キーの接尾辞の内容から、顧客レコードか注文レコードかプログラムが区別できる場合には、レコードタイプフィールドは不要です。プログラムは、追加フィールドを読み取るためにレコードを読み直すか、そうでなければ、Adabas にすべてのレコードタイプに関して全フィールドを返す必要があります。

データの重複

物理的重複

詳細レコードへのアクセス時に、ほぼ毎回、ヘッダーレコード内のいくつかのフィールドが必要になる場合があります。例えば、請求書を作成する場合、注文項目データと製品レコードの一部である製品の記述が必要です。この情報がプログラムで迅速に使用できるようにする最も簡単な方法は、注文項目データ内に製品記述情報のコピーを持つことです。すでに、他で物理的に存在するデータを重複して持つので、これを物理的重複といいます。

各詳細レコードのフィールドをいくつかヘッダーレコード内にピリオディックグループとして格納した場合にも、物理的重複として成立します。

論理的重複

クレジット制御ルーチンが、顧客に送付する請求書に合計が必要な場合を仮定します。この情報は関連する請求書を読み取り、それらを合計して求めることができますが、大量のレコードにランダムにアクセスする必要があります。この情報が顧客レコード内に常に格納され、正しく更新されていればもっと迅速に得ることができます。これを論理的重複といいます。重複した情報は他の場所に存在しているわけではなく、別のレコードの内容として存在しているからです。

物理的または論理的に重複している情報を更新するプログラムでは、重複しているデータも更新しなければならないので実行速度は遅くなります。物理的重複の場合は、通常、更新頻度の少ないフィールドによって大きな問題が起こることはほとんどありません。論理的重複では、たいていの場合1レコードの変更が他の何レコードかのデータに影響するので、更新も重複して行わねばなりません。

Adabas レコード設計

Adabas ファイル構造が決まったら、次はファイルのフィールド定義テーブルを定義します。フィールド定義の入力方法については、「FDT のレコード構造」を参照してください。このセクションでは、フィールドに使用できるオプションとパフォーマンスとの関係について説明します。

フィールドは、読み取りや更新の頻度の高い順に、レコードの先頭になるべく近い位置から並べます。これによって、レコードのフィールドを探すために必要な CPU 時間を短縮できます。読み取られることはほとんどありませんが、主に検索条件として使用されるフィールドは、最後の方に配置します。

グループを使用すると、読み込みや更新コマンドの内部処理の効率が良くなります。すなわち、フォーマットバッファが短くなるので処理時間の節約になり、Adabas が保持する内部フォーマットバッファプールのスペースが少なくて済みます。

数値フィールドは、最も頻繁に使用するフォーマットにロードします。こうすれば、必要になるフォーマット変換が最小限になります。ただし、節約される CPU 時間と必要なディスクスペースの関係を考慮しなければなりません。

固定ストレージ (FI) オプションを使用すると、通常、フィールドの処理時間は短縮されますが、必要なディスクスペースは増えます。特に、ピリオディックグループ内にフィールドが含まれている際は増えます。NUフィールドは、できるだけまとめる必要があります。FIフィールドは、できるだけまとめる必要があります。

各 Adabas 基本レコードを1つのデータブロックに納めることが重要です。データブロックの最大サイズは 32KB です（ブロックサイズの詳細については、「コンテナファイル」を参照）。Adabas フィールドに必要なスペースの詳細については、「ディスクスペースの使用法」を参照してください。Adabas フィールドに必要なスペースに加えて、各データブロックに 4 バイトおよび各 Adabas レコードに 6 バイトが必要です。

ラジオオブジェクト値

通常、各 Adabas レコードは1つのデータブロック内に収まらなければなりません。ただし、例外として、253 バイトを超える値（いわゆるラジオオブジェクト値、LOB 値）は、別の内部 Adabas ファイル（相互に関連付けられた LOB ファイル）に格納されます。元の Adabas ファイルは基本ファイルと呼ばれ、基本ファイル内のレコードは基本レコードと呼ばれます。LOB ファイル内のレコードは、LOB レコードと呼ばれます。LOB 値には、次の規則が適用されます。

- Adabas ファイルと関連付けされた LOB ファイルが存在しない場合、すべての値は基本レコードに格納されます。この場合、最大フィールド長は、16381 バイトに制限されます。
- LOB フィールドがディスクリプタまたは派生ディスクリプタの親フィールドである場合、フィールド値は常に基本レコードに格納され、最大フィールド長は 16381 バイトに制限されます。
- それ以外の場合、253 バイトまでの値は常に基本レコードに格納され、253 バイトを超える値は LOB ファイル内の 1 つ以上の LOB セグメントレコードに格納されます。この場合、基本レコードには 16 バイトの LOB 値参照が代わりに含まれます。

次の表は、Adabas のドキュメントで LOB 値と関連して使用される用語の概要を示しています。

用語	定義
基本ファイル	1 つ以上の LOB フィールドが含まれたユーザー定義の FDT を持つ Adabas ファイルです。
基本レコード	基本ファイル内のレコードです。このレコード内の LOB 値は、LOB ファイル内の LOB セグメントレコードを指し示す LOB 値参照により表現されます。実際の LOB 値はこれらのセグメントレコード内に含まれます。
LOB	ラジオオブジェクト。初期状態では、Adabas の LOB 値は最大約 2 GB（理論値）です。
LOB フィールド	LOB 値を格納する、Adabas ファイルの新しいタイプのフィールドです。
LOB ファイル	定義済みの FDT を持つ Adabas ファイルです。1 つ以上の LOB セグメントレコードに LOB 値が分散して格納されます。
LOB ファイルグループ	基本ファイルと LOB ファイルで構成される組み合わせです。単一のユニットとして表示されます。
LOB セグメント	パーティション分割された LOB 値の一部分です。LOB 値は、1 つ以上の LOB セグメントで構成されます。
LOB セグメントレコード	LOB ファイル内のレコードです。ペイロードデータとして LOB セグメントを、制御データとしてその他の情報を含みます。
LOB 値	LOB のインスタンスです。
LOB 値参照	基本レコードから、分割された LOB 値が含まれる LOB セグメントレコードへの参照またはポインタです。

次のタイプの LOB フィールドを定義できます。

- バイナリ型ラージオブジェクト (BLOB) : これらのLOBフィールドは、フォーマット A (英数字) と次のオプションで定義されます。LB (16381 バイトを超える値がサポートされています)、NB (値の最後の空白は切り捨てられません)、NV (オープンシステムプラットフォームとメインフレームプラットフォーム間で ASCII/EBCDIC 変換を行わないバイナリ値)。
- 文字型 (ASCII/EBCDIC) ラージオブジェクト (CLOB) : これらのLOBフィールドは、フォーマット A とオプション LB で定義されますが、オプション NB と NV は指定しません。さらに、フィールドオプション MU、NC、NN、NU を、これらのオプションの通常の規則を使用して定義できます。LOB フィールドはピリオディックグループのメンバとして定義することもできます。

**注意:**

1. Unicode LOB フィールド (フォーマット W (UNICODE) のフィールド) は現在サポートされていません。大きな Unicode 値を BLOB フィールドに保存することはできますが、フォーマット W で提供される他のエンコードへの変換はサポートされていません。
2. Adabas のフォーマットおよびフィールドオプションの詳細については、「*FDT のレコード構造*」セクションを参照してください。

Adabas システムファイル

Adabas ニュークリアスは、内部の Adabas ファイル (いわゆる Adabas システムファイル) を使用して内部データを保存します。これに対し、チェックポイントファイル、セキュリティファイル、およびユーザーデータファイルは、Adabas データベースで常に必要で、データベースの作成時に定義されます。レプリケーションシステムファイルは、Adabas-to-Adabas レプリケーションの初期化時にのみ作成されます。詳細については、「*Adabas-to-Adabas (A2A) レプリケーション*」セクションを参照してください。RBAC システムファイルは明示的に定義した場合にのみ作成されます。詳細については、「*Adabas ユーティリティの認可 (モード Adabas)*」セクションを参照してください。

次の Adabas システムファイルが存在します。

- チェックポイントファイル
- セキュリティファイル
- ユーザーデータファイル
- レプリケーションシステムファイル

■ RBAC システムファイル

チェックポイントファイル

Adabas チェックポイントファイルは、いくつかの重要なイベントである Adabas チェックポイントを記録するために使用されます。これらのチェックポイントは、次の場合に取り込まれます。

- Adabas ユーティリティの実行
- ニュークリアスの起動と停止
- Adabas ファイルへの排他的アクセス権を持つ Adabas ユーザーセッション
- ユーザー定義のチェックポイント

チェックポイントは、ユーティリティ ADAREP パラメータ CHECKPOINTS を使用して表示できます。このパラメータのドキュメントには、さまざまなチェックポイントタイプの説明が含まれています。

このチェックポイントは、データベースのバックアップ後に実行されたすべてのデータベース更新を再適用するユーティリティ ADAREC（データベースリカバリ）にとって特に重要です。ADAREC は、一部のユーティリティ操作を回復できません。そのため、ADAREC で回復できない、手動での再実行が必要なユーティリティの実行を示す SYNPN チェックポイントを検出すると、ADAREC は停止します。



注意: チェックポイントファイルに保存されている情報には、ユーティリティの再実行に必要なすべての情報が含まれているわけではありません。そのため、Software AG では、データベースを回復できるように、すべてのユーティリティの実行を記録することを強くお勧めしています。

セキュリティファイル

Adabas セキュリティファイルには、Adabas セキュリティ定義が含まれています。Adabas セキュリティの詳細については、Adabas セキュリティ定義の維持に使用する、ADASCR ユーティリティのドキュメントを参照してください。

ユーザーデータファイル

ユーザーデータファイルは、OP コマンドの Additions1 フィールドで指定されたすべてのユーザー ID (EIDS) の最後のトランザクションに関する情報の保存に使用されます。ETID を指定するのは、クラッシュ後に、Adabas を使用したプログラムの再起動処理の実装を可能にするためです。これを実装しないと、ETID を指定する意味がありません。



注意:

1. Software AG では、ユーザーデータファイルにデータを保存することを意図している場合にのみ、ETID を使用することを推奨しています。
2. Natural を使用して Adabas データベースにアクセスする場合は、Natural のドキュメントを参照して、Natural で ETID を使用する方法の詳細を確認してください。
3. Software AG では、ETID としてプロセス ID を使用しないこと（例えば、Natural で「\$\$」を ETID として指定しないこと）を推奨しています。ユーザーデータを生成したプロセスのプロセス ID がわかっている場合にのみ、ユーザーデータにアクセスできます。一部のオペレーティングシステムでは、プロセス ID が非常に長くなる可能性があり、その結果、ユーザーデータファイルに非常に大きなレコードが保存される可能性があります。

ADADBM REFRESH の機能を使用して、ユーザーデータファイルのデータを削除できます。当然ながら、ファイルに保存されているすべてのユーザーデータは失われます。

レプリケーションシステムファイル

レプリケーションシステムファイルの詳細については、「*Adabas-to-Adabas (A2A) レプリケーション*」セクションを参照してください。

RBAC システムファイル

RBAC システムファイルの詳細情報については、「*Adabas ユーティリティの認可 (モード Adabas)*」セクションを参照してください。

データアクセス手法

ディスクリプタの効率的な使用

ディスクリプタは、効率的な検索処理のために Adabas がインデックスを作成したフィールドであり、論理的順次読み取り処理を制御します。また、FIND や SORT ISN LIST のような一部の Adabas コマンドのソートキーとしても使用します。したがって、ディスクリプタの使用は、ファイルに対するアクセス方法と密接に関係しています。各ディスクリプタについて、ディスクスペースの追加とオーバーヘッドの処理が必要です。更新が頻繁なディスクリプタについては特にそうなります。ファイルに対して、どのタイプのディスクリプタをいくつ定義するかについては、次の点を考慮します。

- ディスクリプタの配分は、ディスクリプタを使用したときに、ファイルのレコードのうち数パーセント程度を選択できるほどにする必要があります。
- 既存のディスクリプタを使用して非常に少ない数のレコードを選択できる場合、検索条件をすっきりするためにディスクリプタを追加定義しないでください。
- 2つか3つのディスクリプタが頻繁に組み合わせて使用される（例えば地域、部、課）ときは、別々のディスクリプタで定義せず、スーパーディスクリプタとして定義することができます。

- あるディスクリプタについての選択条件がいつも値の範囲を含むときは、サブディスクリプタを使用することができます。
- あるディスクリプタについての選択条件に空値選択が含まれていないが、そのディスクリプタに多くの空値を指定できるときは、そのディスクリプタは空値省略オプションで定義します。
- 更新の頻繁なフィールドについては、インデックスも更新されるので、ディスクリプタのために作成されたインデックスを使った検索処理が速くなる分、データベース更新処理は遅くなることを認識する必要があります。
- フィールドをディスクリプタとして定義せずに、非ディスクリプタ検索を実行する場合、小規模のテストデータベースではパフォーマンスが良いかもしれませんが、大規模な本番データベースではパフォーマンスが低下する可能性があることを認識する必要があります。
- データベースに 253 バイトを超えるディスクリプタ値を格納する場合、それらは少なくとも 16KB のブロックサイズを持つインデックスブロックに格納されます。そのような値を格納したい場合、データベースは適切な ASSO コンテナファイルを持たなければなりません。または、データベースがそのような ASSO コンテナを作成できるように定義された位置が必要です。
- ディスクリプタ値の最大長は 1144 バイトです。通常、より大きな値を挿入しようとするデータベース処理は拒否されますが、ディスクリプタに対してインデックス切り捨てオプションを指定した場合、より大きなインデックス値は切り捨てられます。この結果、切り捨てられたインデックス値が関連する場合、検索処理の結果が正確ではなくなるかもしれません。そのような場合には警告が発行されます。

スーパーディスクリプタ

スーパーディスクリプタとは、最大 20 個までのフィールド（またはフィールドの一部）を組み合わせて作成するディスクリプタです。スーパーディスクリプタを構成するフィールドは、ディスクリプタであってもなくてもかまいません。検索条件に値の組み合わせが含まれるときは、通常のディスクリプタを組み合わせるよりもスーパーディスクリプタを使用した方が効果的です。これは、複数のインバーテッドリストの代わりに 1 つのインバーテッドリストにアクセスするだけで済むためです。また、該当レコードの最終的なリストを作成するために、ISN リストを AND 結合する必要がなくなります。スーパーディスクリプタは、ファイル読み取りの論理的順序の制御や ISN リストのソートに、通常のディスクリプタと同じように使用できます。

スーパーディスクリプタを使用する検索条件の値は、スーパーディスクリプタのフォーマットと同じでなければなりません（スーパーディスクリプタを構成するフィールドがすべて数値のときはバイナリフォーマット、それ以外のときは英数字フォーマット）。スーパーディスクリプタのフォーマットがバイナリの場合、問い合わせ機能やレポート機能を使用するときに、検索条件値を入力しにくくなることがあります。

サブディスクリプタ

サブディスクリプタとは、フィールドの一部から構成されるディスクリプタです。サブディスクリプタの元のフィールドはディスクリプタであってもなくてもかまいません。検索条件で英数字フィールドの先頭の *n* バイトや数値フィールドの最後の *n* バイトの値を範囲指定する場合、そのフィールドの必要なバイト数分をもとにして、サブディスクリプタとして定義することができます。これによって、検索条件では値の範囲ではなく 1 つの値だけを指定すればよいので、Adabas は中間の ISN リストを作成しマージする必要がなくなり、検索を効率よく行うことができます。例えば、AREA が 8 バイトの英数字フィールドであり、先頭の 3 バイトで REGION、後の 5 バイトで DEPARTMENT を表すとします。REGION が 111 のレコードだけが必要な場合、AREA = 11100000~11199999 という検索条件を指定しなければなりません。AREA の先頭の 3 バイトをサブディスクリプタと定義すれば、REGION = 111 という検索条件を指定することができます。

フォネティックディスクリプタ

フォネティックディスクリプタはフォネティック検索を行うために定義します。FIND コマンドでフォネティックディスクリプタを使用すると、同じようなフォネティック値を含むレコードがすべて返されます。ディスクリプタのフォネティック値はフィールド値の先頭 20 バイトに基づくもので、英字の値のみが考慮されます（数値、特殊文字および空白は無視されます）。

ハイパーディスクリプタ

ハイパーディスクリプタを使用すると、ハイパー出口に記述されているユーザー作成のアルゴリズムに基づいてディスクリプタ値を生成できます。単一の Adabas データベースに対して、最大 255 個までの異なるユーザー作成のハイパー出口を定義できます。各ハイパー出口は、複数のハイパーディスクリプタを処理することができます。

ハイパーディスクリプタは *n* 個から構成されるスーパーディスクリプタ、派生キー、またはその他のキー構造を実装するために使用できます。ハイパーディスクリプタの詳細については、「FDT のレコード構造」および「ユーザー出口とハイパー出口」を参照してください。

ソフトカップリング

検索条件の内部ファイルリンケージに使用するフィールドを指定することによってマルチファイルクエリを実行できます。この機能はソフトカップリングと呼ばれ、ファイルを物理的にカップリングする必要はありません。

ISN

Adabas ファイルの各レコードは、4 バイトの 0 より大きい符号なし整数である、内部シーケンス番号 (ISN) を持っています。問い合わせを効率的に実行するために、Adabas で内部的に ISN が使用され、Adabas FIND コマンドの結果は ISN リストとしての結果を表します。

レコードの ISN が知られている場合、その ISN 経由でレコードをアクセスすることは非常に効率的です (Adabas L1 コマンド)。

ユーザー割り当て ISN

ファイル内の各レコードの ISN は、通常、Adabas が自動的に割り当てますが、オプションとしてユーザーが割り当てることもできます。このことにより、インバーテッドリストを使用せずに、直接 ISN を使用して順次検索することが可能となります。ただし、ユーザー自身が、ISN 割り当てのアルゴリズムを開発しなければなりません。この ISN の値は、ファイルのアドレスコンバータに割り当てられたスペースに入る値 (詳細については、『Adabas ユーティリティ』の「ADAFDU」の MAXISN パラメータを参照) でなければなりません。また、ファイルにレコードを追加するアプリケーションはそれぞれ、必ずユーザーの ISN 割り当てのアルゴリズムを含んでいなければなりません。

ISN のディスクリプタとしての使用

インバーテッドリストを使用せずに、レコード内に関連レコードの ISN を格納して直接関連レコードを読み取ることもできます。

例えば、注文レコードを 1 つ読み取り、この注文に対する全顧客レコードを検索して読み取るアプリケーションを考えます。顧客レコードの ISN (1 つの注文レコードに対して複数の顧客が存在している場合、マルチプルバリューフィールドを使用できます) を注文レコードに格納する場合、注文レコードの中に ISN があるので、顧客レコードを直接読み取ることができます。これによって、その注文に対する顧客レコードを確認する際に、顧客ファイルに対して FIND コマンドを実行する必要がなくなります。この方法では、顧客レコードの ISN を含むフィールドを注文レコード内に設定し保持する必要があるため、ファイルをアンロードしたり、再ロードしても、同じ ISN が保持されていなければ、顧客ファイルの ISN の変更はないことを前提にしています。

ADAM の使用方法

Adabas ダイレクトアクセスメソッド (ADAM) を使用すると、インバーテッドリストをアクセスせずにデータストレージから直接レコードを検索できます。レコードが格納されているデータストレージのブロック番号は、レコードの ADAM キーに基づきランダムマイジングアルゴリズムを使用して計算されます。ADAM を使用することによって、アプリケーションプログラムに対して完全な透過性があります。

ADAM ディスクリプタを使用することによるパフォーマンス面での主な利点は、インバーテッドリストに対するアクセス数が減少することです。ADAM キーとして ISN を使用する主な利点は、アドレスコンバータに対するアクセス数が減ることです。

インバーテッドリストを使用した検索では3~4回のI/Oが必要ですが、ADAMキーを使用してレコードを検索すれば、ADAMでは一般的に平均1.2~1.5回（理論値）のI/Oで済みます。この平均には、アソシエータの制御下でファイルの他のブロックに格納されるオーバーフローレコードの平均も含まれます。オーバーフローレコードについては、通常のインバーテッドリスト参照で取得されます。

ADAMキーの値はユニークでなければなりません。レコードのISNをADAMキーとして使用してもかまいません。

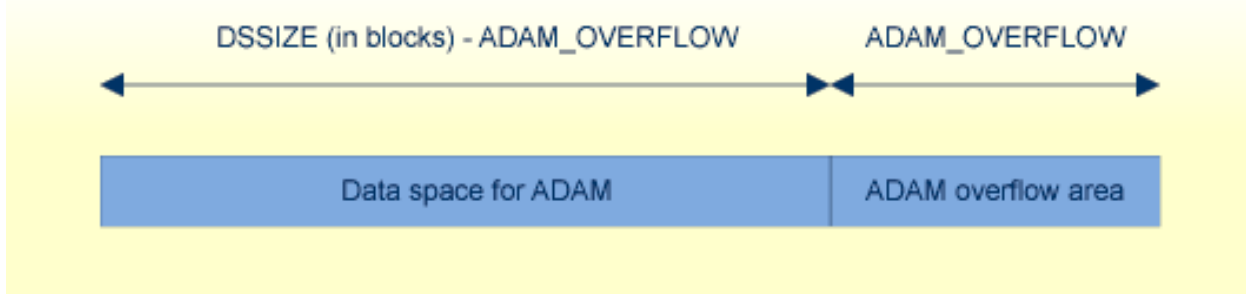
**注意:**

1. ADAMキーを介したADAMファイルへのアクセスは、オーバーフローエリアに保存されるレコードが多くなりすぎないようにファイルが十分なサイズで定義されている場合にのみ効率的です。ハッシュアルゴリズムによって決定されたブロックにレコードが収まらず、このブロック内のレコードの検索が成功しなかった場合は、続けて通常の検索アルゴリズムが実行されます。この場合、レコードへのアクセスには、非ADAMファイルに必要な時間よりも長い時間がかかります。
2. ブロックオーバーフローと、その結果生じるオーバーフローエリアのレコードを回避するには、通常、ブロックの平均フィルファクターが相対的に小さくなるように、十分なサイズでADAMファイルを定義する必要があります。つまり、ユーティリティは一般的に、より多くのI/Oを実行する必要があるので、同じ数のレコードを含む非ADAMファイルよりも時間がかかります。これは特に、ロードするレコードのソート順がハッシュキーの順序になっていない場合のADAMUPで顕著です。なぜなら、ロードする次のレコードは、ほとんどの場合、別のブロックに属しているからです。

ファイル定義ユーティリティ ADAFDUを使用して、ディスクリプタやISNをADAMキーとして定義します。パラメータには次の3つがあります。

パラメータ	説明
ADAM_KEY	ADAM キーの定義
ADAM_PARAMETER	データレコード分布アルゴリズムに関するパラメータ
ADAM_OVERFLOW	オーバーフローブロック数

ADAM のデータスペースは、（DSSIZE（ブロック単位） - ADAM_OVERFLOW）として計算します。



ADAMのデータスペースは後から拡張できません。ADAMオーバーフローエリアのみを拡大できます。しかし、ADAM領域は初期化時に複数のDSエクステンツにまたがることができます。ADAM領域はADAFDU実行中にフォーマットされて、使用中のマークが付けられます。

ADAM_OVERFLOWパラメータでオーバーフロー領域に使用するブロック数を定義します。少なくとも1ブロックは必要で、後でブロックを追加できます。オーバーフローブロックは、新規レコードのADAMに必要なブロックに対する空きがない場合に使用します。オーバーフロー領域に大量のレコードが格納されると、ADAMを使用して得られたパフォーマンス効果が減少します。ADAMファイル内のレコード分布はファイル情報ユーティリティADAFINを使用してチェックできます。

ファイルに領域の再利用オプションを指定している場合、それはオーバーフロー領域にだけ適用します。DATAパディングファクタは両方の領域（DATAとオーバーフロー）に適用します。

ADAM_PARAMETERパラメータを使用して、データレコードの分布を設定します。

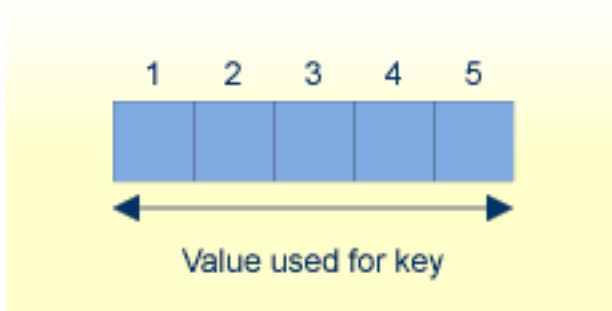
ADAMキーがISNまたは固定小数点形式のディスクリプタである場合、1ブロックに格納される連続する値の数を決定します。基本のアルゴリズムは次のとおりです。

```
DS number = (actual value / ADAM_PARAMETER) modulo number_adam_blocks
```

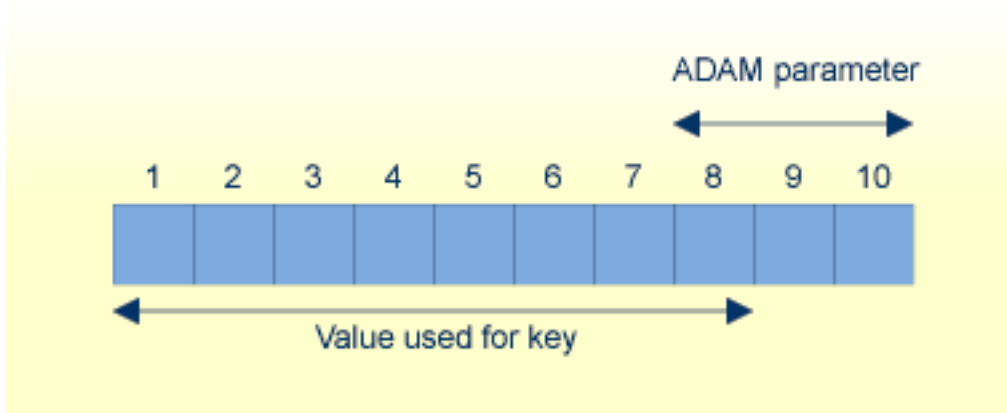
ADAMキーのフォーマットが英数字、バイナリまたは浮動小数点の場合、ADAMパラメータは、8バイト抽出の値の末尾からのオフセットを定義します。

例：フォーマットAのADAMキー：

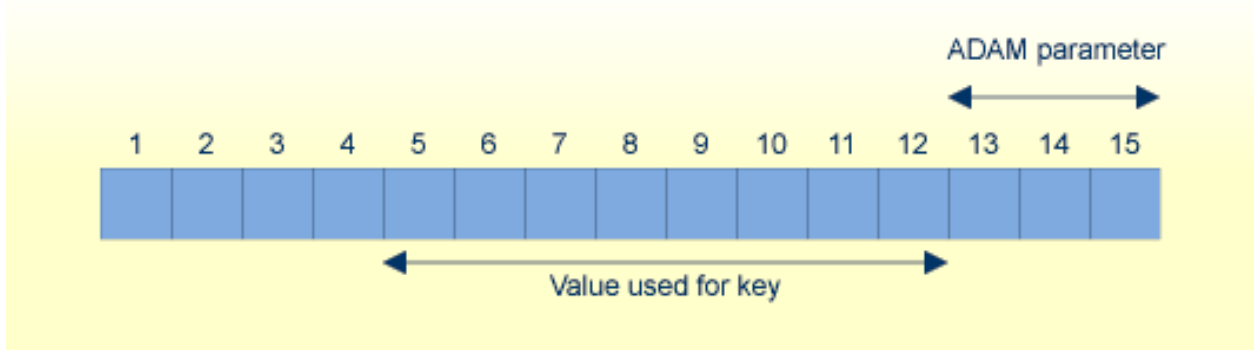
```
ADAM_PARAMETER = 3
Value's lengths = 5, 10, 15
```



値の長さが 8 バイト以下の場合、値全体が抽出値として使用されます。



(値の長さ - ADAM パラメータ) が 8 バイト以下の場合、最初の 8 バイトが抽出値として使用されます。



それ以外の場合は、ADAM パラメータのバイト数を削除した後の、最後の 8 バイトが抽出値として使用されます。

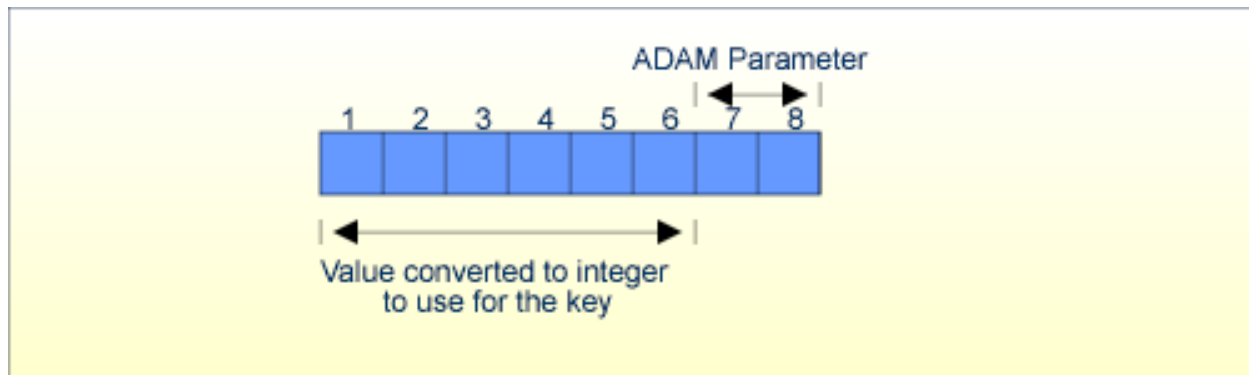
相対 DS 番号の計算アルゴリズムは次のとおりです。

DS number = (extraction value) modulo (number of ADAM blocks)

ADAM キーがパックまたはアンパック形式の場合、ADAM パラメータは値の末尾から ADAM 値に必要と思われる位置までを定義します。位置 1 から位置（値の長さ - ADAM パラメータ）までの ADAM 値は 8 バイトの整数値に変換されます。

例：フォーマット **P** または **U** の **ADAM** キー

```
ADAM_PARAMETER = 2  
Value's length = 8
```



相対 DS 番号の計算アルゴリズムは次のとおりです。

```
DS number = (integer value) modulo (number of ADAM blocks)
```

ADAM キーのフォーマットが固定小数点の場合、または ADAM キーが ISN の場合、抽出値は（ADAM キー値） / （ADAM パラメータ）です。

例：フォーマット **F** の **ADAM** キー

ADAFDU に次の値を入力します。

```
DSSIZE = 40 B  
ADAM_KEY = FF  
ADAM_OVERFLOW = 10  
ADAM_PARAMETER = 12
```

ファイルは ADAM ファイルで、ADAM キーにユニークディスクリプタ FF を使用します。ADAM DS 領域は 30 ブロック使用し、オーバーフロー領域に 10 ブロック確保します。各ブロックには 12 個の連続する値が格納されます。

値は次のように DS ブロックに格納されます。

FF 値	DS ブロック
0～11	1
12～23	2
24～35	3
...	...
348～359	30
360～371	1
372～383	2
...	...

ディスクスペースの使用法

データベース環境においては、次の理由から、ディスクスペースの効果的な使用が重要になります。

- 複数のユーザー間でデータを共有（同時に、異なる組み合わせで）するときは、通常、組織のデータのかなりの部分をオンライン（ディスクを使用する）で格納する必要があります。
- アプリケーションによっては、非常に大容量のデータを保持しています。

ディスクスペースの効率的な使用法については、システムの他の目標（パフォーマンス、柔軟性、使い易さ）を考慮しながら決定を下さなければなりません。このセクションでは、これらの要素とディスクスペースの効果的使用法との兼ね合いに関する考察と手法について説明します。

圧縮

各フィールドは次の3つの圧縮に関する指定のうち1つを定義できます。

1. 通常の圧縮（デフォルト）。Adabasは、英数字フィールドの末尾の空白と数値フィールドの先端のゼロを除去します。しかし、圧縮された値の長さが126以下の場合は追加の長さバイトを1バイト、126より大きい場合は2バイトを必要とします。空値は長さバイト=1に圧縮されます。
2. 空値省略。通常の圧縮に加えて、フィールドが空値のとき省略します。
3. 固定ストレージ（FI）。フィールドの圧縮は行いません。ただし、データストレージ内の追加の長さバイトは省略します。

次の図に Adabas の圧縮の例を示します。この例では、5 バイトの英数字フィールドに各圧縮オプションを使用して値を格納しています。各格納値の前の数字は、長さバイトを含むフィールドのバイト長を示します（FI フィールドにはありません）。空値省略のアスタリスク（*）は、省略されたフィールド数を示しており、レコード内のこの位置で存在する空値の（省略された）フィールド数を示す1バイトのフィールドです。bは空白を示します。

Field Value	Ordinary Compression	Fixed Storage	Null Value Suppression
ABCbb	04414243 (4 bytes)	4142432020 (5 bytes)	04414243 (4 bytes)
ABCDb	0541424344 (5 bytes)	4142434420 (5 bytes)	0541424344 (5 bytes)
ABCDE	064142434445 (6 bytes)	4142434445 (5 bytes)	064142434445 (6 bytes)
bbbbbb	01 (2 bytes)	2020202020 (5 bytes)	* (1 byte)

Field Value	Ordinary Compression	Fixed Storage	Null Value Suppression
ABCbb	4ABC (4 bytes)	ABCbb (5 bytes)	4ABC (4 bytes)
ABCDb	5ABCD (5 bytes)	ABCDb (5 bytes)	5ABCD (5 bytes)
ABCDE	6ABCDE (6 bytes)	ABCDE (5 bytes)	6ABCDE (6 bytes)
bbbbbb	1b (1 byte)	bbbbbb (5 bytes)	* (1 byte)

選択する圧縮オプションは、フィールドのインバーテッドリストの作成（ディスクリプタの場合）、またフィールドの圧縮と圧縮解除に必要な時間にも影響します。

固定ストレージ

固定ストレージとして定義したフィールドは、圧縮の対象になりません。長さバイトも付加されず、フィールドの標準長で格納されます。固定ストレージは、長さの短いフィールドや圧縮できないフィールドに対して有効なオプションです。FIフィールドを使用する際の制約については、「FDT のレコード構造」を参照してください。

通常の圧縮

通常の圧縮では英数字フィールドの末尾の空白や数値フィールドの先頭のゼロを除きます。上記の Adabas の圧縮に示したように、少なくとも 2 バイトの末尾の空白または先頭のゼロが除去されれば、ディスクスペースを節約することができます。

空値省略

フィールドに対して空値省略を指定した場合にフィールド値が空値であると、長さバイトと圧縮済みの空値の代わりに、1 バイトの空値フィールドインジケータが格納されます（前記の図参照）。この空値インジケータは、レコード内のその位置で空値省略されたフィールド（連続フィールド）の数を示します。最高 63 個までの空のフィールドを 1 バイトで表すことができます。したがって、空値になる頻度の高いフィールドはレコード内での物理的な位置を隣接させ、空値省略と定義します。

ディスクリプタフィールドの場合、空値省略を指定すると、インバーテッドリストに空値は作られません。したがって、FIND コマンドでディスクリプタに空値を指定すると、データストレージ内にディスクリプタが空値のレコードがあっても、該当レコードなしという結果になります。空値省略と定義したフィールドから作られるサブディスクリプタやスーパーディスクリプタについても同様です。つまり、元のフィールドの該当バイトが空値で、フィールドが空値省略オプションで定義されていると、サブディスクリプタに対するエントリは作られません。また、元のフィールドの該当バイトが空値で、フィールドが空値省略オプションで定義されていると、スーパーディスクリプタに対してもエントリは作られません。

したがって、ディスクリプタフィールドに対して空値省略を使用するかどうかは、空値による検索が必要かどうか依存し、またディスクリプタを論理的順次読み取りやソートの制御に使用する場合は、ディスクリプタが空値のレコードを読み取る必要があるかどうか依存します。このような必要がなければ、空値省略を通常どおりに使用します（FI オプションを使用しないとき）。

空値省略をマルチプルバリューフィールドやピリオディックグループ内のフィールドに指定すると有効であり、必要なディスクスペースや内部処理を減らすことができます。これらのフィールドの更新は、使用する圧縮オプションによって変わります。空値省略オプションで定義したマルチプルバリューフィールドが空値に更新されると、右側の値はすべて左側にシフトされ、それに対応して値の数も減らされます。ピリオディックグループ内のフィールドがすべて空値省略で定義され、グループ全体が空値に更新されると、更新されたオカレンスが最終のオカレンスであれば、オカレンスカウントが減らされます。マルチプルバリューフィールドやピリオディックグループの更新に関する詳細については、「FDT のレコード構造」、および『コマンドリファレンス』の「A1 コマンド」、および『コマンドリファレンス』の「N1/N2 コマンド」を参照してください。

マルチプルバリューフィールドとピリオディックグループ

マルチプルバリューフィールドとピリオディックグループの値には、通常 8 バイトのヘッダーが先行します（または、1 バイトの MU または PE カウントが先行することもあります）。ピリオディックグループの各オカレンスには 2 バイトの長さインジケータが先行します。ピリオディックグループに空のオカレンスが含まれる場合、最高 32767 までの空のオカレンスが 2 バイトの空のピリオディックグループオカレンスカウントに圧縮されます。

パディングファクタ

大量のレコード更新（A1 コマンド）が発生する場合、かなりの量のレコードが移行、すなわち、現在のブロックから拡張レコード用のスペースがある別のブロックへ移動することになります。ファイルのロード時、このファイルに対するデータストレージのパディングファクタを大きくとれば、移動するレコードの量を大幅に減らすことができます。パディングファクタとは、レコード拡張用に確保しておく各物理ブロック内の比率を表すものです。ファイルをロードしているときや、ファイルに新規レコードを追加するときは、パディングエリアは使われません。レコードの拡張される比率が小さい場合、レコードの拡張が発生しないブロックのパディングエリアは無駄になってしまうので、パディングファクタを大きくとるべきではありません。

大量のレコード更新／追加によって、1つまたは複数のディスクリプタの現在の値の範囲内に多数の新しい値が挿入される場合、アソシエータ内でもかなりの量の移行が発生します。これはアソシエータのパディングファクタを大きくとれば減らすことができます。

パディングファクタを大きくとると、必要なディスクスペースが大きくなり（ブロック当りのレコードやエントリが小さくなる）、読み取られる物理ブロックが多くなるので順次処理の効率が悪くなります。

パディングファクタは、ファイルの定義（ユーティリティ ADAFDU を使用）時に定義し、リオーダ（ユーティリティ ADAORD を使用）のときに変更します。

セキュリティ計画

このセクションでは、データベースのセキュリティに関する一般的な考慮点と、データベース内のデータの安全性を確保する Adabas のセキュリティ機能について説明します。

セキュリティを効果的に実行するには、次の点を考慮しなければなりません。

- システムの安全性はシステムの最も弱い要素で決まります。この要素はシステムのデータ処理部門以外の場合も考えられます。例えば、印刷されたリストを正しく保護しなかったなどです。
- 「絶対に安全な」システムを設計するのはほとんど不可能です。それにかかる費用よりも利益の方が高いと思われる場合に、セキュリティ違反が行われる可能性が高いからです。
- セキュリティにかかるコストが高くなることがあります。コストには、セキュリティを実施する方法の計画の調整とその効果の監視に必要なオーバーヘッド、マシンリソースおよび時間が含まれます。

セキュリティのためのコストは、セキュリティ違反の危険やコストを計るよりも容易ですが、しかし、セキュリティの実施方法によっては、セキュリティの分野以外にも利益をもたらす場合があるため、損得勘定が難しい場合もあります。セキュリティ違反のコストはその違反の種類によります。コストとしては、次のものが含まれます。

- 違反をリカバリするための時間

- 個人的な取り決め、契約などによるペナルティ（罰金）
- 顧客、供給者などとの関係に与える悪影響

Adabas のセキュリティ機能の概要

このセクションでは、Adabas と Adabas サブシステムが提供するセキュリティ機能の概要について説明します。このセクションで扱う機能の詳細については、「*Adabas のセキュリティ機能*」セクションを参照してください。

Adabas 認証

Adabas 認証は、ユーザーに有効な資格情報を提供させることで、アプリケーションがユーザーのコンテキストでデータベースにアクセスする手段を提供します。

Adabas 認証の詳細については、「*Adabas のセキュリティ機能*」セクションの「*Adabas 認証*」を参照してください。

Adabas ユーティリティの認可

ユーティリティの認可は、選択的なアクセス権限を持つ役割をユーザーに割り当てることによって、データベースでの Adabas ユーティリティの使用を制限する手段を提供します。

Adabas ユーティリティの認可の詳細については、「*Adabas のセキュリティ機能*」セクションの「*Adabas ユーティリティの認可*」を参照してください。

Adabas パスワードのセキュリティ (ADASCR)

Adabas パスワードのセキュリティ (ADASCR) の詳細については、「*Adabas のセキュリティ機能*」セクションの「*Adabas パスワードのセキュリティ*」を参照してください。

暗号化

暗号化しておけば、Adabas コンテナファイルの内容を権限のないユーザーに見られる心配がなくなります。Adabas は、コンテナファイルに格納するデータを暗号化できます。しかし、これはデータストレージに格納されるデータレコードにだけ当てはまり、アソシエータ上のインバートドリストには当てはまりません。

暗号化の詳細については、「*Adabas のセキュリティ機能*」セクションの「*暗号化*」を参照してください。

トランザクションのコンセプト

すべてのデータベースにおける重要なコンセプトは、データベースの整合性を保証するためのトランザクションの可用性のコンセプトです。トランザクションによって、一連のデータベース更新処理がコミットされることが保証されます。つまり、すべての更新がデータベース内で永続的になります。エラーが発生した場合は、すでに実行された更新処理が完全にロールバックされます。

このセクションでは、トランザクションの概念について簡単に説明します。詳細については、『コマンドリファレンスセクション』を参照してください。

Adabas には、トランザクションのコンセプトをサポートする、次のデータベースコマンドがあります。

- ET - End of Transaction（トランザクションの終了）、トランザクションをコミットします
- BT - Backout Transaction（バックアウトトランザクション）、トランザクションをロールバックします。

ファイルによっては、通常のトランザクションロジックに組み込まない方が望ましいものがあります。そのファイルについては、トランザクションがロールバックされたとしても、すべてのデータベース更新がデータベース内に保持する必要があります。このようなファイルの例としてログファイルがあります。ログファイルには、すべてのユーザーのアクティビティが記録される必要があり、後でバックアウトされるトランザクション内のアクティビティも記録される必要があります。

ロックのコンセプト

データベースの整合性を保証するために、トランザクションが必要なレコードを、他のユーザーが更新できないようにする必要があります。Adabas では、この目的で、トランザクションの間レコードをロックできます。

Adabas は次のタイプのロックをサポートしています。

ロックタイプ	説明
共有または読み取りロック (S)	レコードの排他的ロックを取得しているユーザーがいない場合は、共有ロックまたは読み取りロックを取得できます。S ロックを使用すると、他のユーザーはロックされているレコードを更新できません。ただし、他のユーザーは依然として共有ロックを取得できます。
排他的または書き込みロック (X)	ある Adabas レコードに対して S または X ロックを取得しているユーザーがいない場合にのみ、その Adabas レコードの排他的ロックまたは書き込みロックを取得できます。レコードの変更や削除は、レコードの X ロックがある場合にのみ可能です。新しいレコードを作成すると、そのレコードは自動的に排他的にロックされます。

サブトランザクション

場合によっては、トランザクション全体ではなく、トランザクションのサブセットのみをロールバックする必要があります。Adabas には、この目的でサブトランザクションの概念があります。これは、ET および BT コマンドの特別なオプションを介して実装されます。

ET 同期

次のような場合に、データベースは整合している状態になっている必要があります。

- ADABCK を使用して、データベースのバックアップを作成する場合
- ADAOPR EXT_BACKUP を使用して、外部バックアップを実行する場合
- 新しいプロテクションログ (PLOG) に切り替えた場合



注意: 新規の PLOG エクステンツに切り替える場合は、PLOG のすべてのエクステンツが 1 つの PLOG と見なされるので、ET 同期は必要ありません。

上記すべての場合で、データベースの ET 同期を実行する必要があります。つまり、以下のことを意味します。

- 現在アクティブなすべてのトランザクションは、トランザクションが終了するか、タイムアウトが発生するまで動作を続けます。タイムアウト時間は、Adabas ニュークリアスパラメータ TT、または ADABCK の場合はパラメータ ET_SYNC_WAIT によって定義されます。
- Adabas コマンドが新しいトランザクションを開始しようとした場合、コマンドは ET 同期が完了するまで待機する必要があります。
- すべてのアクティブなトランザクションが終了すると、すぐに ET 同期のフェーズ 2 が開始されます。これで、整合性のある状態で実行される必要があるすべてのアクティビティを実行できます。例えば、整合の取れた状態でデータベースをディスクに保存するためのバッファフラッシュなどです。
- ET 同期のフェーズ 2 が終了すると、ET 同期の終了を待機していたすべてのコマンドを続行できます。

オプション NEW_PLOG なしで ADABCK DUMP ダンプを使用して、ファイルレベルでバックアップを作成した場合、ET 同期はファイルレベルでのみ実行されます。

- ET 同期が待機するのは、ダンプされたファイルにアクセスするトランザクションの終了のみです。
- 新しいトランザクションは、それが他のファイルにのみアクセスしているのであれば、トランザクションを開始できます。ただし、このようなトランザクションがダンプされたファイルの 1 つに拡張された場合は、コマンドは ET 同期の終了を待機する必要があります。

リカバリ／再スタート設計

このセクションでは、データベースのリカバリ／再スタートの設計について説明します。

システムを設計するうえで、リカバリ／再スタートの計画を十分に練っておくことが重要になりますが、データベースを使用するシステムでは、このことが特に重要になります。障害の原因の大部分は、基本的なシステム設計プロセスの段階で、予測、評価し、解決することができます。

リカバリ機能は、当然の機能として特に明示されないことがよくあります。すなわち、どのような事態になっても、システムがリカバリを行い再スタートできることが前提とされています。しかし、システムのいろいろなユーザーが必要とするリカバリレベルについて決定すべき事柄が存在します。リカバリ機能は、DBAがイニシアティブをとり、必要な事柄を設定すべき分野です。まず、システムを利用する予定のユーザーに対して、リカバリ／再スタートの要件について質問しなければなりません。考慮の必要な最も重要な点として次のものがあります。

- システムが使用できなくなっても、どの位の期間ユーザーは業務を継続できるか。
- どの位の期間、各フェーズを遅らすことが可能か。
- 手作業でユーザーが入出力チェックを行う場合、どのような方法があるか。また、どのくらいの時間が必要か。
- リカバリ／再スタートが発生した場合、データ保全性が維持されていることを確認するためにどのような手順を行う必要があるか。

リカバリの計画とインストール

一度、リカバリ／再スタートが必要だと認められたら、DBA はこれらの要求を満たすのに必要な方法の計画に進むことができます。ここで述べる手法は上記を行うための基本的な手引として使えるはずです。

1. システム内のさまざまなユーザーがデータを共有するレベルや程度について決定します。
2. システムに対するリカバリパラメータを設定します。このパラメータとしては、予測／実際のブレークダウン率、平均の遅れと影響を受ける項目、およびセキュリティに使用する項目を含みます。
3. リカバリ設計の要点をまとめたアウトラインを準備します。このアウトライン内には次の情報を含むようにします。
 - 整合性チェックに関する計画。データの整合性チェックは、できるだけシステムへのデータ入力時に近い時点で行う必要があります。レコード入力時と別にデータを途中で更新すると、リカバリはより難しくコスト高になります。
 - データベースや選択ファイルのダンプ（バックアップコピー）。
 - ユーザーチェックポイントと Adabas チェックポイント。
 - ET ロジック、排他ファイル制御、ET データの使用。

リカバリ方法の適合性、有効性およびコストを決定するために、リカバリ方法はすべての起こり得る故障の状況に対応できるものでなければなりません。

4. リカバリ／再スタートに必要なすべてのリソースが必要なときに使用できるかどうか判断できるようオペレーション担当者とも相談する必要があります。
5. 最終的なリカバリ設計はドキュメント化し、ユーザー、オペレーション担当者、その他システムに関連する人々に知らせ、再確認しなければなりません。

データベースコンテナ、バックアップファイル、およびプロテクションログの場所

データベースのクラッシュ後にデータベースを再起動すると、自動再スタートが実行されます。ニュークリアスがクラッシュしたときにアクティブだったすべてのトランザクションがロールバックされ、不足しているすべてのデータベース更新が ASSO および DATA コンテナに書き込まれます。この目的のために、更新処理は WORK コンテナに記録されています。ただし、ディスクが破損している場合は、自動再スタートが失敗する可能性があります。このような場合に、バックアップおよびプロテクションログからデータベースの状態を復元できることが重要です。これは、バックアップファイルとプロテクションログ (PLOG) が別の独立したディスクに保存されている場合にのみ保証されます。Adabas は、更新処理を 2 回ログに記録します。つまり、自動再スタートの場合に WORK に 1 回、リストア／リカバリの場合に PLOG に 1 回です。これは、ログが保存されているディスクが破損した場合に現在のデータベース状態のリカバリを可能にするためです。



注意:

1. ディスクの問題を回避するために、RAID システムなどのハードウェアベースのソリューションも検討できます。ただし、RAID システムではソフトウェアや処理のエラーから保護されないため、バックアップと PLOG を作成することをお勧めします。
2. また、コンピュータシステム全体が壊れた場合のディザスタリカバリも検討する必要があります。
3. 現在のコンピュータには通常、十分なバッファプールを使用できる十分なメモリが搭載されています。そのため、ほとんどの場合、ASSO や DATA コンテナのアクセス特性はパフォーマンスに関連しなくなります。ただし、大規模なデータベースは例外で、物理 I/O の数を減らすための十分な容量のバッファプールを定義できません。
4. 特に、更新処理が 1 回または数回しかない短いトランザクションが多い場合は、WORK コンテナと PLOG にレイテンシ時間が短いディスクを使用する必要があります。たとえば、SSD ディスクやキャッシュが搭載されたストレージデバイスを使用します。これは、ログ情報が WORK および PLOG に書き込まれた後でのみトランザクションをコミットできるためです。PLOG および WORK に使用されるデバイスによっては、多くの短い更新トランザクションを実行するアプリケーションのスループットが最大 10 倍以上になることが確認されています。

要求と機能のマッチング

大体のリカバリ要件が決まったら、次にリカバリ／再スタートを行うための関連機能（Adabas 機能も Adabas 以外の機能も）を選択します。以下に、リカバリ／再スタートに関連する Adabas 機能について説明します。

トランザクションリカバリ

多くのバッチ更新プログラムは、以下のような特性を持つ入力トランザクションストリームを処理します。

- トランザクションにより、プログラムが検索、追加、更新、削除を行うレコードの数は少数です。例えば、注文入力プログラムでは、各注文に対して顧客レコードと製品レコードを検索し、データベースにその注文と注文項目データを追加し、製品レコードの注文フィールドの量を更新します。
- プログラムは、トランザクション開始から終了までに使用するレコードの排他制御が必要ですが、トランザクションが完了したら、レコードを解放し、他ユーザーが更新したり削除したりできるようにします。
- トランザクションが未完了の状態になってはなりません。すなわち、2レコードを更新する場合は、2レコードを両方とも更新するか、いずれも更新されないかのどちらかでなければなりません。

Adabas エンドトランザクション (ET) コマンド

Adabas ET コマンドを使用すると、次のことが可能です。

- 完了したトランザクションによる追加、更新、削除処理がすべてデータベースに適用されるようになります。
- 全体的または部分的なシステム障害により中断されたトランザクションについては、すべての更新処理をデータベースから取り除かれます。
- プログラムで、Adabas システムファイルにユーザー再スタートデータ (ET データ) を格納することができます。この ET データは、Adabas の OP または RE コマンドで再スタート時に取り出すことができます。
- トランザクション処理中にホールド状態にあった全レコードを解放します。

Adabas クローズ (CL) コマンド

Adabas CL コマンドでユーザーの最新 ET データを更新できます（例えばジョブ完了フラグの設定）。

ET データの読み込み

Adabas の OP コマンドを使用して、ユーザー再スタート後または新規ユーザー／Adabas セッションの開始時に、ET データを読むことができます。このとき、ユーザー識別子（ユーザー ID）も指定しなければなりません。このユーザー ID を使用して、Adabas はユーザーの ET データを検索します。

他のユーザーのユーザー ID がわかっている場合は、RE コマンドを使用して他のユーザーの ET データを読み取ることができます。これは、例えばオンライン更新処理のスタッフ管理を行う場合に有効です。

システムおよびトランザクション障害

Adabas セッションが異常終了すると、各 Adabas セッションの始めに自動的に呼び出された Adabas AUTOBACKOUT ルーチンが、中断されたすべてのトランザクションが行った更新処理をデータベースから除去します。

個々のトランザクションが中断された場合は、トランザクションが行った更新をすべてデータベースから自動的に除去します。また、アプリケーションプログラムでも、Adabas の BT コマンドを使用して、現在のトランザクションの処理をバックアウトするように明示的に指定できます。

No-BT ファイル

ニュークリアスがクラッシュした場合、次の事項を考慮に入れる必要があります。

- 最後の ET がデータベースに適用される前に発行された No-BT ファイルへのすべてのデータベース更新。
- 最後の ET の適用後に発行された No-BT ファイルのデータベース更新が、データベースに適用されるかどうかは定義されません。

Adabas トランザクションリカバリの制限

Adabas トランザクションリカバリについては、次のような制約を考慮しなければなりません。

- アソシエータの重要な部分と Adabas WORK のデータプロテクションエリアのどちらか一方でも物理的に読み取りができなかったり、上書きされたりした場合には、トランザクションリカバリ機能は動作しません。このような状況はほとんど発生しませんが、発生した場合の対処手順を作成しておく必要はあります。物理的に破壊されたデータベースのリカバリ方法に関する詳細な手順については、リカバリ／再スタート手順の節を参照してください。
- トランザクションリカバリ機能はデータベースの内容を回復するだけです。非Adabas ファイルを再配置したり、ユーザープログラムのステータスを元どおりに戻すことはしません。
- 特定ユーザーのトランザクションによる処理だけをバックアウトすることはできません。このユーザーが追加または更新したレコードに対して別ユーザーがトランザクションを実行した可能性があるからです。

Adabas チェックポイントコマンド

次の理由から、ある種のプログラムでは ET コマンドの使用は効果的ではありません。

- 各トランザクションの実行中、プログラムが多数のレコードをホールドしなければならない場合。このことは、デッドロックの可能性を増加させ（Adabas では 2 ユーザーの一方のトランザクションをバックアウトすることでデッドロックを自動的に解消しますが、大量のトランザクションを再処理しなければならなくなります）、Adabas ホールドキューをかなり大きくとる必要があります。
- 複合検索により見つかるレコードが多く長いリストをプログラムで処理する場合、このリストの途中から再スタートするのは困難です。

上記のようなプログラムでは、必要に応じて Adabas チェックポイントコマンド (C1) を使用して、プログラムが更新中である 1 つ以上のファイルのリストア可能地点を設定できます。実行中の更新のタイプ（排他制御）に応じて、特定のコマンドが使用されます。

排他ファイル制御

ユーザーは 1 つ以上の Adabas ファイルの排他更新制御を要求できます。OP コマンドで排他制御を要求し、他ユーザーがファイルを現在更新中でなければ、排他制御を行えます。あるユーザーに排他制御が与えられると、他ユーザーはそのファイルを読み込むことはできますが、更新することはできません。

論理順にレコードを処理するときも、検索を実行するときも、プログラムが読み取ったり更新したりするレコードの件数が増えて処理時間が長くなりますが、排他制御を使用すれば、プログラムが使用するレコードをユーザーが更新する懸念はなくなります。これによって、各レコードに対してレコードホールドコマンドを実行する必要がなくなります。

排他制御ファイルのチェックポイント

排他制御ユーザーは ET コマンドを使用しても使用しなくてもかまいません。ET コマンドを使用しないときは、C1 コマンドを発行して（ユーザーデータを格納したい場合）、チェックポイントを設定することができます。

システムおよびプログラム障害

システムまたはプログラム障害が発生した場合、排他制御下で更新中のファイル（群）は、ADABCK または ADAMUP を使用して、障害が発生したプログラムの実行開始以前の状態にリストアすることができます。

排他ファイル制御の制限

排他ファイル制御には、次のような制限があります。

- 最終チェックポイントへのリカバリは自動的には行われず、障害が起きたときのデータプロテクションログがリカバリ処理に必要です。ただし、ユーザーが ET コマンドを発行したときは、これは適用されません。
- システム障害後の再スタート状態で、Adabas は、システム中断時排他制御の下で更新されていたファイルを他ユーザーが更新するのをチェックせず、また回避もしません。

ユーザー再スタートデータ

Adabas の ET および CL コマンドでは、オプションとして Adabas システムファイルに 2000 バイトまでのユーザーデータを格納できます。

Adabas の ET、C3 および CL コマンドでは、オプションとして Adabas システムファイルに 2000 バイトまでのユーザーデータを格納できます。

ユーザーごとにユーザーデータを 1 レコード保持します。ユーザーが新規にユーザーデータを書き読むたびに、このレコードは書き換えられます。OP コマンドでユーザー ID を指定したときだけ、Adabas セッションが終了してもデータが保持されます。ユーザーデータは OP または RE コマンドで読み取ることができます。また、他のユーザーのユーザー ID がわかっているならば、RE コマンドでそのユーザーデータを読み取ることも可能です。

ユーザーデータの第一の目的は、プログラムが自身で再スタートが可能となり、リカバリ手順が適切に行われたかをチェックできることです。ユーザーデータとして有効な情報タイプは、次のものです。

- プログラムの実行日時および最終更新時刻。これにより、プログラムは端末ユーザー、コンソールオペレータまたはプリンタに適切なメッセージを送ることができ、ユーザーやオペレータはリカバリ／再スタート処理が正しく行われたかをチェックできます。特に、端末ユーザーが知らない真夜中に重大な障害が発生した場合、どの作業から再実行すべきかわかります。
- 入力データ収集日。

- バッチ番号。この番号により、管理者は端末からの再入力が必要な作業を識別し、割り当てることができます。
- 識別データ。ある場合は、上記項目に対する単なる追加データまたは参照用のデータであり、また他の場合には、再スタート地点を決める場合に基づく基本データとなります。例えば、論理順スキャンを行うプログラムでは再開するキー値が必要です。
- トランザクション番号／入力レコード位置。これにより、端末ユーザーやバッチプログラムが開始位置を決める労力を最小にできます。各トランザクションに Adabas はトランザクションシーケンス番号を返すが、次の理由からユーザーもシーケンス番号を保持したい場合があります。
 - 再スタート後、Adabas シーケンス番号がリセットされる時。
 - 各トランザクションの処理内容が複雑で、Adabas トランザクションシーケンス番号と次の入力レコードやドキュメントの位置との関係が単純ではないとき。
 - 入力エラーによりプログラムがトランザクションをバックアウトするとき。トランザクションを直ちに再入力するか（単純なキーイングミスなど）、または後で修正するために拒否するか（入力ドキュメントやレコードの基本的なエラーが発生した場合）を Adabas は判断できません。
- その他の記述データや中間データ。例えば、繰越合計、レポートのページ番号や見出し、実行統計などです。
- ジョブ／バッチ完了フラグ。すべての処理が完了後でオペレータやユーザーに知られる前にシステムがダウンする場合があります。この場合、オペレータがプログラムを再スタートしても、プログラムでは入力の終りまで実行せずに、このフラグをチェックできます。同様なことが端末から入力したドキュメントのバッチに対しても適用されます。
- 最終のジョブ／プログラム名。いくつかのプログラムが一定の順序でデータベースを更新しなければならないとき、同一のユーザー ID を使用し、順序が正しく保たれているかをチェックするためにユーザーデータを使用できます。

3 コンテナファイル

■ 一般的な知識	42
■ Adabas 論理エクステント	43
■ Adabas 物理エクステント	44
■ Adabas 物理エクステント	44
■ コンテナファイルのアクセスメソッド	46
■ Adabas ブロックサイズ	47
■ データベース自動拡張	49
■ インデックスブロックサイズ	50
■ SORT データセットの格納先	50
■ TEMP データセットの格納先	51
■ ファイルシステムまたは RAW デバイス内のコンテナファイル	51

一般的な知識

コンテナファイルは、Adabasユーティリティで作成したディスクファイルです。これらはAdabasニュークリアスおよびAdabasユーティリティによって管理されます。これらのファイルの内部構造はAdabasで構成し、保守します。したがって、非常に効率的なディスク使用アルゴリズムを使用できます。

コンテナファイル内のデータは、データベースの作成者により定義されたブロックサイズを持つデータブロックで構成されています。各コンテナタイプのすべてのデータブロックは、いわゆる相対Adabasブロック番号(RABN)によりアドレスが付与されます。このアドレスは、4バイトの符号なし整数で、その値は0よりも大きな値です。したがって、Adabasデータベースには、各コンテナタイプのブロックを最大 $2^{32}-1$ ブロック含めることができます。RABNという用語は、ブロック番号だけでなく、対応するブロックにも使用されます。

Adabasデータベースに必要なコンテナファイルはASSO、DATA、およびWORKといいます。いくつかのユーティリティについては、SORTやTEMPといった追加のコンテナファイルが必要になります。

アソシエータ (ASSO)

ASSOには、データベースやデータベースファイルの構成データが含まれます。ASSOに格納されるデータの例として次のものがあります。

- データベースの物理レイアウトおよび論理レイアウトの概要
- データベースの使用ブロックおよび未使用ブロックのリスト
- 各ファイルのレコードフィールドの記述
- 非順次データベース検索処理に使用されるディスクリプタ（検索キー）フィールドのリスト
- データベースがオフラインのときにAdabasユーティリティを使用するためのプロテクションメカニズム

データストレージ (DATA)

データストレージ（単にDATAとも呼ぶ）には、データベースのユーザーデータが含まれます。ディスクスペース要件を削減するために、Adabasはデータ圧縮技術を使用します。これは、ユーザーデータがDATAに格納される前によりコンパクトな形式に変換されるので、ストレージ要件とディスクI/Oを著しく削減することを意味しています。

WORK

Adabasニュークリアスは、バックアウトトランザクションと自動再スタートのために必要な更新ログ情報用の一時的なストレージエリアとしてWORKを使用します。

WORK のサイズは、次の条件が常に適用されるように選択します。まず、現在アクティブな最も古いトランザクションの開始時から実行されたすべての更新、削除および格納処理を対象に考えます。次に、WORK のサイズを

- (変更されたまたは削除されたすべての圧縮レコードのサイズ
- + 変更または挿入後のすべての新しい圧縮レコードのサイズ
- + 変更または削除されたすべての古いインデックス値のサイズ
- + 変更または挿入後のすべての新しいインデックス値のサイズ)

x 4 と等しいかそれ以上にします。



注意: LOB データを持つデータベースは、LOB データのサイズも考慮の対象となるため（更新されたレコードの場合は更新された LOB のサイズのみ考慮）、必然的にかなり大きな WORK サイズとなります。データベースに LOB データが含まれる場合、4 KB の WORK ブロックサイズが推奨されます。

SORT、TEMP

これらは一時的なストレージエリアおよびワークエリアとして、いくつかの Adabas ユーティリティに使用されます。事前定義された SORT と TEMP コンテナに加えて、Adabas は、ワークスペースとしてニュークリアスまたはユーティリティによって作成される一時ファイルも使用します。これらのファイルは使用後に削除されます。詳細については「一時ワークスペース」セクションを参照してください。

Adabas 論理エクステンツ

Adabas 論理エクステンツは、Adabas ニュークリアスや Adabas ユーティリティが割り当てた一連の RABN 群です。

データベースにロードされた各ファイルに対して、次の各タイプの Adabas 論理エクステンツを最低 1 つファイルに割り当てられます。

- データストレージ論理エクステンツ
(データストレージ物理エクステンツから割り当てられる)
- アドレスコンバータ論理エクステンツ
(アソシエータ物理エクステンツから割り当てられる)
- ノーマルインデックス論理エクステンツ
(アソシエータ物理エクステンツから割り当てられる)

- アッパーインデックス論理エクステント
(アソシエータ物理エクステントから割り当てられる)

追加の論理エクステントは、ファイル更新の結果として追加スペースが必要になったときに、Adabas ニュークリアスや Adabas ユーティリティによって割り当てられます。

Adabas 物理エクステント

アソシエータおよびデータストレージは、最大 16 個のエクステントを持つことができます：ASSO1 ～ ASSO16、DATA1 ～ DATA16。SORT および WORK は 2 個のエクステントを持つことができます：SORT1 と SORT2、WORK1 と WORK2。Temp は、1 個のエクステントを持つことができます：TEMP1。これらのエクステントは、連続する数字で定義される必要があります。例えば、ASSO1、ASSO2 および ASSO4 と定義すると、ASSO3 の定義がないため、ASSO4 は認識されません。

Adabas 物理エクステント

ディスクやその他の補助的な記憶媒体の領域が物理的に分離している場合など、ASSO、DATA、WORK、SORT、および TEMP の各データセットのエクステントが複数になることがあります。ユーティリティがこれらのエクステントのいずれかを参照するとき、参照用の環境変数を使用します。ASSO データセットに対する環境変数は ASSO1、ASSO2 など、DATA データセットについては DATA1、DATA2 など、WORK、SORT および TEMP についても同様です。例えば、ユーティリティが 3 つのエクステントを持つ ASSO データセットにアクセスする場合には、環境変数 ASSO1、ASSO2 および ASSO3 に、これらのエクステントの参照先を設定しておく必要があります。

ASSO、DATA、および WORK コンテナのエクステントを見つけるための検索方法は次のとおりです。

1. ASSO の場合は、環境変数 ASSO1、ASSO2 など、DATA の場合は、環境変数 DATA1、DATA2 など、WORK の場合は、環境変数 WORK1 を調べてみます。そのような環境変数が存在している場合、それは対応するコンテナエクステントのファイル名を含んでいなければなりません。
2. DBnnn.INI ファイルに該当するエントリがあるかどうか探してみます。そのようなエントリが存在している場合、それはコンテナエクステントのファイル名を含んでいなければなりません。DBnnn.INI ファイルの詳細については、*Adabas 拡張オペレーション*のドキュメントを参照してください。
3. ファイル CONTx.nnn を次のデータベースディレクトリで検索します：UNIX：
\$ADADATADIR/dbnnn、Windows：%ADADATADIR%\dbnnn。ここで、CONT は ASSO、DATA、WORK のいずれか、x はエクステント番号、nnn は 3 桁のデータベース ID をそれぞれ表します。

SORT と TEMP を使用する検索方法については、「一時ワークスペース」を参照してください。

ASSO エクステントの最大数は $(ASSO1 \text{ ブロックサイズ} - 2) / 12$ で求められます。DATA エクステントの最大数は $(ASSO1 \text{ ブロックサイズ} * 3 - 2) / 12$ で求められます。ただし、下記の条件に従えば、計算値よりも小さくすることができます。

ASSO と DATA エクステントの合計数が 2721 を超えることはできません。隣り合った DATA エクステントのブロックサイズを比較していったときに、互いのブロックサイズが異なっている回数分だけ、この最大数は 1 つずつ減っていきます。したがって、公式は次のようになります。

ASSO エクステント数 + DATA エクステント数 + (隣り合わせの DATA エクステントを比較したときにブロックサイズが違っている回数) ≤ 2721

例えば、データベースが 1 個の ASSO エクステントと 1360 個の DATA エクステントから構成されていて、隣り合った DATA エクステントのブロックサイズがどれもサイズが違っているとすると、データベースのエクステントの合計は、1 個の ASSO エクステント + 1360 個の DATA エクステント + 1359 個の異なる DATA ブロックサイズ = 2720 となります。

次の表は、コンテナファイル ASSO1 のサイズと、ASSO と DATA のエクステントの数との対応関係を例示したものです。"最良のケース" の欄のエントリは、すべての DATA エクステントが同じブロックサイズの場合の DATA エクステントの最大数です。"最悪のケース" の欄のエントリは、隣り合った DATA エクステントのブロックサイズを比較するとどれも違っている場合の DATA エクステントの最大数を示しています。

ASSO1 blocksize	max. number of ASSO extents	max number of DATA extents	
		best case	worst case
2 KB (2048 bytes)	170	511	511
3 KB (3072 bytes)	255	767	767
4 KB (4096 bytes)	341	1023	1023
5 KB (5120 bytes)	426	1279	1148
6 KB (6144 bytes)	511	1535	1105
7 KB (7168 bytes)	597	1971	1062
8 KB (8192 bytes)	682	2047	1020

SORT は最高 50 までのエクステントを持つことができます。SORT1、SORT2、...、SORT50。

WORK はエクステントを 1 個のみ持つことができます：WORK1。

TEMP は最高 10 までのエクステントを持つことができます：TEMP1、TEMP2、...、TEMP10。

PLOG および WORK のバッファサイズの大きさに関する考慮事項

WORK ブロックサイズに 8K 以下の値を指定した場合、PLOG ブロックサイズは 8K に設定されます。WORK ブロックサイズに 8K よりも大きい値を指定した場合、PLOG ブロックサイズは 32K に設定されます。

完了したトランザクションがデータベースのリカバリ時にすべて再適用されるように、ET コマンドの発行後に毎回、PLOG バッファがフラッシュされ PLOG に反映されます。この処理は、PLOG バッファがいっぱいになっているかどうかに関係なく行われます。後に ET コマンドが続く場合、PLOG バッファがいっぱいになるまで、ET コマンドの発行時に、現在の PLOG ブロックが繰り返し上書きされます。新しい PLOG ブロックは、PLOG バッファがいっぱいになったときにだけ使用されます。WORK パート 1 ブロックも同様に、自動再スタート後にデータに不整合が生じないように、WORK パート 1 バッファがいっぱいになるまで、ET コマンド発行後に、現在の WORK パート 1 ブロックが繰り返し上書きされます。

通常、PLOG と WORK のブロックサイズが大きい場合は、それらが小さい場合に比べて、PLOG バッファと WORK パート 1 バッファがいっぱいになるまでのトランザクションの量が多くなります。この場合、I/O 転送の平均サイズは大きくなりますが、ET コマンドの数は変わらないため、I/O 転送の全体的な回数は変わりません。

このような理由があるため、データレコードを圧縮して 8K 以下になる場合は、WORK ブロックサイズを 8K 以下に指定し、PLOG ブロックサイズを 8K に設定することをお勧めします。

コンテナファイルのアクセスメソッド

次に示すように、データベースコンテナファイルを作成およびアクセスするためのメソッドが 2 つ用意されています。

- デバイスタイプ非依存型アクセスメソッド
- デバイスタイプ依存型アクセスメソッド

デバイスタイプ非依存型アクセスメソッド

デバイスタイプ非依存型アクセスメソッドでは、コンテナファイルの作成時に、できるだけ物理的に連続した領域を確保するように、Adabas はオペレーティングシステムに要求します。Adabas ブロックは、物理デバイスの特性によらず連続して書き込まれます。あるコンテナファイルのアクセスメソッドとして、デバイスタイプ非依存型アクセスメソッドを選択する場合には、そのコンテナファイルの作成時にサイズをメガバイト単位で指定します。

最新のハードウェア（RAID システム、可変トラックサイズディスク、ストレージサーバーなど）では、システムにより返されるトラックサイズは、不定数であり、ディスクの物理的特性との関連性を持ちません。この場合、デバイスタイプ非依存型アクセスメソッドを使用します。

デバイスタイプ非依存型アクセスメソッドは、常に SORT および TEMP コンテナファイルへのアクセスに使用されます。

デバイスタイプ依存型アクセスメソッド

コンテナファイルのアクセスメソッドとして、デバイスタイプ依存型アクセスメソッドを選択する場合には、そのコンテナファイルの作成時に、シリンダ境界線を基点とする連続したシリンダの数を指定します。セクタ／トラックおよびトラック／シリンダの値には、システムからデバイス情報として返される値を使用します。また、クラスタサイズは、1つのシリンダに割り当て可能なサイズにする必要があります。言い換えれば、1シリンダあたりのセクタ数は、クラスタサイズの倍数になります。

このアクセスメソッドを使用してブロックがコンテナファイルに書き込まれると、このブロックがトラックに収まることが確実にになります。トラックサイズが Adabas ブロックサイズの倍数ではない場合、最後のトラックは使用されません。これにより、Adabas ブロックは1回のディスク回転で読み込まれます。

セクタ／トラック、トラック／シリンダおよびクラスタサイズを表示するには、OpenVMSDCL コマンド、"SHOW DEVICE/FULL <ディスク名>" を使用します。

Adabas ブロックサイズ

デバイスタイプ非依存型アクセスメソッドを使用する場合、DATA コンテナおよび WORK コンテナファイルのブロックサイズには、ASSO ブロックサイズの倍数を選択する必要があります。これにより、異なるコンテナファイルタイプのブロックを置き換えるときに、Adabas バッファプール内の一時的に未使用なスペースを最小化できます。

この規則を使用すると、異なる組み合わせのブロックサイズが可能になります。

例：

ASSO	DATA	WORK
3 K	6 K	6 K
2.5 K	7.5 K	7.5 K
2 K	4 K	8 K

デバイスタイプ依存型アクセスメソッドを使用する場合でも、この規則を使用することをお勧めします。このメソッドを使用するためにブロックサイズを選択する場合は、トラックの終わりの未使用のスペースが過度に大きくならないように、トラックあたりのセクタ数も考慮に入れる必要があります。

例：

トラックあたり 62 セクタのディスクの場合（すなわち、トラックサイズは 31K）のディスクの場合、コンテナファイルに選択したブロックサイズごとのトラックあたりの未使用スペース量がどうなるかを次の表に示します。

ASSO/DATA/WORK ブロックサイズ	トラックあたりの Adabas ブロック	トラックの終わり未使用スペース
4 K	7	3 K
8 K	3	7 K
3 K	10	1 K
6 K	5	1 K

PLOG および WORK のバッファサイズの大きさに関する考慮事項

WORK ブロックサイズに 8K 以下の値を指定した場合、PLOG ブロックサイズは 8K に設定されます。WORK ブロックサイズに 8K よりも大きい値を指定した場合、PLOG ブロックサイズは 32K に設定されます。

完了したトランザクションがデータベースのリカバリ時にすべて再適用されるように、ET コマンドの発行後に毎回、PLOG バッファがフラッシュされ PLOG に反映されます。この処理は、PLOG バッファがいっぱいになっているかどうかに関係なく行われます。後に ET コマンドが続く場合、PLOG バッファがいっぱいになるまで、ET コマンドの発行時に、現在の PLOG ブロックが繰り返し上書きされます。新しい PLOG ブロックは、PLOG バッファがいっぱいになったときにだけ使用されます。WORK パート 1 ブロックも同様に、自動再スタート後にデータに不整合が生じないように、WORK パート 1 バッファがいっぱいになるまで、ET コマンド発行後に、現在の WORK パート 1 ブロックが繰り返し上書きされます。

通常、PLOG と WORK のブロックサイズが大きい場合は、それらが小さい場合に比べて、PLOG バッファと WORK パート 1 バッファがいっぱいになるまでのトランザクションの量が多くなります。この場合、I/O 転送の平均サイズは大きくなりますが、ET コマンドの数は変わらないため、I/O 転送の全体的な回数は変わりません。

このような理由があるため、データレコードを圧縮して 8K 以下になる場合は、WORK ブロックサイズを 8K 以下に指定し、PLOG ブロックサイズを 8K に設定することをお勧めします。

データベース自動拡張

データベースの空き容量がなくなると、Adabas はデータベースコンテナ ASSO と DATA を自動的に拡張することができます。このための必要条件は、ニュークリアスパラメータ `OPTIONS=AUTO_EXPAND` が指定されていることです。新しいスペースを割り当てるために用いられる方法は以下のとおりです。

1. 新しいスペースを必要としているコンテナの最後のエクステントを増やそうとします。エクステントが、コンテナの新しいスペースに必要なブロックサイズと同じブロックサイズを持っている場合にのみ可能です。
2. 次のコンテナエクステントのための環境変数があるかどうかをチェックします。環境変数が存在している場合、それは次のエクステントのファイル名を含んでいなければなりません。そして、指定された位置には新規コンテナエクステントに利用可能な十分なスペースを持っていなければなりません。
3. `DBnnn.INI` ファイルがセクション `RESERVED_LOCATIONS` にエントリを含んでいるかどうかをチェックします。含んでいる場合は、指定された位置の 1 つに新規コンテナエクステントを割り当てます。`DBnnn.INI` ファイルの詳細については、*Adabas 拡張オペレーション* のドキュメントを参照してください。新しいコンテナエクステントのファイル名は、`CONTx.nnn` になります。`CONT` は ASSO または DATA、`x` はエクステント番号、`nnn` は 3 桁のデータベース ID をそれぞれ表します。
4. 新規コンテナエクステントをデータベースディレクトリに割り当てます（UNIX：`$ADADATADIR/dbnnn`、Windows：`%ADADATADIR%\dbnnn`）。新しいコンテナエクステントのファイル名は、`CONTx.nnn` になります。`CONT` は ASSO または DATA、`x` はエクステント番号、`nnn` は 3 桁のデータベース ID をそれぞれ表します。



注意:

1. ニュークリアスがアクティブなとき、ユーティリティはオンラインモードでだけ、データベースを拡張します。例外として、`ADABCK` はオフラインモードでもデータベースを拡張できます。
2. ファイルシステム内で自動拡張が行われる場合、同じ名前のファイルがすでに存在してはなりません。自動拡張が RAW セクションへ行われる場合、RAW セクションにすでにこのタイプの同じエクステント番号およびデータベース ID を持つコンテナが存在してはなりません。コンテナエクステントが `ADADEV` のみによって割り当てられているかどうか、または、コンテナが実際にデータベース内に含まれているかどうかは問題にはなりません。
3. スペース割り当てに際し明示的な `RABN` を指定する場合、要求されたすべての `RABN` がデータベースに含まれていない場合は、自動拡張は行われません。

インデックスブロックサイズ

Adabas はインデックスブロックを作成するとき、ディスクリプタ値のサイズに依存するブロックサイズによって、次のようにブロックを割り当てます。

- 253 バイトより大きなディスクリプタ値は、ブロックサイズが 16 KB 以上の大きなインデックスブロックに格納されます。
- 小さ目のディスクリプタ値は、ブロックサイズが 16 KB 未満の小規模のインデックスブロックに格納されます。

したがって、大きなディスクリプタ値を格納する場合、データベースに大規模のブロックサイズで ASSO コンテナを定義しなければなりません。

SORT データセットの格納先

SORT データセットは ASSO および DATA と同じボリュームに配置しないことをお勧めします。10 万件を超えるレコードを含むファイルを処理するとき、ディスクアーム移動を最小化するため、SORT エリアを 2 つのボリュームに分割すべきです。

小規模のデータだけを処理するときには、SORT データセットを省略することができます（例えば空ファイルのフィールドのインバート）。このとき使用する Adabas ユーティリティはインコアソートを行います。

SORT データセットは、処理する最大ディスクリプタをソートできるほど十分な大きさが必要です。将来的にデータの圧縮や圧縮解除を行うときに備えて、ADACMP または ADAULD ログにディスクリプタが並んで出力されているかどうか、SORT と TEMP の推奨サイズを確認しておいてください。

ADAINV SUMMARY 機能はメモリ常駐のソートに対して必要な SORT および LWP も表示します。



注意: ADAINV にメモリ常駐のソートを行わせたい場合、LWP パラメータがメモリ常駐ソートに十分な大きさであっても、ADAINV は最初のディスクリプタに対してファイルベースのソートを行う可能性があるため、SORT データセットを指定しないでください。これは、ADAINV がディスクリプタのサイズを前もって知らないからです。可能であれば、後続のディスクリプタは常にメモリで処理されます。

TEMP データセットの格納先

TEMP データセットは DATA および SORT を同じボリュームに設置しないことを推奨します。

TEMP データセットは、次の状況で使用されます

- ノーマルインデックスとメインインデックスのロード
- アップパーインデックスの更新

TEMP のサイズはノーマル／メインインデックスのロード時のシステムパフォーマンスと密接な関係がありますが、ロードが正常に終了するかどうかは、TEMP のサイズとは関係がありません。一方、アップパーインデックスの更新時は、必要なすべてのデータは TEMP データセットに収まる必要があります。

ADACMP および ADAULD では、ディスクリプタの概要に推奨の TEMP サイズが表示されます。

TEMP データセットは、複数ディスクリプタをインバートする場合のディスクリプタ値の中間ストレージに使用されます。

TEMP のサイズはノーマル／メインインデックスのロード時のパフォーマンスと密接な関係がありますが、ロードが正常に終了するかどうかは、TEMP のサイズや有無とは関係がありません。TEMP データセットは、最低でも 2 番目に大きなディスクリプタの格納に十分な大きさにすることをお勧めします。TEMP データセットのサイズを拡張すると、パスの数（つまり、処理されるファイルの DATA エリアが読み込まれる回数）を減らすことができます。ADAINV/ADAMUP SUMMARY 機能により、TEMP データセットに望ましいサイズが表示されます。

ファイルシステムまたは RAW デバイス内のコンテナファイル

Adabas コンテナファイルは、ファイルシステムまたは RAW デバイス（UNIX のみ）のいずれかに作成できます。

次の点を考慮する必要があります。

- 一般的に、ファイルシステムのコンテナか、RAW デバイスのコンテナのどちらが優れているかを判断することはできません。これは、Adabas の使用方法に大きく依存します。ファイルシステムにはデータをバッファできるという利点があります。つまり、ファイルシステム I/O が必ずしもディスク I/O になるとは限らず、ファイルシステムで I/O 動作が最適化される可能性もあります。一方、ファイルシステムには、RAW デバイスでは回避されるオーバーヘッドがあります。したがって、Software AG では、両方を試して、特定の環境では最高の性能を発揮する I/O システムを使用することを推奨しています。
- RAW デバイスは 2 テラバイトに制限されています。



注意: Adabas では、RAW デバイスが2テラバイト以下であるかどうかは確認されませんが、より大きなRAWデバイスを使用すると、予期しないエラーが発生する可能性があります。

- 2 テラバイトを超えるコンテナは、ファイルシステムに作成する必要があります。
- ファイルシステムでコンテナを使用し、RAW デバイスのコンテナと同様の動作を望む場合は、ADANUC パラメータ UNBUFFERED を使用することをお勧めします。
- Adabas コンテナは、ローカルディスクまたはリモートストレージサーバーに作成できます。
- ストレージサーバー上のディスクを使用する場合、Adabas が実行されているコンピュータとストレージサーバーの間のネットワーク速度によって I/O 速度が制限されることがあります。これにより、Adabas の全体的なパフォーマンスが低下する可能性があります。
- ファイルシステムのスナップショットをサポートする一部のファイルシステムでは、更新されたブロックが上書きされず、別の場所にコピーが書き込まれます。データベースに多数の更新があると、データが断片化されて、I/O パフォーマンスが大幅に低下することがあります。Software AG では、ストレージシステムのベンダーに問い合わせ、ストレージシステムでこれが発生する可能性があるかどうか、およびこれらの問題を回避するために実行できる方法について確認することを推奨しています。
- バッファプールが十分に大きい場合（ADANUC パラメータ LBP）、通常、I/O パフォーマンスは重要ではありません。これは、ほとんどの論理 I/O が物理 I/O を必要としないことが理由です。ただし、WORK および PLOG（Adabas プロテクションログ）が含まれているデバイスのパフォーマンスは重要です。これは、WORK および PLOG に、データベースの整合性を保証するために必要なログ情報が含まれていることが理由です。したがって、ET（トランザクション終了）コマンドは、ログ情報が WORK および PLOG デバイスに安全に保存されている場合にのみ完了できます。これらの理由から、Software AG では、更新の負荷が高い場合に、非常に高速なストレージデバイスを使用することを推奨しています。WORK および PLOG 用の通常のストレージデバイスを高速デバイスに置き換えると、パフォーマンスが最大 30% 向上します。もちろん、パフォーマンスの向上は、実行する Adabas コマンドの組み合わせと、ストレージデバイスの速度の差に大きく依存します。

4 一時ワークスペース

Adabas ニュークリアスまたはユーティリティがディスク上に一時ワークスペースを必要とする場合、TEMP や SORT コンテナを使用することができます。つまり、スペースが割り当てられ、その後一時作業位置において解放されます。一時作業位置は、十分な空きスペースを含むディレクトリを指定する必要があります。

ディスク上の一時ワークスペースは次の規則に従ってアクセスされます。

- TEMP コンテナ（ADAFRM ユーティリティで定義）だけが、ユーティリティ ADAINV と ADAMUP におけるディスクリプタ値の中間ストレージの一時ワークスペースとして使用することができます。

TEMP コンテナを見つけるための検索方法は次のとおりです。

1. TEMP1、TEMP2 などの環境変数が存在するかどうか確認します。このような環境変数が存在する場合、この環境変数には TEMP コンテナのファイル名が含まれています。
 2. DBnnn.INI ファイルに該当するエントリがあるかどうか探してみます。該当するエントリが存在している場合、それは TEMP コンテナのファイル名を含んでいなければなりません。DBnnn.INI ファイルの詳細については、*Adabas 拡張オペレーション*のドキュメントを参照してください。
 3. ファイル TEMPx.nnn を次のデータベースディレクトリで検索します（UNIX：
\$ADADATADIR/dbnnn、Windows：%ADADATADIR%\dbnnn）。ここで、x はエクステン
tent 番号、nnn は 3 桁のデータベース ID を表します。
- ユーティリティ ADAINV と ADAMUP でソート作業スペースを見つけるための検索方法は次のとおりです。
1. SORT1、SORT2 などの環境変数が存在するかどうか確認します。このような環境変数が存在する場合、この環境変数には SORT コンテナのファイル名が含まれています。SORT は最高 50 までのエクステン
tent を持つことができます。TEMP コンテナ（ADAFRM で作成したファイルのみ使用可能）の場合とは異なり、現在存在していないファイルは ADAINV/ADAMUP によって直接作成されます。

2. DBnnn.INI ファイルに該当するエントリがあるかどうか探してみます。該当するエントリが存在している場合、それはSORT コンテナのファイル名を含んでいなければなりません。DBnnn.INI ファイルの詳細については、*Adabas 拡張オペレーション*のドキュメントを参照してください。TEMP コンテナ (ADAFRM で作成したファイルのみ使用可能) の場合とは異なり、現在存在していないファイルはADAINV/ADAMUPによって直接作成されます。
3. 環境変数のTEMPLOC1とTEMPLOC2が存在するかどうか確認します。その環境変数が存在している場合、それらは、ユーティリティがSORT コンテナを作成しようとするディレクトリの名前を含んでいなければなりません。ファイル名は SORTpid.dbid になります。ユーティリティが処理を終了するときに、ソートファイルは削除されます。
4. DBnnn.INI ファイルに該当するエントリ (TEMPORARY_LOCATION) があるかどうか確認します。該当するエントリが存在している場合、それらは、ユーティリティがSORT コンテナを作成しようとするディレクトリの名前を含んでいなければなりません。DBnnn.INI ファイルの詳細については、*Adabas 拡張オペレーション*のドキュメントを参照してください。ファイル名は SORTpid.dbid になります。
5. データベースディレクトリにファイルを作成します (UNIX: \$ADADATADIR/dbnnn、Windows: %ADADATADIR%\dbnnn)。ファイル名は SORTpid.dbid になります。

ユーティリティが処理を終了するとき、ユーティリティによって作成されたソートファイルは削除されます。ADAINVとADAMUPは、ユーティリティの異常終了の結果として削除できなかったソートファイルを削除する、クリーンアップ処理も組み込んでいます。

- ADAFRMによるTEMPおよびSORT コンテナ作成用のファイル名検出のための検索方法は、ADAINV/ADAMUPでの検索方法の手順1~3に対応しています。
- ADANUCに必要なソートおよび他の一時ワークスペースは、次の検索方法に従ってADANUC自身が直接作成します。
 1. 環境変数のTEMPLOC1とTEMPLOC2が存在するかどうか確認します。これらの環境変数が存在している場合は、ADANUCが一時コンテナを作成しようとするディレクトリまたはRAW セクションの名前が、環境変数に含まれている必要があります。
 2. DBnnn.INI ファイルに該当するエントリ (TEMPORARY_LOCATION) があるかどうか確認します。該当するエントリが存在している場合、それらは、ADANUCが一時コンテナを作成しようとするディレクトリの名前を含んでいなければなりません。DBnnn.INI ファイルの詳細については、*Adabas 拡張オペレーション*のドキュメントを参照してください。
 3. データベースディレクトリにファイルを作成します (UNIX: \$ADADATADIR/dbnnn、Windows: %ADADATADIR%\dbnnn)。これは、Adanucが読み取り専用モードで起動していない場合にのみ適用されます。

ファイル名は NUCSRTx.dbid および NUCTMPx.dbid になります。ADANUCのシャットダウン処理中に、これらのファイルは再度削除されます。



注意: ユーティリティの一時的な場所には、RAW セクションを指定しないでください (UNIXプラットフォームのみ)。作成されたファイル名を一意にするために、一時的な場所に作成されたファイルのファイル名にPIDが追加されます。これは、RAW セクションでは実行できません。一時的な作業スペースにRAW セクションを使用する場合は、

ADAFRM を使用して RAW セクションに TEMP コンテナと SORT コンテナを作成し、それらを SORTn および TEMPn 環境変数に割り当てます。SORT コンテナと TEMP コンテナが一度に 1 つのユーティリティのみで使用されるように注意する必要があります。

5 FDT のレコード構造

■ データ定義構文	58
■ 定義オプション	61
■ サブディスクリプタ	78
■ スーパーディスクリプタ	81
■ フォネティックディスクリプタ	89
■ ハイパーディスクリプタ	90
■ 照合ディスクリプタ	93
■ 参照制約	96

この章では、データベース内のファイルのレイアウトを定義するための構文とデータ定義の使用について説明します。この入力内容は、ADAFDUユーティリティの入力であるシーケンシャルファイル FDUFDT で使用されます。

データ定義は、そのファイルのフィールド定義テーブル（FDT）を作成するのに使用されます。このテーブルは、Adabas コマンド実行中にファイル内のすべてのフィールド（またはグループ）の論理構造と特徴を判断するために Adabas が使用します。

データ定義構文

定義対象となるフィールド（またはグループ）ごとに別々のデータ定義ステートメントが要求されます。

データ定義エントリを構成するのに使用される構文は次のとおりです。

```
level-number, name [,standard_length, standard_format] [(,definition_option)...]
```

レベル番号および名前は必須です。1行内にある各定義エントリの間には空白をいくつでも挿入できます。セミコロンの後のテキストは、すべてコメントとして扱われ、セミコロンで開始された行は、コメント行として扱われます。空白行は何行あってもかまいません。

レベル番号

フィールドのグループに関しては、01 から 07 までの範囲内の 1 桁ないし 2 桁の番号が、レベル番号として使用されます。先頭にゼロは付けても付けなくてもかまいません。01 から 07 までの範囲でフィールドのレベルを定義できますが、02 以降のレベル番号を持つフィールドは、そのレベルよりも 1 つ若いレベルのグループのメンバとみなされます。

グループのレベルは、01 から 06 までの範囲で定義でき、グループの中にグループを入れることができます。グループに対してレベル番号を割り当てる場合、レベル番号を飛ばして指定することはできません。

グループ定義を行うことによって、ユーザーはそのグループ名を使用することで一連のフィールド（1フィールドのみでも可能）を参照することができます。これは一連の連続フィールドを参照するのに便利で効果的な方法です。

例

```
01,GA      ; Group
02,A1,...  ; Elementary or Multiple Value field
02,A2,...  ; Elementary or Multiple Value field
01,GB      ; Group
02,B1,...  ; Elementary or Multiple Value field
02,GC      ; Group
03,C1,...  ; Elementary or Multiple Value field
03,C2,...  ; Elementary or Multiple Value field
```

フィールド A1 および A2 は、グループ GA に属します。フィールド B1 およびグループ GC（フィールド C1 および C2 からなる）は、グループ GB に属します。ピリオド (...) は、その先にも定義が続くことを表しています。

名前

フィールドまたはグループに割り当てられる名前です。

名前の長さは2文字でなければなりません。最初の文字はアルファベット、2番目の文字はアルファベットまたは数字にする必要があります。大文字と小文字を使用できます。特殊文字は使用できません。単一の Adabas レコードでは、最大 3214 個のフィールドを定義できます。

E0からE9までの値は、編集マスクとして予約されているので使用できません（編集マスクの詳細については、『Adabas コマンドリファレンスマニュアル』の「Adabas の呼び出し」を参照）。

名前はファイル内でユニークでなければなりません。英語の前置詞および冠詞等に該当する AN、AT、BY、IF、IN、OF、ON 等の文字列は、NATURAL が使用する構文用語との混同をまねく恐れがあるので使用しないでください。

Valid Names	Invalid Names
-----	-----
AA	A (not 2 characters)
e3	E3 (edit mask)
S3	F* (special character)
wm	3M (first character not alphabetic)

標準長

フィールドの標準長（バイト単位）です。標準長は、コマンド処理中に Adabas によって使用される（デフォルトの）標準長を定義するのに使用されます。標準長を指定すると、フィールド定義テーブル（FDT）に格納され、フィールドの読み込み／更新時に、ユーザーから特に長さの指定がない場合に使用されます。

指定できる最大フィールド長は次のとおりです。

フォーマット	最大長
ALPHANUMERIC	LA、L4/LB オプション：LOB ファイルが定義されていない場合、またはフィールドがディスクリプタまたは派生ディスクリプタの親である場合は、16381 バイトです。その他の場合は、LA フィールドが 65533 バイト、L4/LB オプションが 2147483543 バイトです。その他のフィールドは 253 バイト
BINARY	126 バイト
FIXED POINT	8 バイト（1、2、4、8 バイトのみ可能）
浮動小数点	8 バイト（4、8 バイトのみ可能）
PACKED DECIMAL	15 バイト
UNPACKED DECIMAL	29 バイト
Unicode	LA、L4 オプション：UTF-8 エンコードで 16381 バイト その他：UTF-8 エンコードで 253 バイト（下記の注意参照）



注意: Unicode フィールドの長さは使用されるエンコードに依存します。内部的に、Adabas は、Unicode フィールドを格納するために、UTF-8 エンコードを使用しますが、Adabas ユーザーは、Unicode フィールドにアクセスするために他のエンコードを使用することが可能であり、このエンコードを使用するフィールドに対して決まった最大サイズはありません。

標準長をグループ名に指定することはできません。

FI オプションを使用しない限り、標準長によってフィールド値のサイズが制限されることはありません。ユーザーは、標準長以上の長さを指定した READ または UPDATE コマンドを発行することができます。

標準長が省略された場合には、そのフィールドは可変長フィールドとみなされます。可変長フィールドには標準（デフォルト）長はありません。Adabas コマンドで可変長フィールドを長さ指定なしで参照すると、フィールド値の先頭に値の長さ（長さバイトを含む）を示す 1 バイトのフィールドが付けられて返されます。ユーザーは、フィールドの更新時にこの長さバイトを指定する必要があります。

標準形式

フィールドの標準フォーマットを指定します（1 文字のコードで表す）。

コード	フォーマット
A	英数字（左詰め）
B	バイナリ（右詰め、符号なし）
F	固定小数点（右詰め、符号付き）
G	浮動小数点（浮動小数点、倍精度）
P	パック 10 進数（右詰め、符号あり）
U	アンパック 10 進数（右詰め、符号あり）
W	Unicode（下記の注意参照）



注意: フィールドは UTF-8 エンコードで内部的に格納されますが、フィールドにアクセスするとき、異なるエンコード（値が変換される）を指定することができます。

標準形式は、コマンド処理中に Adabas によって使用される（デフォルトの）標準のフォーマットを定義するのに使用されます。標準フォーマットはを指定すると、フィールド定義テーブル内に格納され、ユーザーから特にフォーマットがない場合に、フィールドの読み込み時または更新時に使用されます。

標準フォーマットはフィールドに必ず指定しなければなりません。グループ名に指定することはできません。グループにデフォルトのフォーマットは不要です。グループを参照するときに、グループ内のフィールドは必ず個々の各フィールドの標準フォーマットに従って返されます。

定義オプション

定義オプションは、2 文字のコードによって指定されます。これらのコードは、データ定義ステートメントの最後のエントリとして指定します。コードの順番はどのような順番でもかまいませんが、コードの間はコンマで区切ります。

ディスクリプタ (DE)

DE は、そのフィールドがディスクリプタであることを意味します。エントリがフィールドのアソシエタインバーテッドリストに作成されることにより、このフィールドは FIND コマンドのソートキーとして検索式で使用したり、論理的な順次読み込み処理を制御する方法として使用したりできるようになります。

1 つのファイルに対しては、最大 256 個のディスクリプタ（フォネティックディスクリプタ、サブディスクリプタ、スーパーディスクリプタおよびハイパーディスクリプタを含む）を指定することができます。

ディスクリプタオプションは慎重に使用する必要があります。特に、ファイルが大きく、ディスクリプタとみなされるフィールドが頻繁に更新される場合は、注意が必要です。

日付／時刻 (DT)

次のようなさまざまな方法で、日付／時刻の値をデータベースに保存できます。

- YYYYMMDDhhmmss フォーマットのタイムスタンプ
- Natural の日付／時刻フィールド
- UNIX time_t

DT オプションで指定した日付／時刻編集マスクは、日付／時刻値を内部的に保存する際にの日付／時刻フォーマットを定義します。

フィールド定義のフィールドオプション DT の構文は次のとおりです。

```
DT=date_time_edit_mask
```

ここで、date_time_edit_mask は次のようになります。

```
E(date_time_edit_mask_name)
```

次の日付／時刻編集マスクがサポートされています。

Date_time_edit_mask	説明	値 = 0 の場合
E(DATE)	日付：YYYYMMDD	無効な日付 - 不明
E(TIME)	時刻：HHIISS	00:00:00
E(DATETIME)	日付と時刻：YYYYMMDDHHIISS	無効な日付 - 不明
E(TIMESTAMP)	マイクロ秒精度の数値タイムスタンプ： YYYYMMDDHHIISS6	無効な日付 - 不明
E(NATTIME)	Natural の T フォーマット (0000-01-02 以降の 10 分の 1 秒)	0000-01-02:00:00:00.0 1 年目以前 - 不明
E(NATDATE)	Natural の D フォーマット (0000 年以降の日数)	0000-01-02 1 年目以前 - 不明
E(UNIXTIME)	UNIX time_t type (1970 年からの秒数)、常に UTC (協定世界時) ベース	1970-01-01:00:00:00
E(XTIMESTAMP)	1970 年以降のミリ秒の UNIX タイムスタンプ (UNIXTIME * 1000000) + マイクロ秒、常に UTC ベース	1970-01-01:00:00:00.000000

DT オプションは、フォーマット B、F、P、U でのみ使用できます。DT オプションを使用するフィールドには、日付／時刻値の格納に十分な長さを指定する必要があります。次の表は、最低限必要な長さ、およびフィールドオプション TZ (ローカルタイムゾーン) が許可されているかどうかを示しています (詳細については TZ の説明を参照してください)。

Date_time_edit_mask	フォーマットに必要な最短フィールド長				TZ オプションを許可
	B	F	P	U	
E(DATE)	4	4	5	8	×
E(TIME)	3	4	4	6	×
E(DATETIME)	6	8	8	14	○
E(TIMESTAMP)	-	-	11	20	○
E(NATTIME)	5	8	7	12	○
E(NATDATE)	3	4	4	6	×
E(UNIXTIME)	4	4	6	10	○
E(XTIMESTAMP)	7	8	9	16	○

注：

- フォーマット B と F は、E（タイムスタンプ）と一緒に使用することはできません。
- 日付／時刻編集マスクを使用する場合は、0001-01-01:00:00:00.000000 から 9999:12-31:23:59:59.999999 までの日付／時刻値を使用できます。値 0 は、0 が有効な日付／時刻値でない日付／時刻編集マスクにも使用できます。このような場合、値 0 の意味は「不明」です。値 0 が NC フィールドに指定されている場合は、有意性インジケータは -1 に設定されます。フォーマット／レコードバッファで提供されている有意性インジケータの影響は受けません。値 0 を使用して日付／時刻編集マスクを変換し、かつ変換先の日付／時刻編集マスクのフィールドで値 0 が不明な日付を表す場合、結果は常に 0 になります。ターゲットフィールドが NC フィールドの場合、有意性インジケータは -1 に設定されます。
- グレゴリオ暦が導入された 1592 年より前の日付も、グレゴリオ暦として処理されます。
 - 歴史的に存在しなかった日付を入力できます。
 - 歴史的に存在していても、先発グレゴリオ暦で定義されていない日付は、NATDATE、NATTIME、UNIXTIME、XTIMESTAMP で拒否されます。
 - 時間間隔を計算する場合、歴史的な時間間隔と等しくない結果になる場合があります。
- DATE、DATETIME、および TIMESTAMP では、ユリウス暦で歴史的に存在していて、先発グレゴリオ歴には存在しない日付も指定できます。そのような日付／時刻フィールドにどのセマンティクスを割り当てるかは、ユーザーの責任です。ただし、このような日付を別の日付／時刻編集マスクに変換しようとする、エラーが表示されます（Adabas レスポンスコード 55）。
- UNIXTIME または XTIMESTAMP フィールドに使用されるフォーマット／長さによっては、0 から 9999 年までの範囲内のサブセットのみがサポートされる可能性があります。
 - UNIXTIME または XTIMESTAMP フィールドでフォーマット B を使用すると、1970 年より前の日付／時刻値を保存できなくなります。このような場合は、負の値が必要になりますが、フォーマット B ではサポートされていません。
 - UNIXTIME で長さが 4 のフォーマット F を使用すると、2038 年 1 月 19 日より後の日付／時刻値を保存できなくなります。

- UNIXTIME で長さ 4 のフォーマット B を使用する場合は、2106 年までの日付／時刻値を指定できます。
- 長さ 10 のフォーマット U を使用する場合は、UNIXTIME の最大日付は 23 世紀です。
- 日付／時刻編集マスクのあるフィールドは、時間間隔の保存に使用しないでください。

既存ファイル内にあるフィールドへの DT オプションの追加

既存ファイルのフィールドに DT オプションを追加できます。既存のアプリケーションとの互換性を保証するために、DT オプション（ただし TZ オプションは指定していない）で定義されているフィールドは次のように処理されます。

- Adabas では、DT オプションを追加する前に、保存されたすべての値が正しい日付／時刻値になっているかどうかをチェックしません。これをチェックするのはユーザーの責任です。このようなファイルの整合性への対処は、ユーザーの責任です。
- 日付／時刻編集マスクを使用してフィールドを読み込もうとしたときに、フィールド値が有効な日付／時刻値でない場合は、Adabas レスポンスコード 55 が表示されます。日付／時刻編集マスクなしでフィールドを読み込んだ場合、Adabas は値が正しい日付／時刻値かどうかをチェックしません。
- 追加／更新コマンドのフォーマットバッファでフィールドの日付／時刻編集マスクを指定しない場合、DT オプションなしでフィールドが定義されているとみなされてフィールドが処理されます。正しい日付／時刻値のチェックは行われません。フィールドに正しい日付／時刻値が含まれていることを確認するため、日付／時刻フィールドに実行するすべての更新に対して、フォーマットバッファで日付／時刻編集マスクを使用することをお勧めします。この場合、無効な日付／時刻値は Adabas レスポンスコード 55 で拒否されます。

固定ストレージ (FI)

FI は、そのフィールドが固定量のストレージを保有し、また圧縮できないことを意味します。

データストレージ内にこのフィールド値は内部的な長さバイトなしで格納されます。

空値を含む可能性の低い 1 バイトまたは 2 バイトの長さのフィールド、および圧縮できない値を含むフィールドに対しては、FI オプションを使用することをお勧めします。

レコードの末尾に、マルチプルバリューフィールドまたはピリオディックグループ内のフィールドとして定義したフィールドには、このオプションを使用しないことをお勧めします。このようなフィールドに空値があると、ディスクストレージが無駄になり、処理時間が長くなるかの可能性があるため、フィールドの省略（または圧縮）が行われないからです。

例

	Without FI option -----	With FI option -----
Definition	01,AA,3,P	01,AA,3,P,FI
User Data	33104C	33104C
Internal Representation	0433104C (4 bytes)	33104C (3 bytes)
User Data	00003C	00003C
Internal Representation	023C (2 bytes)	00003C (3 bytes)

FI オプションの使用制限

- FI、NC、NU オプションは併用できません。
- FI オプションを可変長フィールド（標準長省略）に指定してはいけません。
- FI オプションで定義されたフィールドは、フィールドの標準長を超える値によって更新することはできません。

上位優先（HF）

アプリケーションが Adabas バイナリフィールドのフォーマット B を使用するときの方法は 2 つあります。符号なし整数値として使用する方法と、任意のビットの組み合わせと長さを持つビット文字列として使用する方法です。前者の場合は、値はハードウェアアーキテクチャに従って順序付けされ、異なる整数アーキテクチャ間で交換される場合にスワップされることが想定されています。後者の場合は、値は常に上位優先（ビッグエンディアン）値として解釈されます。つまり、異なる整数アーキテクチャ間で値が交換されても、値はスワップされません。

これら両方とも使用できるようにするために、バイナリフィールドに次のような上位優先オプション（HF）が導入されました。

- HF オプションを指定せずにバイナリフィールドが定義されている場合は、現在のハードウェアで定義されているバイト順序に従って、符号なし整数として値が解釈されます。
- HF オプションを指定してバイナリフィールドが定義されている場合、これらの値は常に上位優先値として解釈され、下位優先のプラットフォームでも同様に解釈されます。



注意: Natural は、Natural フォーマット B のフィールドが常に、HF オプションで定義されたバイナリフィールドになっていることを想定しています。Natural でオプション HF を指定せずにバイナリフィールドを使用すると、処理エラーが発生します。
- 整数アーキテクチャが異なるマシン上のデータベースから、これらのフィールドにアクセスすると、値がスワップされます。

- フィールドがディスクリプタで、かつ下位優先アーキテクチャのマシンに保存されている場合、ソート順は期待どおりにはなりません。

HF オプション付きで定義された B フィールドがスーパーディスクリプタの一部であり、結果のスーパーディスクリプタのフォーマットが英数字でない場合は、スーパーディスクリプタにも HF オプションが適用されます。つまり、スーパーディスクリプタ値は上位から保存されます。

例：

FDT に次のフィールド定義をします。

```
1,B1,4,B
1,B2,4,B,HF
```

フォーマットバッファに次の指定をします。

```
FB="B1,B2"
```

次の配列がレコードバッファとして用いられます。

```
unsigned char RB[8];
```

上位から保存するマシンでは、レコードバッファは次のコマンドで値が設定されます。

```
RB[0] = 1; /* B1 = 0x01020304 high-order first*/
RB[1] = 2;
RB[2] = 3;
RB[3] = 4;

RB[4] = 1; /* B2 = { 1, 2, 3, 4 } */
RB[5] = 2;
RB[6] = 3;
RB[7] = 4;
```

下位から保存するマシンで読み込むと内容は次のようになります。

```
RB[0] = 4; /* B1 = 0x01020304 low-order first*/
RB[1] = 3;
RB[2] = 2;
RB[3] = 1;

RB[4] = 1; /* B2 = { 1, 2, 3, 4 } */
RB[5] = 2;
RB[6] = 3;
RB[7] = 4;
```

ロング英数字（LB/L4、LA）フィールド

LB/L4（ロング英数字-4バイト長）またはLA（ロング英数字2バイト長）オプションは、英数字およびUnicodeのフィールドに指定することができます。LBとL4はどちらも同じ意味です。LB/L4またはLAオプションのうちの1つだけを特定のフィールドに指定することができます。LB/L4またはLAオプションを指定して定義されたフィールドには、下記の条件において、最大16,381バイトの長さの値を含めることができます

- フィールドがディスクリプタとして定義されている場合、
- または、派生ディスクリプタの親フィールドの場合、
- または、ファイルにLOBファイルが関連付けられていない場合、
- または、フィールドがUnicodeフィールドの場合。

 **注意:** 上記の場合、フィールド値は常にプライマリレコードに保存されます。このようなフィールドを定義する場合には、プライマリレコードがデータブロック（最大サイズ 32 KB）に収まる必要があることを考慮する必要があります。このようなフィールドは、それがレコードオーバーフローの原因とならない場合にのみ定義する必要があります。

それ以外の場合は、LA フィールドには最大 65533 バイト長の値を、LB/L4 フィールドには最大 2147483543 バイト長の値を含めることができます。

ファイルにLOBファイルが関連付けられている場合、またはフィールドがディスクリプタでも、派生ディスクリプタの親でもなく、値の長さが253バイトより長い場合は、フィールド値はLOBファイルに格納され、LOB参照が基本レコードに格納されます。フィールド値が253バイト以下の場合は、LB/L4またはLAオプションが指定されていないフィールドと同じ方法でフィールド値が圧縮されます。LAオプション指定フィールドが実際に持つことができる最大長は、圧縮レコードが格納されるブロックのブロックサイズによって制限されます。圧縮レコードは、1つのブロックに納まらなければなりません。

LAオプションを持つフィールドが可変長で更新または読み込まれるとき、その値は、レコードバッファに指定または返されます。値の前に、包括的な2バイトの長さ値（フィールド長に2を加えた値）が付きます。

L4オプションを持つフィールドが可変長で更新または読み込まれるとき、その値は、レコードバッファに指定または返されます。値の前に、包括的な4バイトの長さ値（フィールド長に4を加えた値）が付きます。

L4またはLAオプションを持つフィールドについては次の点に注意してください。

- NU、NC/NN、またはMUオプションを持つこともできます。
- PEグループのメンバにすることができます。
- FIオプションと一緒に指定できません。
- ディスクリプタフィールドにすることはできますが、この場合、フィールドがTRオプションが指定されていないと、1144の最大長（排他的フィールド長）を持つ値だけを格納すること

ができます。ディスクリプタフィールドに TR オプションが指定されている場合、1144 バイトより大きい値を格納可能ですが、インデックスのディスクリプタ値は 1144 バイトに切り捨てられます。

L4/LA の使用例

オプション	定義	ユーザーデータ (可変長) (上位優先)	ユーザーデータ (可変長) (下位優先)
L4 または LA なし	01,BA,0,A	"\x06HELLO"	"\x06HELLO"
L4 指定	01,BA,0,A,L4	"\x00\x00\x00\x09HELLO" "\x00\x00\x07\xD4" (2000 データ バイト)	"\x09\x00\x00\x00HELLO" "\xD4\x07\x00\x00" (2000 データ バイト)
LA 指定	01,BA,0,A,LA	"\x00\x09HELLO" "\x07\xD2" (2000 データバイト)	"\x09\x00HELLO" "\xD2\x07" (2000 データバイト)

マルチプルバリューフィールド (MU)

MU は、そのフィールドが 0 個、1 個、または複数の値から構成されていることを意味します。

その値は、内部的には該当フィールドに対する他のオプション指定に応じて格納されます。NU オプション定義フィールドの場合、後続の空白値は省略されます。MU オプションと NC オプションは併用できません。

ADACMP ユーティリティで使用される構文 MU(n) は指定できますが、オカレンスカウントは無視されます。

例：NU 定義のマルチプルバリューフィールド

```
Definition: 01,AA,5,A,MU,NU

Original content after file loading:

      3      L  value A      L  value B      L  value C
count      field AA1      field AA2      field AA3
```

L は L バイトを含む後続値の長さを意味します。

値 B を空値に更新した後で、

```
2   L   value A   L   value C
```

```
count   field AA1   field AA2
```

AA カウントは 2 になります。

例：NU 指定のないマルチプルバリューフィールド

```
Definition: 01,AA,5,A,MU
```

Original content after file loading:

```
3   L   value A   L   value B   L   value C
```

```
count   field AA1   field AA2   field AA3
```

値 B を空値に更新した後で、

```
3   L   value A   L   null value   L   value C
```

```
count   field AA1   field AA2   field AA3
```

AA カウントは 3 になります。

非空白圧縮 (NB)

NB オプションは、値が格納される時、末尾の空白が抑制されないことを示します。値は、常にレコードバッファに指定された値と同じ長さでデータベースに格納されます。別の文字列の始めと一致する値を持っている文字列は、常に、他の文字列より値が小さいものとして扱われます。このため、値の順序は次のようになります。

NB オプションなし

```
"xxx\x00\x00" < "xxx\x00" < "xxx" = "xxx " = "xxx " < "xxx0"
```

NB オプションあり

```
"xxx" < "xxx\x00" < "xxx\x00\x00" < "xxx " < "xxx " < "xxx0"
```

NB オプションを FI オプションとともに指定することはできません。NB オプションは、フォーマット A または W のフィールドにだけ指定できます。

NB オプション付きで定義された値が、値の長さよりも大きい固定長で読み込まれる場合、その値は、NB オプションのないフィールドの値と同様に、末尾が空白で埋められます。ただし、同じフォーマットとレコードバッファを使って更新した場合は、データベースの値が修正されます（値の末尾に空白が付加されます）。

SQL 空値表現 (NC)

NC オプションは、そのフィールドが SQL で使用される NULL 値を表現できることを意味します。このオプションが使用されると、該当フィールドに次の2つの状態のうち1つで空値を持つことができます。

- 存在しない (NULL)
- 空 (空白)

そのフィールドが空か存在しないかは、特別なフォーマットバッファ (S エlement) によって示されます。詳細については、『Adabas コマンドリファレンスマニュアル』の「フォーマットバッファ構文」を参照してください。

FI、NU および NC オプションは、いずれも共用することはできません。NC オプションは、マルチプルバリューフィールドには指定できません。また、NC オプションをピリオディックグループのメンバに指定することもできません。

例

Definition: 01,AA,2,B,NC

	Value	Blank	NULL
User S element	0	0	-1
User data	0005	0000	0000
Internal representation	0205	01	C1

非空値オプション (NN)

NN オプションで定義されたフィールドを更新または追加する場合には、必ず値を割り当てる必要があります。何らかの値または空白を各データレコードに設定する必要があり、そうしないと、Adabas にからレスポンスコードが返されます。このオプションは、NC オプションとの共用でのみ指定することができます。

例

Definition:	01,AA,2,B,NC	01,AA,2,B,NC,NN
User S element	-1	-1
User data	0000	0000
Internal representation	C1	not permitted

空値省略（NU）

NU は、そのフィールドに対する空値が省略されることを意味します。

空値省略が行われると、空値は内部的に1バイトの空フィールドインジケータで表されます。空値は格納されません。

空値のフィールドが連続していて、さらにどのフィールドにも NU オプションを指定している場合、そのフィールド群は、空値を含む連続フィールドの数を含む1バイトの空フィールドインジケータによって内部的に表現されます。したがって、NU オプションを定義したフィールドは、できる限り連続させて定義した方が効率的です。

ディスクリプタに対してNUオプションを指定した場合、そのディスクリプタの空値はインバートドリストには格納されません。したがって、このディスクリプタの空値に対してFINDコマンドを使用すると、その結果は常に該当レコードなしになり、たとえデータストレージ内に空値を持つレコードが存在していたとしても結果は同じです。

NU オプションで定義されたディスクリプタが READ LOGICAL SEQUENCE コマンド内で論理順序の制御のために使用された場合、ディスクリプタに対して空値を持つレコードは読まれません。ディスクリプタに NU オプションと UQ オプションの両方がある場合は、一意性に違反することなく、空値を複数回保存できます。

FI、NC、NU オプションは併用できません。

通常の圧縮（NU も FI も指定しない）を行うと、空値は1バイトで表現されます。

例（圧縮）

	Normal Compression -----	With FI Option -----	With NU Option -----
Definition	01,AA,2,B	01,AA,2,B,FI	01,AA,2,B,NU
User data	0000	0000	0000
Internal	01	0000	C1

Representation	(1 byte)	(2 bytes)	(1 byte)
			C1 indicates
			1 empty field
			follows

非値変換 (NV)

NV オプションで定義されたフィールドは、アーキテクチャの異なるマシンから UPDATE または READ コマンドが発行された場合に、データ変換が行われないことを意味します。

Unicode フィールドに NV オプションを指定することはできません。

例

Definition:	01,AA,2,A	01,AA,2,A,NV
EBCDIC data has	convert value	
to be stored in	of AA from EBCDIC	no conversion
a database on an	to ASCII	
ASCII machine		

ピリオディックグループ (PE)

PE は、ピリオディックグループが定義されていることを意味します。

1 つのピリオディックグループは、1 つまたは複数のフィールドから構成され、1 レコード内で可能なオカレンスの数は、0、1、または複数です。

ピリオディックグループは、01 レベルで定義されます。ピリオディックグループ内に含まれるすべてのフィールドは、その直後に 02 以上のレベルで定義されなければなりません（レベルの増分は 1、最大レベルは 7 です）。次の 01 レベル定義がピリオディックグループの終了を表します。

PE はグループ名とともにのみ指定できます。長さパラメータとフォーマットパラメータをグループ名とともに指定することはできません。1 つのピリオディックグループには、ディスクリプタやマルチプルバリューフィールド、またはその両者を含むことができますが他のピリオディックグループを含むことはできません。

例

```

01,GA,PE                ; PERIODIC GROUP GA
  02,A1,6,A,NU
  02,A2,2,B,NU
  02,A3,4,P,NU
01,GB,PE                ; PERIODIC GROUP GB
  02,B1,4,A,DE,NU
  02,B2,5,A,MU,NU        ; MU fields in PE groups
                          ; are permitted.
  02,B3                  ; Grouping of fields within
    03,B4,20,A,NU        ; PE groups is permitted.
    03,B5,7,U,NU

01,XA,PE
  02,X1,3,A,NU
  02,X2,4,U,NU
  02,YA,PE                ; Invalid. Nested periodic group not permitted.
    ^
%ADAFDU-E-PGL1  periodic group may only be defined at level 1

```

ピリオディックグループ内のフィールドに対しては NU オプションを指定することをお勧めします。これによって、圧縮が最大限活用され、そのフィールドの読み込み時、または更新時の処理時間が短縮します。

システム生成 (SY)

システム生成フィールドの値は、Adabas によって自動的に生成されます。更新コマンドまたは保存コマンドのレコードバッファで指定されている値は無視されます。システム生成フィールドは、ピリオディックグループのフィールドであってはなりません。CR オプションを持つシステム生成フィールドは、マルチプルバリューフィールドであってはなりません。

システム生成フィールドは、次の構文で定義されます。

```
SY = keyword [,CR]
```

ここで、keyword は次の値をとることができます。

TIME

作成または最後の更新のタイムスタンプ。このフィールドは DT オプションを指定して定義する必要があります。値は UTC 値として保存されます。ローカルタイム値としてこの値にアクセスする場合は、TZ オプションを指定してフィールドを定義する必要があります。

例：

```
1,CR,14,U,DE,DT=E(DATETIME),TZ,SY=TIME,CR ; Creation timestamp
```

SESSIONID

レコードが作成された Adabas ユーザーセッションの Adabas セッション ID。このフィールドはオプション A、NV を指定して定義する必要があります。推奨されるフィールド長は 28 です。フィールド定義で長さが短く指定されていると、値は切り捨てられます。レイアウトは次のとおりです。

バイト数	説明
符号なし文字 s_node[8]：	Adabas クライアントノード名
符号なし文字 s_user[8]：	Adabas クライアントユーザー ID
符号なし整数値 s_pid[4]：	プロセス ID
符号なし文字 s_timestamp[8]：	セッションタイムスタンプ：1970 年以降のマイクロ秒数（バイナリ値）



注意:

1. Adabas セッション ID は、Adabas セッションを識別するバイナリ文字列です。プラットフォーム間で変換されることはありません。特定の時点で、Adabas セッション ID は一意ですが、Adabas セッション ID は後で再利用できます。メインフレームのレイアウトは、オープンシステムとは異なります。
2. Adabas セッション ID は、lnk_set_adabas_id 関数を使用して変更できます。つまり、この Adabas セッション ID のコンポーネントに、ユーザーに関する情報が実際に含まれているという保証はありません。
3. アプリケーションが SESSIONID のコンポーネントにアクセスする必要がある場合は、上記の問題を考慮する必要があります。
4. セッションタイムスタンプは、位置合わせの理由で符号なし文字 [8] として定義されていますが、バイナリ値が含まれています。
5. Adabas バージョンが 6.2 SP2 未満または Net-Work バージョン 7.3 が使用されている場合、オープンシステムプラットフォームのセッションタイムスタンプは 0 です。これは、以前のバージョンで、タイムスタンプなしの 20 バイトのセッション ID が使用されていたためです。
6. Adabas セッション ID のレイアウトに関するこれらのルールは、オープンシステムプラットフォームにのみ適用されます。メインフレームでは、28 バイトの Adabas セッション ID もありますが、コンポーネントは異なります。詳細については、メインフレームのマニュアルを参照してください。

例（オープンシステム）：

```
1,CA,16,A,NV,SY=SESSIONID ; Node ID and client user ID of last update
UN=CA(9,16) ; Subdescriptor for user ID of last update
```

SESSIONUSER

レコードが作成または更新された Adabas ユーザーセッションのログイン ID。フィールドはフォーマット A で定義する必要があります。推奨されるフィールド長は 8 です。

SESSIONUSER フィールドの値は SESSIONID フィールドのバイト 9～16 です。



注意: SESSIONID フィールドも定義した場合は、SESSIONUSER フィールドの代わりにこのフィールドのサブディスクリプタを定義できます。SESSIONID の例を参照してください。

例：

```
1,CU,8,A,DE,SY=SESSIONUSER,CR ; Login ID of creator
```

OPUSER

レコードが作成された Adabas セッションの OP コマンドで、アディション 1 に指定されているユーザー ID。このフィールドは、フォーマット A と長さ 8 を持つ必要があります。

例：

```
1,C0,8,A,DE,SY=OPUSER ; User ID specified in OP command for last update
```

オプション CR（作成）が指定されたシステム生成フィールド

オプション CR が指定されたシステム生成フィールドの値は、レコードの作成時に Adabas によって自動的に生成されます。これらの値は、以後の更新処理で変更されることはありません。

オプション CR もオプション MU も指定されていないシステム生成フィールド

オプション CR もオプション MU も指定されていないシステム生成フィールドの値は、レコードの作成時に Adabas によって自動的に生成されます。値は、その後の更新処理のたびに更新されます。

オプション CR は指定されず、オプション MU が指定されているシステム生成フィールド

MU オプションが指定されたシステム生成フィールドには、最大 SYFMAX（ファイルパラメータ、詳細については、ユーティリティ ADAFDU のマニュアルを参照してください）の値を指定できます。

レコードが作成されるときに、MU フィールドの最初の値が生成されます。

レコードが更新されるときに、新しい値が生成され、MU フィールドの既存値の先頭に追加されます。

以後、MU インデックス > SYFMAX の値は削除されます。例えば、SY フィールドのフィールド名が SY であると仮定すると、インデックス $1 < i \leq \text{SYFMAX}$ の場合、SY(i) の新しい値は、SY(i-1) の古い値になります。

システム生成フィールドとユーティリティ ADACMP

ADAMUP で外部データを一括ロードするために ADACMP を使用した場合、システム生成フィールドの値が、これらの外部データにすでに含まれている可能性があります。そのため、ADACMP パラメータ SYFINPUT を使用して、ADACMP でのシステム生成フィールドの処理方法を指定できます。

- SYFINPUT=SYSTEM を指定した場合は、ADACMP プロセスでデータベースに挿入された場合と同様に、ADACMP によってシステム生成フィールドの値が生成されます。
- SYFINPUT=USER を指定した場合、システム生成フィールドは、SY オプションが指定されていないフィールドとして、ADACMP によって処理されます。

詳細については、ユーティリティ ADACMP のマニュアルを参照してください。

インデックスの切り捨て (TR)

TR オプションは、L4/LA オプションおよび DE オプションで指定する必要があります。

ディスクリプタ値の最大長は 1144 バイトです。ディスクリプタが TR オプション付きで定義されない場合、1144 バイトより大きいディスクリプタ値を挿入するすべての更新処理は拒否されます。TR オプションが指定された場合、これらの値をデータベースに挿入することは可能ですが、ディスクリプタ値はインデックスでは切り捨てられます。その結果、インデックスに 1144 バイトに切り捨てられた値と同じディスクリプタ値を持つレコードが複数あると、検索処理で正確な結果が返されません。これが発生すると、警告が発行されます。TR オプション付きで定義されたディスクリプタの詳細な動作は次のとおりです。

- 1144 バイトより大きいディスクリプタ値がデータベースに挿入される場合、値はインデックスで切り捨てられ、レスポンスコード 2 が返されます。
- 切り捨てにより結果が正確でない可能性のある検索処理を実行する場合、レスポンスコード 2 が返されます。

- TR オプション付きで定義されるディスクリプタによってソートする場合に、インデックスの中に、切り捨てられた結果、値が同じになっている可能性があるディスクリプタ値を持つレコードが複数あると、レスポンスコード 2 が返されます。
- インデックスの中に、切り捨てられた結果、値が同じになっている可能性があるディスクリプタ値を持つレコードが複数あると、論理順読み込み処理 (L3/6/9) でレスポンスコード 2 が返されます。
- 切り捨てチェックオプションは S1 コマンドで利用可能です。このオプションを指定する場合、サーチバッファには TR オプション付きで定義されたディスクリプタの名前を指定し、バリュースバッファには、チェックする値を指定します。値が切り捨てられない場合はレスポンスコード 0 が返され、値が切り捨てられる場合はレスポンスコード 2 が返されます。データベースでの検索処理は実行されません。

UQ オプションとともに TR オプションを指定した場合、1144 バイトに切り捨てた直後に同じ値になる、2 つの異なるディスクリプタ値を格納すると、一意性エラーが発生します。

ローカルタイムゾーン (TZ)

このオプションを指定した場合、出力フィールド値はローカル時間（またはユーザー定義のタイムゾーン）で使用されることが予期されます。内部的には、UTC で値が保存されます。このオプションは、オプション DT、および日付／時刻編集マスク名 DATETIME、TIMESTAMP、NATTIME、UNIXTIME、XTIMESTAMP とのみ一緒に使用できます。

注：

- タイムゾーンは、日付と時刻の両方の情報が使用可能な場合にのみ関係するため、TZ は、DATE、TIME、および NATDATE では使用できません。
- 定義上、UNIXTIME と XTIMESTAMP は UTC に基づいています。これらの値に使用可能な標準変換ルーチンには、タイムゾーン処理が含まれます。ただし、UNIXTIME および XTIMESTAMP フィールドを、他のいずれかの日付／時刻編集マスクを使用してローカル時間との間で変換する場合は、これらのフィールドを TZ オプションを使用して定義する必要があります。フィールドがオプション TZ なしで定義されている場合、外部値のタイムゾーンは UTC と見なされます。
- 日付／時刻編集マスク DATETIME、TIMESTAMP、または NATTIME で定義され、TZ オプションが指定されていないフィールドを使用して、UTC 時間値またはローカル時間値を保存する必要がある場合は、ユーザーが指定します。ただし、次の点を考慮してください。
 - このようなフィールドを日付／時刻編集マスク UNIXTIME または XTIMESTAMP との間で変換する場合、Adabas は内部値に UTC 時間値が含まれていると仮定します。
 - このようなフィールドを使用してローカル時間値を保存する場合、夏時間が標準時間に戻されたときに 2 回発生する時間を、一意に区別することはできません。

これらの点のいずれかが問題になる場合は、TZ オプションを指定してフィールドを定義する必要があります。

- 既存ファイルのフィールドに UTC 値が含まれていて、日付／時刻編集マスク DATETIME、TIMESTAMP、NATTIME のいずれかを使用して DT および TZ オプションを追加する必要がある場合は、ADADBAM で新しいオプションを追加してオプションを追加できます。ファイルにローカル時間値が含まれている場合は、元のファイルをアンロードして圧縮解除する必要があります。その後、新しいオプションを使用してファイルを圧縮し、再ロードできます。
- TZ オプションを使用してフィールドにアクセスし、かつフォーマットバッファで日付／時刻編集マスクを指定していない場合は、フィールド定義内の日付／時刻編集マスクがフォーマットバッファに指定されているものとして、フィールドが処理されます。

ユニークディスクリプタ (UQ)

UQ では、フィールドがユニークディスクリプタであることを示します。ユニークディスクリプタには、ファイル内の各レコードで異なる値が含まれている必要があります。ただし、1つのレコードの中では、マルチプルバリューフィールドに同じ値がいくつあってもかまいません。

UQ オプションは、DE オプションと一緒に指定する必要があります。1 ファイル内に 1 つ以上の UQ オプションにフィールドを指定できます。

サブディスクリプタ

サブディスクリプタは、エレメンタリフィールドの一部から派生したディスクリプタです。そのエレメンタリフィールドは、ディスクリプタ自体であってもなくてもかまいません。また、サブディスクリプタは、マルチプルバリューフィールドやピリオディックグループ内のフィールドに対しても定義できますが、マルチプルバリューフィールドのある特定の値やピリオディックグループのある特定のオカレンスに対しては定義することができません。

サブディスクリプタは、最終フィールド定義の後に定義しなければなりません。

サブディスクリプタは、派生元のフィールドと同じフォーマットでなければなりません。例外的に、固定小数点および浮動小数点はバイナリになり、Unicode は英数字になります。

パック値から派生したサブディスクリプタには、派生元の符号も付加されます。

サブディスクリプタ定義構文

```
name [,UQ]= field-name (from, to)
```

名前

サブディスクリプタの名前です。サブディスクリプタの名前の付け方は、Adabas の命名規則と同じです。

フィールド名

サブディスクリプタエレメントを派生させるソースフィールドの名前です。

ソースフィールドにできるものは次のとおりです。

- エレメンタリフィールド
- マルチプルバリューフィールド
- ピリオディックグループ内のフィールド
- ディスクリプタまたは非ディスクリプタ

ソースフィールドにできないものは次のとおりです。

- マルチプルバリューフィールドの特定の値
- ピリオディックグループの特定のオカレンス
- 他のスーパーディスクリプタ、サブディスクリプタおよびフォネティックディスクリプタ

ソースフィールドの定義に NU/NC オプションの指定がある場合には、サブディスクリプタにも NU/NC オプションが指定されます。したがって、ソースフィールドが空の場合には、そのサブディスクリプタも空であり、インバーテッドリスト内には含まれません。

from

サブディスクリプタ定義の開始位置となる、ソースフィールド内の相対的なバイト位置を意味します。

to

サブディスクリプタ定義の終了位置となる、ソースフィールド内の相対的なバイト位置を意味します。

英数字フィールドや Unicode フィールドに **from** と **to** を指定する場合は、左から右に向かって 1 から順番にカウントされます。

アンパック形式のフィールドやパック形式のフィールドに **from** と **to** を指定する場合は、左から右に向かって 1 から順番にカウントされます。ソースフィールドが P フォーマットで定義されている場合には、派生サブディスクリプタ値の符号は、下位バイト（バイト 1）の下位 4 ビットから取得されます。

バイナリ形式、固定小数点形式、および浮動小数点の各フィールドに **from** と **to** を指定する場合は、下位から上位に向かって 1 から順番にカウントされます。

to は 253 以下にする必要があります。

UQ

サブディスクリプタは、ユニークディスクリプタとして定義できます。

サブディスクリプタの標準長および標準形式

サブディスクリプタの標準長は、サブエレメントの長さによって定義され、また検索コマンド処理時に Adabas によって使用されます。例えば、サーチバッファがサブディスクリプタのみを含み、長さの変更が行われない場合には、この標準長が使用されます。

Unicode の親フィールドを持つサブディスクリプタ

サブディスクリプタの構成要素を W フィールドから派生させる場合、その構成要素は、W フィールド (UTF-8) の内部エンコードに基づいて作成されます。ユーザーセッションにユーザーエンコードを定義しても、そのエンコードとの間の変換は行われません。

サブディスクリプタ定義の例

例

```
Source Field Definition:    01,AR,10,A,NU
Subdescriptor Definition:   SB = AR(1,5)
```

サブディスクリプタ SB の値は、ソースフィールド AR のすべての値の最初の 5 バイト (左から右にカウント) から派生します。

AR values	SB values
-----	-----
DAVENPORT	DAVEN
FORD	FORD
WILSON	WILSO

例

```
Source Field Definition:    02,PF,6,P
Subdescriptor Definition:   PS = PF(4,6)
```

サブディスクリプタ PS の値は、ソースフィールド PF のすべての値の 4~6 バイト (右から左にカウント) から派生します。

PF values	PS values
-----	-----
(shown in hex)	(shown in hex)
00243182655C	02431C
00000000186C	0C*
78426281448D	0784262D

* PF に対して NU オプションが指定されている場合、この値に対する PS 値は作成されません。

例

Source Field Definition: 02,PF,6,P

Subdescriptor Definition: PT = PF(1,3)

PT の値は、PF のすべての値の 1~3 バイト（右から左にカウント）から派生します。

PF values	PT values
-----	-----
(shown in hex)	(shown in hex)
00243182655C	82655C
00000000186C	186C
78426281448D	81448D

スーパーディスクリプタ

スーパーディスクリプタは、複数のフィールド、フィールドの一部、またはこれらの組み合わせから派生するディスクリプタです。スーパーディスクリプタの定義に使用される各ソースフィールド（またはフィールドの一部）は、エレメントと呼ばれます。1つのスーパーディスクリプタは、2 から 20 までのエレメントから定義されます。

スーパーディスクリプタは、最終フィールド定義の後（サブディスクリプタ定義の前か後、またはその両方）に定義しなければなりません。

どのフィールドフォーマットも、スーパーディスクリプタの一部として扱われます。



注意:

1. 親フィールドの先頭から 253 バイトのみを指定できます。
2. メインフレームの Adabas データベースでは、浮動小数点フォーマット（フォーマット G）のフィールドを、スーパーディスクリプタの親フィールドとして使用することはできません。オープンシステムの Adabas データベースでは、浮動小数点フォーマットのフィールドを、スーパーディスクリプタの親フィールドとして使用できます。

スーパーディスクリプタ定義構文

```
name [,format] [,PF] [,UQ] = field-name (from, to[, encoding]),
                                     field-name (from, to[, encoding])
                                     [[,field-name (from, to[, ←
encoding]])]...]
```

名前

スーパーディスクリプタの名前です。スーパーディスクリプタの命名規則は、Adabas の名前の命名規則と同一です。

format

フォーマットは、次の場合にのみ指定できます

- すべての親フィールドにアンパックフォーマットがある場合。この場合に、A（英数字）、B（バイナリ）、または U（アンパック）を指定できます。以前のバージョンの Adabas との互換性を考慮して、デフォルトは B ですが、A または U のいずれかを指定することを強くお勧めします。これは、下位優先のプラットフォームでのスーパーディスクリプタの動作によって、予期しない結果が生じる可能性があるためです。
- • 少なくとも 1 つの親フィールドがフォーマット W の場合。この場合に、A または W を指定できます。デフォルトは A です。

PF

このオプションを指定することにより、スーパーディスクリプタにパック形式のフィールドが含まれている場合、メインフレームシステムの Adabas データベースとの互換性が保証されます。メインフレームデータベース上では、パック形式の値の符号用ハーフバイトは 0x0F であるのに対して、UNIX/Windows では 0x0C です。PF オプションを使うことで、パック形式の値がスーパーディスクリプタの中で 0x0F として格納されます。



注意: インデックス内では、符号ハーフバイトは 0x0F ですが、読み取りコマンドのフォーマットバッファにスーパーディスクリプタを指定した場合、0x0F は得られません。この場合、符号ハーフバイトは 0x0C に変換されます。これはまた、インデックス内の値のソート順が、読み取ったスーパーディスクリプタ値の英数字比較を実行して得られるソート順とは異なる可能性があることも意味します。ディスクリプタ値の範囲を読み取る場合は、L3 または L9 コマンドのサーチバッファに終了条件を指定すること、および終了条件を満たしているかどうかを確認するために読み取りディスクリプタ値をチェックしないことをお勧めします。

UQ

スーパーディスクリプタは、ユニークディスクリプタとして定義できます。

フィールド名

スーパーディスクリプタ要素の派生元にあたるソースフィールドの名前です。

ソースフィールドにできるものは次のとおりです。

- エレメンタリフィールド
- マルチプルバリューフィールド（ただし、スーパーディスクリプタごとに 1 つのみ）

- ピリオディックグループ内の任意のエレメンタリフィールド

- ディスクリプタまたは非ディスクリプタ

ソースフィールドにできないものは次のとおりです。

- マルチプルバリューフィールドの特定の値

- ピリオディックグループの特定のオカレンス

- 他のスーパーディスクリプタ、サブディスクリプタ、ハイパーディスクリプタまたはフォネティックディスクリプタ

1 つ以上のソースフィールドの定義に NU/NC オプションの指定がある場合には、スーパーディスクリプタにも NU/NC オプションが指定されます。したがって、1 つ以上のエレメントが空の場合には、スーパーディスクリプタも空でありインバーテッドリストにも入りません。

from

スーパーディスクリプタ要素の開始位置となる、ソースフィールド内の相対的なバイト位置を意味します。

to

スーパーディスクリプタ要素の終了位置となる、ソースフィールド内の相対的なバイト位置を意味します。

英数字フィールドや Unicode フィールドに from と to を指定する場合は、左から右に向かって 1 から順番にカウントされます。

アンパック形式のフィールドやパック形式のフィールドに from と to を指定する場合は、左から右に向かって 1 から順番にカウントされます。

バイナリ形式、固定小数点形式、および浮動小数点の各フィールドに from と to を指定する場合は、下位から上位に向かって 1 から順番にカウントされます。

to は 253 以下にする必要があります。

from は to 以下でなければなりません。スーパーディスクリプタ値全体の長さには制限があり、英数字の場合には 1144 バイト、バイナリの場合には 126 バイト、アンパック形式の場合には 29 バイトです。

エンコード

encoding は W フィールドにのみ使用できます。*encoding* は、Unicode エンコードである必要があります。*encoding* が指定されている場合は、指定されたバイトがフィールド値から選択される前に、フィールド値は指定されたエンコードに変換されます。

スーパーディスクリプタの標準長および標準形式

format の説明では、すべてのコンポーネントがアンパックされている、または少なくとも 1 つのコンポーネントが Unicode の、スーパーディスクリプタの標準フォーマットに関する情報を提供します。

このフォーマットは、少なくとも 1 つの親フィールドが英数字の場合は英数字になり、それ以外ではバイナリになります。

スーパーディスクリプタのフォーマットは、すべての親フィールドがアンパックされている場合にのみ指定できます。この場合、アンパックとバイナリのみ指定できます。デフォルトはバイナリです。一部の親フィールドがアンパック形式ではなく、少なくとも 1 つの親フィールドが英数字または Unicode である場合、フォーマットは英数字であり、それ以外の場合はバイナリです。一部の親フィールドがアンパック形式ではなく、少なくとも 1 つの親フィールドが英数字または Unicode である場合、フォーマットは英数字であり、それ以外の場合はバイナリです。

スーパーディスクリプタの標準長は、そのエレメントの合計で決まり、検索コマンド処理時に Adabas によって使用されます。例えば、サーチバッファに 1 つのスーパーディスクリプタ名しかなく、長さの変更が行われない場合にはこの標準長が使用されます。

Unicode の親フィールドを持つスーパーディスクリプタ

スーパーディスクリプタの親フィールドにエンコードが指定されている場合、スーパーディスクリプタエレメントは、指定されたエンコードに変換された W フィールド値から派生します。W フィールドから派生したスーパーディスクリプタエレメントにエンコードが指定されていない場合、スーパーディスクリプタエレメントの値は、W フィールドの内部エンコード (UTF-8) から作成されます。ユーザーセッションに定義されたユーザーエンコード間の変換は、スーパーディスクリプタエレメントによって行われます。変換は、エンコードが指定されたスーパーディスクリプタエレメントに対してのみ行われます。同じスーパーディスクリプタに異なるエンコードを指定することはできません。

注意：

1. フォーマット W のスーパーディスクリプタを使用する場合、そのスーパーディスクリプタ値は、通常、有効な Unicode フィールドではありません。これは、Unicode フィールドではないエレメントをスーパーディスクリプタに含めることができること、および Unicode フィールドから派生したスーパーディスクリプタエレメントが文字の途中で開始または終了することが理由です。
2. UTF-16 または UTF-32 エンコードを使用する場合は、UTF-16LE または UTF-32LE は指定せずに、常に UTF-16BE または UTF-32BE を指定することを強くお勧めします。Unicode エレメントのソート順は英数字に基づくので、期待通りの検索順は、ビッグエンディアンコーディングを使用した場合にのみ実現されます。
3. Unicode のスーパーディスクリプタエレメントに指定されているエンコードとは異なるユーザーエンコードが指定されている W フィールドの親を持つスーパーディスクリプタを使用すると、正しくない結果や未定義の結果になる場合があります。例えば、スーパーディスクリ

プタエレメント FN(1,2,UTF-16BE) を、フィールドの最初の文字がスーパーディスクリプタに含まれるように定義し、かつユーザーエンコードが UTF-8 だとします。FN の最初の文字が 3 バイトの UTF-8 文字になっている値を検索しようとする、サーチバッファの値には文字の一部のみが含まれます。⇒ スーパーディスクリプタエレメントを UTF-8 から UTF-16 に変換することはできません。

4. スーパーディスクリプタの長さを超える W フィールドの親を持つスーパーディスクリプタを読み取る際には、パディングに以下のルールが使用されます。
 - 最後の親フィールドが W フィールドの場合、指定された長さに従って、W フィールドが最後のバイトまで拡張されます。
 - 最後の親フィールドが W フィールドでない場合、スーパーディスクリプタには A フィールドの空白がパディングされます。
5. UTF-8 エンコーディングでスーパーディスクリプタの親を指定するか、エンコーディングなしで指定するかによって違いが生じます。エンコーディング UTF-8 を明示的に指定した場合にのみ、Adabas コールでスーパーディスクリプタを使用したときに、ユーザーエンコーディングへの変換またはユーザーエンコーディングからの変換が実行されます。
6. スーパーディスクリプタにアクセスするときに府明示的に指定するフォーマットは、スーパーディスクリプタのフォーマットである必要があります。ただし、スーパーディスクリプタに使用されているフォーマットに関係なく、スーパーディスクリプタの処理は同じです。

バイナリの親フィールドを含むスーパーディスクリプタ

スーパーディスクリプタにバイナリの親フィールドが含まれる場合（HF オプションなし）、スーパーディスクリプタの値は、それが使用されるプラットフォームに依存します。

- 上位優先プラットフォームでは、バイナリの構成は上位優先に従います。
- 下位優先プラットフォームでは、バイナリの構成は下位優先に従います。

ただし、下位優先プラットフォームでのスーパーディスクリプタの照合は、上位優先プラットフォームと同じです。この点で正常値への相違がありますが、値は、同じフォーマットの他の値と同様に扱われます。例えば、A フォーマットのスーパーディスクリプタをスーパーディスクリプタ長より短い長さでサーチバッファに指定した場合、スーパーディスクリプタ値の足りない部分を補うために、値に空白が埋め込まれます。

スーパーディスクリプタ定義の例

次の定義は、下記の次の 2 つの例で使用されます。

```

01, LN, 40, W, DE, NU      ;Last-Name
01, FN, 40, W, MU, NU      ;First-Name
01, ID, 4, B, NU           ;Identification
01, AG, 3, U               ;Age
01, AD, PE                 ;Address
    02, CI, 20, A, NU      ;City
    02, ST, 20, A, NU      ;Street
01, FA, PE                 ;Relatives
    02, NR, 20, A, NU      ;R-Last-Name
    02, FR, 20, A, MU, NU  ;R-First-Name

```

例

Superdescriptor definition: SD = LN(1,4),ID(1,2),AG(2,3)

スーパーディスクリプタ SD を作成します。スーパーディスクリプタの値は、フィールド LN のバイト 1~4（左から右にカウント）、フィールド ID のバイト 1~2（下位バイトから上位バイトにカウント）、およびフィールド AG のバイト 2~3（右から左にカウント）から派生します。フィールド LN にはエンコードが指定されていないので、内部で使用するエンコード UTF-8 が維持されます。すべての値は 16 進数で示されます。次の例では、照合順を制御するために値が内部的にどのように表現されるのか（図中の *internal*）、また、上位優先と下位優先のプラットフォームでは、レコードバッファまたはバリュースタックの中で、値がそれぞれどのように表現されるのかを示しています（図中の *high-order first* と *low-order first*）。

LN	ID	AG	SD
464C454D494E47	0x862143 (logical value) 00862143 (h-o first) 43218600 (l-o first)	303433	464C454D21433034 (internal) 464C454D21433034 (h-o first) 464C454D43213034 (l-o first)
4D4F52524953	0x2461866 (logical value) 02461866 (h-o first) 66184602 (l-o first)	303338	4D4F525218663033 (internal) 4D4F525218663033 (h-o first) 4D4F525266183033 (l-o first)
5041524B4552	00000000	303336	No value is stored with index
202020202020	0x432144 (logical value) 00432144 (h-o first) 44214300 (l-o first)	303030	No value is stored with index
414141414141	0x144 (logical value) 00000144 (h-o first) 44010000 (l-o first)	313131	4141414101443131 (internal) 4141414101443131 (h-o first) 4141414144013131 (l-o first)

少なくとも1エレメント（LN）がWフォーマットで定義されていて、かつ明示的なフォーマットが指定されていないので、SD のフォーマットは英数字となります。

サーチバッファに次を指定することにより、スーパーディスクリプタ値を切り捨てを指定した場合、

SD,5

サーチバッファの値

464C45D21

は、スーパーディスクリプタ値の不足部分を補うために空白が埋め込まれます。

464C45D21202020

この値が上位優先プラットフォームで指定された場合、それは問い合わせを解決するために使用される内部値でもあります。値が下位優先プラットフォームで指定された場合、対応する内部値は、次のようになります。

464C454D20212020

例

Superdescriptor definition: SY,W = LN(1,8,UTF-16BE),FN(1,2,UTF-16BE)

スーパーディスクリプタ SY をフィールド LN および FN (マルチプルバリューフィールド) から作成します。すべての値は文字フォーマットで示されます。フォーマットは W です。

LN	FN	SY
FLEMING	DAVID	FLEMD
UTF-16BE: 0046 004C 0045 004D 0049 004E 0047 0045 004D 0044	0044 0041 0056 0049 0044	0046 004C 0045 004D 0049 004E 0047 0020 0046 004C 0045 004D 0044
WILSON	JOHN SONNY	WILSJ WILSS
UTF-16BE: 0057 0049 004C 0053 004F 004E004A 0053 004A	004F 0048 004E 0053 004F 004E 004E 0059	0057 0049 004C 0053 004F 004E 004E 0059 0057 0049 004C 0053 004E

すべての値が 1 バイトまたは 2 バイトの UTF-8 文字のみで構成されている場合は、ユーザーエンコード UTF-8 も使用できます。この場合、FLEMING、DAVID 用に作成されたスーパーディスクリプタ値は、46 4C 45 4D 20 20 20 20 44 に変換されます。また、アプリケーションプログラムで、フィールド値からスーパーディスクリプタ値を作成した場合に、期待どおりに動作します。作成される値は 46 4C 45 4D 49 4E 47 20 44 41 です。スーパーディスクリプタ値の最初の要素は、0046 004C 0045 004D 0049 004E 0047 0020 に変換され、その後切り捨てられて 0046 004C 0045 004D になります。スーパーディスクリプタの 2 番目の要素は 0044 0041 に変換され、その後切り捨てられて 0044 になります。つまり、変換されたスーパーディスクリプタ値は、期待どおりの 0046 004C 0045 004D 0044 になります。

ただし、3 バイトの UTF-8 文字を含んでいる値がある場合にユーザーエンコード UTF-8 を使用すると、問題が発生します。

例

Field Definitions:		
01,PN,6,U,NU		
01,NA,20,A,DE,NU		
01,DP,1,B,FI		
Superdescriptor Definition: SZ = PN(3,6),DP(1,1)		
Source Field Values SZ Values		

(shown in hex) (shown in hex)		
PN	DP	SZ
303234363732	04	3032343604
383430333938	00	3834303300
303030303131	06	3030303006
303030303031	00	3030303000

SZ のフォーマットはバイナリになります。これは、A フォーマットで定義されたソースフィールドから派生したエレメントが1つも存在しないことが理由です。スーパーディスクリプタフォーマットがバイナリであり、最初の値にアンパック形式のゼロ (X'30') が含まれており、バイナリのゼロ (X'00') ではないため、この最終値として空値が格納されます。

例

Field Definitions:		
01,PF,4,P,NU		
01,PN,2,P,NU		
Superdescriptor Definition: SP = PF(3,4),PN(1,2)		
Source Field Values SP values		

(shown in hex) (shown in hex)		
PF	PN	SP
0002463C	003C	0002003C
0000045C	043C	0000043C
0032464C	000C	No value is stored with index
0038000C	044C	0038044C

SP のフォーマットは、A フォーマットで定義されたソースフィールドが1つも存在しないのでバイナリとなります。

例

```
Field Definitions:

01,AD,PE
02,CI,4,A,NU
02,ST,5,A,NU
Superdescriptor Definition:  XY = CI(1,4),ST(1,5)
Source Field Values          XY values
-----
CI              ST              XY

(1st occ.)      (1st occ.)
BALT            MAIN            BALTMAIN

(2nd occ.)      (2nd occ.)
CHI             SPRUCE          CHI SPRUCE

(3rd occ.)      (3rd occ.)
WASH            11TH            WASH11TH

(4th occ.)      (4th occ.)
DENV            <null value>    No value stored with index
```

少なくとも1つのエレメントが、Aフォーマットで定義されているソースフィールドから派生しているので、XY のフォーマットは英数字となります。

フォネティックディスクリプタ

フォネティックディスクリプタは、フォネティック検索を行うために定義します。FIND コマンドでフォネティックディスクリプタを使用することにより、よく似たフォネティック値を持つレコードがすべて返されます。フォネティックディスクリプタのフォネティック値は、ソースフィールド値の最初の 20 バイトが基本になっています。大文字／小文字のアルファベット値だけが認められ、数値、特殊文字および空白は無視されます。

フォネティックディスクリプタは最後のフィールド定義の後に定義されます。フォネティックディスクリプタは、サブディスクリプタまたはスーパーディスクリプタ定義の前後にあってもかまいません。

フォネティックディスクリプタの定義構文

```
pn = PHON(fn)
```

pn

フォネティックディスクリプタの名前です。前述した Adabas での命名規則に準じる必要があります。

PHON(fn)

文字 PHON はソースフィールド名の値に続き、フォネティック判定対象を意味します。

ソースフィールドは、エレメンタリフィールドかマルチプルバリューフィールドであり、英数字フォーマットで定義されていなければなりません。また、ソースフィールドは、ディスクリプタであってもなくても構いませんが、サブディスクリプタおよびスーパーディスクリプタは指定できません。

ソースフィールドはピリオディックグループ内に含まれていても構いません。

例

```
Source Field Definition:      01,AA,20,A,DE,NU
```

```
Phonetic Definition:        PA = PHON(AA)
```

ハイパーディスクリプタ

ハイパーディスクリプタは、その値がユーザー指定アルゴリズムを基にしているディスクリプタです。

値は、特殊なユーザー出口（ハイパー出口 1 から 255）にコーディングされたアルゴリズムを基にしています。各出口は、複数のハイパーディスクリプタを扱えます。各ハイパーディスクリプタは、1 つのハイパー出口に割り当てられなければなりません。

ハイパーディスクリプタ値が、Adabas ニュークリアスまたは ADAINV、ADACMP または ADAULD ユーティリティによって生成されるときは、必ずハイパー出口がコールされます。

ハイパーディスクリプタに割り当てられたオプション（PE、MU）によっては、1 つまたは複数の値が返されることがあります。入力値に割り当てられている元の ISN は変更可能です。

ハイパーディスクリプタのフォーマット、長さ、およびオプションについては、ユーザーが定義します。これらは、ハイパーディスクリプタ指定によって定義される親フィールドから取得されるわけではありません。

ハイパーディスクリプタ値を使用した検索は、標準ディスクリプタの場合と同様のやり方で行われます。

ハイパーディスクリプタ値が正しく作成されているかどうかは、ユーザーが確認する必要があります。データストレージに対するハイパーディスクリプタ値の完全性をチェックする標準的な方法はありません。ユーザーは、それぞれの値定義規則を設定し、値が正しいかどうかをチェックします。

ハイパーディスクリプタがパック形式またはアンパック形式で定義されている場合、Adabas で返された値の正当性がチェックされます。

ハイパーディスクリプタの詳細については、「ユーザー出口とハイパー出口」を参照してください。

ハイパーディスクリプタ定義構文

```
hy-name,length,format[,option... = HYPER(exit_number,parent_field
                                     [,parent field...])
```

hy-name

ハイパーディスクリプタに使用する名前です。前述した Adabas での命名規則に準じる必要があります。

長さ

ハイパーディスクリプタのデフォルト長です。

format

ハイパーディスクリプタのフォーマットです。次のフォーマットがサポートされています。

フォーマット	最大長
英数字 (A)	253 バイト
2 進数 (B)	126 バイト
固定小数点 (F)	4 バイト (常に 4 バイト)
浮動小数点 (G)	8 バイト (常に 4 または 8 バイト)
パック 10 進数 (P)	15 バイト
アンパック 10 進数 (U)	29 バイト

オプション

ハイパーディスクリプタに割り当てられるオプションです。ハイパーディスクリプタとともに使用できるオプションを次に示します。

オプション	説明
HE	サーチバリュー生成：親フィールドが 1 個の場合のみ使用できます。内部的な検索値ではなく、対応する親フィールドを指定しなければなりません。その後、Adabas は内部的な検索値に変換するためにハイパー出口を呼びます。
MU	マルチプルバリューディスクリプタ
NU	空値省略
PE	ピリオディックグループインデックス使用

オプション	説明
UQ	ユニークディスクリプタ

exit_number

ハイパーディスクリプタに割り当てられるハイパー出口番号です。この番号は、Adabas ニュークリアスおよびユーティリティによって使用され、コールされるハイパーディスクリプタユーザー出口を決定します。

parent field

1から20個までのエレメンタリフィールドの名前です。フィールド名とアドレスはユーザー出口に渡されます。

ハイパーディスクリプタ定義の例

次の定義は、下記の例に対して使用されます。

```
01, LN, 20, A, DE, NU      ;Last-Name
01, FN, 20, A, MU, NU      ;First-Name
01, ID, 4, B, NU           ;Identification
01, AG, 3, U               ;Age
01, AD, PE                 ;Address
    02, CI, 20, A, NU       ;City
    02, ST, 20, A, NU       ;Street
01, FA, PE                 ;Relatives
    02, NR, 20, A, NU       ;R-Last-Name
    02, FR, 20, A, MU, NU   ;R-First-Name
```

例

```
Hyperdescriptor definition: HN,60,A,MU,NU=HYPER(2,LN,FN,FR)
```

ハイパー出口2がこのハイパーディスクリプタに割り当てられ、名前は HN です。

ハイパーディスクリプタの長さは 60、フォーマットは英数字です。このハイパーディスクリプタは、空値省略 (NU) のマルチプルバリュー (MU) ディスクリプタです。

ハイパーディスクリプタ値は、フィールド LN、FN および FR から派生します。

例

```
Hyperdescriptor definition: SN,20,A,HE,NU=HYPER(3,LN)
```

ハイパー出口3がこのハイパーディスクリプタに割り当てられ、名前は SN です。

ハイパーディスクリプタ長は20、フォーマットは空値省略 (NU) の英数字です。ハイパー出口がコールされ、サーチおよびバリューバッファを使用する検索および読み込みコマンドのためにサーチバリューの生成 (HE) を行います。

ハイパーディスクリプタ値は、フィールド LN から派生されます。

照合ディスクリプタ

照合ディスクリプタは、Unicode フィールドの照合に ICU 照合キーを使用するディスクリプタです。ICU 照合キーは、Unicode 照合アルゴリズムと言語特有のルールに従って、変換前の文字列から生成されたバイナリ文字列です。この方法で文字列から生成された照合キー同士をバイナリとして比較する場合、お使いの環境に応じた文字列の比較になります。



注意:

1. 照合ディスクリプタ値は 1144 バイトに切り捨てられます。そのため、実際には異なる値が同じ値として扱われる可能性があることを意味しています。また、照合キーは、派生元の値よりも非常に長くなる可能性もあります。
2. Adabas バージョン 6.4 までは、ICU 3.2 が使用されていました。Adabas バージョン 6.5 以降では、ICU 5.4 がデフォルトで使用されます。既存のディスクリプタのために、ICU 3.2 も並行してサポートされています。ICU およびサポートされているバージョンの詳細については、「セクションユニバーサルエンコーディングサポート (UES)」を参照してください。

照合ディスクリプタ定義構文

```
col-name [,max_length] [,LA|L4] [,HE] [,UQ] = ↵
COLLATING(parent_field[,collation_attribute]...)
```

col-name

照合ディスクリプタに使用する名前。前述した Adabas での命名規則に準じる必要があります。

max_length

ディスクリプタ値として格納される最大バイト数。親フィールドから派生した照合キーの方が大きい場合、照合キーは切り捨てられます。デフォルト値と最大値は、ともに最大ディスクリプタ値の長さ (1144) です。

LA、L4

これらのオプションのうちの 1 つを指定すると、可変長のディスクリプタにアクセスする場合に、長さインジケータは 2 バイト (LA オプションの場合) または 4 バイト (L4 オプションの場合) になります。

HE

このオプションを指定する場合、内部照合キーではなく、検索処理のバリュースタックに対応する親フィールド値を指定する必要があります。ディスクリプタ値の読み込み (L9 コマンド) はできません。

このオプションを指定しない場合には、サーチバッファに応じて、バリュースタックに内部照合キーまたは対応する親フィールド値のどちらかを指定することができます。この場合に、ディスクリプタ値の読み込み（L9 コマンド）は可能です。

**注意:**

1. アプリケーションプログラムで ICU も使用する場合を除いて、特に必要がなければ、内部照合キーを処理することは避けるようにします。したがって、通常は HE オプションを指定してください。
2. HE オプションを使用しない場合、照合キーは、それらの親フィールド（親値の長さの 4 倍が照合キーの一般的な長さ）よりかなり大きくなることに注意してください。つまり、照合キーの長さとしては、ほとんどの場合 1 バイトでは足りないわけですが、それでも親値には LA/L4 オプションを定義せずに、照合ディスクリプタに LA または L4 オプションのどちらかを指定することを推奨します。しかし、照合キーは大きくなっても、インデックスに格納されます。これには LA オプションや L4 オプションの指定の有無は関係ありません。ただし、L9 コマンドを使って、この照合キーを可変長で読み込むことはできません。読み込もうとすると、Adabas レスポンスコード 55 が返されます。

UQ

照合ディスクリプタはユニークディスクリプタとして定義することができます。

parent_field

照合ディスクリプタの派生元となるソースフィールドの名前。フォーマットは W でなければなりません。

collation_attribute

すべての照合属性はオプションであり、それらは任意の順序で指定することができます。次の照合属性を指定することができます。

ロケール文字列

ICU によってサポートされるロケールの 1 つ。これは通常、2 文字の ISO-3166 言語コードです。続けて、"@" と "collation=" <照合指定子> を指定できます。この文字列は単一引用符で囲む必要があります。例: 'de@collation=phonebook'

デフォルトは、"（空の文字列）です。この場合、照合キーは、Unicode デフォルト照合テーブルと互換性があります。このテーブルは、言語に依存しませんが、よい結果が得られる言語がいくつかあります。

照合の強度

次のいずれかのキーワードを指定することができます。PRIMARY、SECONDARY、TERTIARY、QUARTERNARY、IDENTICAL。指定する値は、比較レベルを表します。詳細については、下記の参考資料 1 と 2 を参照してください。

PRIMARY を指定すると、大文字と小文字の違いや付加記号の有無は無視され、SECONDARY を指定すると、大文字と小文字の違いだけが無視されます。また、TERTIARY を指定すると、句読記号が無視され、QUARTERNARY を指定すると、単語に句読点が付いているか、付いていないかが区別されます。例えば、"ab" と "a-b" を比較するとして、TERTIARY を指定した場合には、どちらも同じとみなされますが、

QUARTERNARYを指定した場合には、"ab"の方が小さいとみなされます。IDENTICALを指定すると、同じ標準分解の単語だけが等しいとみなされます。

デフォルトは TERTIARY です。

ケース優先オプション

次のいずれかのキーワードを指定することができます。UPPERFIRST または LOWERFIRST。

UPPERFIRST を指定すると、大文字は小文字の前にソートされます（例えば、'AB' > 'ab'）。

LOWERFIRST を指定すると、小文字は大文字の前にソートされます（例えば、'ab' > 'AB'）。

指定されていない場合、大文字小文字の優先処理は未定義になります。

alternate_option

次のいずれかのキーワードを指定することができます。SHIFTED または NON_IGNOREABLE。

これらのキーワードは、スペースやハイフンなどの句読記号文字のソート順序に影響します。例えば、単語 "bi-weekly" と "biweekly" をソートしたときの結果は、SHIFTED を指定した場合には、両者は並びますが、NON_IGNOREABLE を指定した場合には、両者は並びません。詳細については、下記の参考資料 1 と 2 を参照してください。

デフォルトは NON_IGNOREABLE です。

case_level_option

次のいずれかのキーワードを指定することができます。CASELEVEL または NO_CASELEVEL。

CASELEVEL を指定する場合、第 2 ケースレベルと第 3 ケースレベルとの間にもう 1 つのケースレベルが形成されます。現在、ケースレベルは日本語のために使用されますが、ピンインなどの別の用途にも使用可能であることが分かっています。詳細については、下記の参考資料 2 を参照してください。

デフォルトは NO_CASELEVEL です。

french_option

次のいずれかのキーワードを指定することができます。FRENCH または NO_FRENCH。

このオプションの設定は、フランス語のように、付加記号がソートされるかどうかを決定します。

デフォルトは NO_FRENCH です。

normalization_option

次のいずれかのキーワードを指定することができます。NORMALIZATION または NO_NORMALIZATION。

このオプションの設定は、Unicode の標準的同値が考慮されるかどうかを決定します。NO_NORMALIZATION が設定されていたとしても、ICU はほとんどの世界言語の非正規化テキストを正しく処理できます。しかし、1文字で複数の付加記号を使用できる言語（例えば、ヘブライ語、タイ語、ベトナム語など）では、入力が Unicode 正規化フォーム D に従って正規化されていない場合に、このオプションの設定が必要になります。詳細については、下記の参考資料 2 を参照してください。

デフォルトは NO_NORMALIZATION です。

例

```
Collation descriptor definition: C1,HE,UQ=COLLATING(W1,'en',PRIMARY) ↵
```

固有の照合ディスクリプタは HE オプションで定義され、言語は英語、照合の強度は PRIMARY です。

```
Collation descriptor definition: C2,HE=COLLATING(W2,'de@collation=phonebook') ↵
```

照合ディスクリプタは HE オプションで定義され、言語はドイツ語、電話帳の順序が使用されます。照合の強度はデフォルト（TERTIARY）です。

ICU 照合に関する参考情報

1. Mark Davis、Ken Whistler：『Unicode Technical Standard #10, Unicode Collation Algorithm』（<http://www.unicode.org/reports/tr10/>）
2. International Components for Unicode ホームページ（<https://www-01.ibm.com/software/globalization/icu/>）

参照制約

参照制約は、2つのキー間の参照整合性を保証します。キーは、ディスクリプタ、スーパーディスクリプタ、または ISN のいずれかです。参照整合性とは、「外部キー」と呼ばれるディスクリプタのすべての値について、プライマリファイル内で「プライマリキー」と呼ばれるディスクリプタ内に値があることを意味します。プライマリキーは一意として定義され、かつオプション NC および NN が設定されている必要があります。外部キーの場合は、オプション NC を設定する必要があります。プライマリキーと外部キーのペアは、同じフォーマットになっている必要があります。ISN をプライマリキーとして使用できます。また、対応する外部キーがバイナリになっている必要があります。プライマリキーと外部キーがスーパーディスクリプタの場合は、次のようになります。

- 対応するキーはスーパーディスクリプタである必要があります。
- スーパーディスクリプタは、同じ数の親フィールドを持つ必要があります。
- 対応する親フィールドは、同じフォーマットになっている必要があります。

- 1つのスーパーディスクリプタで親フィールドが2回出現することはありません。
- 複数の外部キーで親フィールドが出現することはありません。
- すべての親フィールドに、オプション NC が必要です。
- プライマリキーのすべての親フィールドに、オプション NN が必要です。
- プライマリキーのスーパーディスクリプタは、一意になっている必要があります。

プライマリキー値が変更された場合に外部キーレコードで実行される参照アクションを指定できます。

参照制約は、外部キーが属するファイルに追加されます。

参照制約の構文

```
Constraint-name = REFINT(foreign-key, primary-file,
    primary-key[/referential-action[,referential-action]])
```

foreign-key

外部キーフィールドの名前です。

primary-file

プライマリキーが属するファイルの番号です。

primary-key

プライマリキーの名前です。

referential-action

次のいずれかのキーワードです。

キーワード	説明
DC	削除時にカスケード：プライマリファイル内のレコードが削除されると、外部キーとしてプライマリキーを含んでいるレコードも削除されます。これらのレコードに、参照制約のプライマリキーも含まれている場合は、これらのキーに対して、対応する参照アクションも実行されます。
DX	削除時にアクションなし（デフォルト）：プライマリファイル内のレコードを削除する際に、そのプライマリキーを外部キーにしているレコードが存在していない必要があります。存在していると、Adabas レスポンスコード 196 で削除処理が失敗します。
DN	削除時に空値を設定：プライマリファイル内のレコードが削除されると、そのプライマリキーを外部キーにしているレコードで、外部キーフィールドが空値に設定されます。オプション NN を使用して外部キーフィールドが定義されている場合は、このオプションは指定できません。
UC	更新時にカスケード：プライマリファイルのレコードでプライマリキーが更新されると、そのプライマリキーを外部キーにしているレコードで、当該外部キーが新しいプライマリキー値に設定されます。外部キーが参照制約のプライマリキーにもなっている場合は、対応する参照アクションもこれらのキーに対して実行されます。

キーワード	説明
UX	更新時にアクションなし（デフォルト）：プライマリファイル内のレコードでプライマリキーを更新する際に、古いプライマリキー値を外部キーにしているレコードが存在していない必要があります。存在していると、Adabas レスポンスコード 196 で更新処理が失敗します。
UN	更新時に空値を設定：プライマリファイル内のレコードでプライマリキーが更新された場合、そのプライマリキーを外部キーにしているレコードで、外部キーフィールドが空値に設定されます。オプション NN を使用して外部キーフィールドが定義されている場合は、このオプションは指定できません。

参照制約に対しては、最大1つの削除処理と最大1つの更新オプションを指定できます。ISN をプライマリキーとして参照する制約の場合は、「削除時にカスケード」および「更新時にアクションなし（DC、UX）」アクションのみが可能です。

例：

```
Primary key definition in file 9:      1, AA,8,A,DE,UQ,NC,NN
Foreign key definition in file 12:    1, AC,8,A,DE,NC

HT=REFINT(AC,9,AA)
HT=REFINT(AC,9,AA/UC)
HT=REFINT(AC,9,AA/UC,DN)
```

6 ディスクリプタの定義

■ ADAINV の処理に関する考慮事項	100
----------------------------	-----

ADAINV の処理に関する考慮事項

新しいディスクリプタの生成

新しいディスクリプタを定義すると、フィールド定義テーブル（FDT）が修正され、インバーテッドリストが作成されます。新しいメインインデックスおよびノーマルインデックスのブロックが、新しいディスクリプタのインバーテッドリストエントリの格納に必要です。ADAINV では、一回の実行につき複数のディスクリプタを設定できます。

ADAINV は、1つのディスクリプタの新しいノーマルインデックスとメインインデックスをディスクリプタ単位で構築します。このパスの間は、アッパーインデックスの中のリンキングエントリはまだ存在していないので、新しいインバーテッドリストのどれもアクセスできません。すべての新規ディスクリプタのノーマルインデックスとメインインデックスが構築されると、FDT が更新され、対応するエントリがアッパーインデックスに追加されます。

ノーマルインデックスとメインインデックスのロード

ファイル内のすべてのデータレコードは、新しいノーマルインデックスとメインインデックスの構築のために読み込まなければなりません。各レコード内で、新しいディスクリプタに関するフィールドは、ディスクリプタ値とその ISN の生成に使用されます。これらの値は、昇順のディスクリプタ値と ISN に従ってソートされます。ソートの出力は、新しいノーマルインデックスとメインインデックスの構築に使用されます。ユニークオプションで定義されたディスクリプタは、ノーマルインデックスがディスクリプタ値ごとに 1つのみの ISN を持っているかどうかチェックされます。複数の ISN が発見されると、矛盾する ISN がエラーログに書き込まれ、UQ_CONFLICT が RESET の場合に FDT 内のユニークフラグがリセットされ、処理は続行します。UQ_CONFLICT が ABORT に設定されているか省略されている場合、ADAINV は異常終了します。

ディスクリプタ値のソートに加えて、データレコードの読み込みは、膨大な入出力のために非常に時間を要します。そのため、1回の実行で大量のディスクリプタを作成する場合、ADAINV は、データストレージを介した読み込みに必要なパス数の最小化を試みます。データストレージを介した最初のパスでは、1つのディスクリプタの値は直接ソートに渡されます。2つの追加のディスクリプタ値が存在すれば TEMP データセットに書き込まれ、残りのディスクリプタの他のすべての値があれば、それらの合計サイズおよび量が累積されます。このデータの累積は、データストレージを介する残りのパスを最適化します。各パス中に、並行して TEMP を使用するディスクリプタの数が大きくなるほど、インバート処理も速くなります。ADAINV は、実行の最後で、必要とされたパスの合計数を表示します。

ファイルのロード時に指定したパディングファクタに従って、すべてのインデックスブロックに値が設定されます。新しいインデックスブロックは、既存のエクステントから必要に応じて割り当てられます。これらのブロックがすべて使用された後、一括更新ユーティリティ ADAMUP で説明する規則にしたがって自動拡張が実行されます。

拡張が成功すれば、上記の処理が続行し、失敗すれば、ADAINV はエラーメッセージを発行して終了します。

アッパーインデックスの更新

ノーマルインデックスとメインインデックスがディスクリプタ単位で構成されるのに対し、インデックスレベル3以上のアッパーインデックスは、すべてのディスクリプタを含みます。新しいメインインデックスにリンクするために、新しいメインインデックスブロックごとに、1エントリがアッパーインデックスに作成されなければなりません。新しいエントリは、次の2つの方法で作成できます。

- 更新がレベル3ブロックのみに影響する場合は、新しいエントリは既存のアッパーインデックスに直接挿入されます。
- ブロック分割またはレベル3を超えるレベルでの更新が必要になると、アッパーインデックス全体が再構築されます。ファイルのロード時に、指定したパディングファクタに従って、再度、値が設定されます。既存のインデックスブロックと、あらかじめ割り当てられているブロックは、新しくブロックが割り当てられるまで使用されます。新しいブロックが必要な場合は、ノーマルインデックスとメインインデックスのロードの説明にある手順に従ってください。

ディスクリプタの解放

ディスクリプタを解放すると、フィールド定義テーブル（FDT）が修正され、アクセスパスが削除されます。ディスクリプタに対するインバーテッドリストエントリを持つ、すべてのメインインデックスブロックとノーマルインデックスブロックは、解放されますが再使用できません。任意数のディスクリプタを1回の実行で解放できます。

アッパーインデックスの更新

ディスクリプタが解放されると、対応するエントリをアッパーインデックスから削除しなければなりません。これらのエントリは、次の2つの方法で削除できます。

- 更新がレベル3ブロックのみに影響する場合は、新しいエントリは、既存のアッパーインデックスから直接削除されます。
- ブロックが空になるか、レベル3を超えるレベルの更新が必要になると、アッパーインデックス全体が再構築されます。ファイルのロード時に指定したパディングファクタに従って、再度、値が設定されます。既存のインデックスブロックと、あらかじめ割り当てられているブロックは、新しくブロックが割り当てられるまで使用されます。新しいブロックが必要な場合は、ノーマルインデックスとメインインデックスのロードの説明にある手順に従ってください。

メインインデックスとノーマルインデックスの解放

原則的には、対応するエントリを FDT およびアップパーインデックスから削除するだけで、ディスクリプタを解放できます。下のインデックスレベルへのリンクを無効にすれば、メインインデックスとノーマルインデックスは実質的に削除されます。必須ではありませんが、ADAINV は、旧ディスクリプタのすべてのノーマルインデックスブロックとメインインデックスブロックを物理的にクリアします。その結果としてオーバーヘッドが発生しますが、空ブロックはダンプされないため、データのセキュリティおよびバックアップユーティリティ ADABCK のパフォーマンスが向上します。

インバーテッドリストの整合性のチェック

インバーテッドリストは、ファイル内に定義されたエレメンタリである、サブディスクリプタ、スーパーディスクリプタ、ハイパーディスクリプタおよびフォネティックディスクリプタごとに Adabas によって維持管理されます。これらの整合性を維持するには、次のことを確認する必要があります。

- インバーテッドリスト内の各 ISN が既存のデータレコードに関連付けられており、このデータレコードが正しいものであること。
- データストレージ内の各レコードが、その ISN および生成されたディスクリプタ値エントリによってインバーテッドリスト内に表されていること。

ディスクリプタの検証時には、ADAINV は、ノーマルインデックスのロードをシミュレートし、ソートの出力とインバーテッドリストの内容を照合します。これは、一度の実行で前述した点の両方をチェックし、一意性の矛盾を検出します。発見されたすべての矛盾は出力されます。このとき、ファイルは修正、変更されません。

拒否されたデータレコード

ADAINV によって拒否されたすべてのレコードは、ADAINV エラーファイルに書き込まれます。このエラーファイルの内容は、ADAERR ユーティリティを使用して表示します。レコードの中に印刷できない文字が含まれているため、オペレーティングシステム標準の印刷ユーティリティでエラーファイルを印刷しないでください。

詳細については、『Adabas ユーティリティ』の「ADAERR」を参照してください。

7 ユーティリティの使用

■ 入出力デバイスの割り当て	104
■ ユーティリティの実行 (UNIX)	104
■ ユーティリティの実行 (Windows)	108
■ ユーティリティのリモート実行	111
■ ユーティリティ構文	111
■ 単一機能ユーティリティと多機能ユーティリティ	114
■ ユーティリティの終了	114
■ エラー処理	115
■ Adabas シーケンシャルファイル	115
■ ADAMUP および ADAINV の実行の最適化	124
■ ニュークリアスとユーティリティの間の同期	126

なお、プラットフォーム（UNIXおよびWindows）によって操作に大幅な相違がある場合には、特定のセクションが設けられています。

入出力デバイスの割り当て

ユーティリティに対するすべてのコマンドは *stdin* または *SYS\$INPUT* から読み込まれ、出力は *stdout* または *SYS\$OUTPUT* に送られます。標準入出力は、通常オペレーティングシステムでサポートされているファイルに対して向けられます。デフォルトでは、指定されているユーティリティパラメータは標準出力に含まれません。ただし、環境変数 *ADAPARLOG* を *YES* に設定すると、パラメータが対話形式で指定されていない場合に、パラメータの仕様が *stdout/SYS\$OUTPUT* にコピーされます。

ユーティリティが1つ以上の出力ファイル进行处理する場合、ユーティリティを開始する前に、各出力ファイルに対応する環境変数／論理名を論理ファイル名に設定する必要があります。環境変数／論理名の割り当てに関する詳細については、当該ユーティリティの記述部分を参照してください。ユーティリティから出力されるファイルと、そのファイルに割り当てられる環境変数／論理名については、各ユーティリティの説明の初めの方に記載されています。ユーティリティを起動する前に出力ファイルの環境変数／論理名を設定しないと、出力ファイルは現在のディレクトリに作成され、出力ファイル名は論理名と同じになります。コマンドログファイルまたはプロテクションログファイルの環境変数／論理名がユーティリティ開始前に設定されていないと、出力ファイルは現在のディレクトリに作成され、出力ファイル名はシーケンス番号付きの環境変数／論理名と同じになります。

いくつかのユーティリティは、ユーザー出口やハイパー出口に対しても環境設定を必要とします。詳細については、「ユーザー出口とハイパー出口」セクションを参照してください。

ユーティリティの実行（UNIX）

ユーティリティの実行形態には、対話形式とバックグラウンド形式があります。

前提条件

ユーティリティを実行するためには必要な権限を持たなければなりません。インストールでは、次のように権限を設定します。Software AG 製品管理者ユーザー（例えば、*sag*）およびそれに相当するグループに属しているすべてのユーザー（例えば、*sag*）によりユーティリティは実行されます。このグループが、現在の所属グループでない場合は、ユーティリティ（*ADANUC*を含む）を実行する前に *newgrp* コマンドでグループを変更することをお勧めします。グループを変更しないと、他のユーザーに権限の問題が生じる可能性があります。これは、ユーザーが作成した IPC リソース用のグループがユーティリティを実行するユーザーの現在のグループとなっていて、一部のIPCリソースはそのグループにしかアクセスを許可していないことが理由です。

使用するファイル名がデフォルト値と異なる場合は、ユーティリティを実行する前に、必要な環境変数を設定する必要があります（この章の「環境変数」を参照）。

プログラム dbgen は、それらが新規データベースを作成するときに、assign.bsh（Bourne シェルおよび Korn シェルの場合）または assign.csh（C シェルの場合）というファイルを生成します。このファイルの中の情報がシェルに読み込まれ、コンテナファイル、コマンドログ、およびプロテクションログの環境変数が設定されます。このファイルを使って環境変数を設定する場合は、ワーキングディレクトリをデータベースディレクトリに変更してから、次のコマンドを入力してください。

```
. ./assign.bsh
```

コンテナファイルに対して外部的に環境変数を設定していない場合、ユーティリティは、コンフィグレーションファイル Adabas.INI および DBxxx.INI を介して設定を抽出します（『Adabas 拡張オペレーションセクション』を参照）。

ユーティリティの入力行にコントロールステートメントを入力する場合、入力できる内容は、パラメータの設定と実行対象の機能のいずれか、または両方を定義する文字列です。

機能とパラメータに関する説明はドキュメントに記載されていますが、ヘルプ機能のメッセージの中にも出力されます。

ユーティリティのインストール先はディレクトリ \$ADAPROGDIR です。このディレクトリを PATH 環境変数に定義している場合は、オペレーティングシステムプロンプトにその名前を指定して、ユーティリティを直接実行できます。定義していない場合は、ユーティリティ名の前に「\$ADAPROGDIR」を指定する必要があります。次からの例では、ディレクトリ名 \$ADAPROGDIR が PATH 環境変数に定義されていることを前提にしています。

対話形式でのユーティリティの実行

ユーティリティを対話形式で実行する場合は、オペレーティングシステムのプロンプトでユーティリティ名を入力し、Return キーまたは Enter キーを押します。

ユーティリティが起動すると、日付、時刻、およびバージョン番号から成るメッセージが表示されます。次に、ユーティリティ名とコロンの続く部分に、コントロールステートメントを入力します。

```
%ADAREP-I-STARTED, 18-JUL-2005 11:39:49, Version 5.1.1  
adarep:
```

対話モードの場合、入力行の開始位置は、ユーティリティのプロンプトのすぐ後になります。

入力行が物理的に1行におさまらず、複数行にわたって入力する必要がある場合は、各行の終わりにバックスラッシュ（「\」）と改行コードを続けて入力します。ユーティリティの画面では、継続行の開始位置にプロンプト（「>」）が表示されます。

コールレベルでのユーティリティの実行

コントロールステートメントにはコールレベルを指定することもできます。ADAREP ユーティリティの例では、次のようになります。

```
adarep dbid=20 summary
```

また、コントロールステートメントを入力ファイルからリダイレクトすることも可能です。例えば、`rep.in` というファイルが次の行で構成されているとします。

```
dbid=20  
summary
```

この場合、次のように指定すれば、この2つのコントロールステートメントを指定して ADAREP を実行できます。

```
adarep <rep.in
```

ユーティリティが生成する情報の出力先は、通常、`stdout` ですが、リダイレクトして変更することもできます。したがって、次のように指定すれば、サンプルの ADAREP コールの出力は `rep.out` というファイルにリダイレクトされます。

```
adarep <rep.in >rep.out
```



注意:

1. UNIX シェルはカッコ、一重引用符および二重引用符などの一部の文字に対して特殊な取り扱い方法がある点に注意してください。これは、パラメータの値に1つ以上のこれらの文字が含まれている場合、適宜パラメータの指定を変更する必要があることを意味します。
2. 対話型入力では、空白は改行と等価です。パラメータ値に空白が必要な場合は、バックスラッシュを先行させるか、一重または二重引用符で囲みます。
3. 環境変数 `ADAPARLOG` を `YES` に設定すると、対話形式で指定されたとおりにパラメータが表示されるので、パラメータの指定のエラーを見つけるのに役立つことがあります。

例

```
adadcu fields 'NEW RECORD',AA,AB end_of_fields
```

このコントロールステートメントは、意味としては次のようになるため無効です。

```
adadcu  
fields  
NEW RECORD,AA,AB  
end_of_fields
```

引用符が必要です。

```
adadcu fields \'NEW RECORD\',AA,AB end_of_fields
```

このコントロールステートメントは、意味としては次のようになるため無効です。

```
adadcu
fields
'NEW
RECORD',AA,AB
end_of_fields
```

リテラルは 1 行内に指定する必要があります。

```
adadcu fields \'NEW\ RECORD\',AA,AB end_of_fields
```

このコントロールステートメントは、意味としては次のようになるため有効です。

```
adadcu
fields
'NEW RECORD',AA,AB
end_of_fields
```

```
adadcu "fields 'NEW RECORD',AA,AB end_of_fields"
```

このコントロールステートメントは、意味としては次のようになるため無効です。

```
adadcu
fields 'NEW RECORD',AA,AB end_of_fields
```

fields および end_of_fields は別の行で指定する必要があります。

```
adadcu fields "'NEW RECORD'",AA,AB end_of_fields
```

または

```
adadcu fields "'NEW RECORD',AA,AB" end_of_fields
```

これらの両方のコントロールステートメントは、意味としては次のようになるため有効です。

```
adadcu
fields
'NEW RECORD',AA,AB
end_of_fields
```

コマンド行から標準入力へのパラメータ入力の切り替え

ユーティリティのパラメータ入力をコマンド行から標準入力に切り替える場合は、コマンド行に指定した最後のパラメータの後に「+」（プラス記号）を入力します。

例：

```
adafdu dbid=35 file=36 + <employee.fdu
```

ユーティリティの実行（Windows）

ユーティリティの実行形態には、対話形式とバックグラウンド形式があります。

インストールした Adabas バージョンの Adabas ユーティリティを実行するために、いくつかの環境変数を設定しなければなりません。これは、**Adabas Server Command Prompt** を選択して実行します。

標準のコマンドプロンプトから Adabas コマンドを実行する場合は、「インストールの完了」の説明に従って、「ローカル」の Adabas サーバー環境を構成する必要があります。

- **[Set Adabas System Environment]** を選択すると、Adabas で必要とされる環境変数をシステム環境変数として設定できます。**[Unset Adabas System Environment]** を選択すると、これらの環境変数を削除して元に戻すことができます。
- 環境変数をシステム環境変数として設定しない場合は、現在の Adabas バージョンの bin フォルダにある *startenv.cmd* コマンドを実行します。このフォルダの名前は、**[Adabas Command Prompt]** を右クリックして **[Properties]** を選択すればわかります。

使用するファイル名がデフォルト値と異なる場合は、ユーティリティをオフラインで実行する前に、必要な環境変数を設定する必要があります（この章の「環境変数」を参照）。

コンテナファイルに対して外部的に環境変数を設定しない場合、ユーティリティはコンフィグレーションファイル Adabas.INI および DBxxx.INI を経由してそれ自身の設定を抽出します。

ユーティリティの入力行にコントロールステートメントを入力する場合、入力できる内容は、パラメータの設定と実行対象の機能のいずれか、または両方を定義する文字列です。

機能とパラメータに関する説明は、このマニュアルに記載されていますが、ヘルプ機能のメッセージの中にも出力されます。

ユーティリティは、インストールディレクトリのサブディレクトリ「Adabas」にあります。このディレクトリが PATH 環境変数に定義している場合は、オペレーティングシステムプロンプトにその名前を指定して、ユーティリティを直接実行できます。定義していない場合は、ユーティリティ名の前に「%ADAPROGDIR%」を指定する必要があります。以下の例では、ディレクトリ名 %ADAPROGDIR% が PATH 環境変数に定義されていることを前提にしています。

対話形式でのユーティリティの実行

ユーティリティを対話形式で実行する場合は、オペレーティングシステムのプロンプトでユーティリティ名を入力し、Return キーまたは Enter キーを押します。

ユーティリティが起動すると、日付、時刻、およびバージョン番号から成るメッセージが表示されます。次に、ユーティリティ名とコロンに続く部分に、コントロールステートメントを入力します。

```
%ADAREP-I-STARTED, 18-JUL-2005 11:39:49, Version 5.1.1
adarep:
```

対話モードの場合、入力行の開始位置は、ユーティリティのプロンプトのすぐ後になります。

入力行が物理的に1行におさまらず、複数行にわたって入力する必要がある場合は、各行の終わりにバックスラッシュ（「\」）と改行コードを続けて入力します。ユーティリティの画面では、継続行の開始位置にプロンプト（「>」）が表示されます。

コールレベルでのユーティリティの実行

コントロールステートメントにはコールレベルを指定することもできます。ADAREP ユーティリティの例では、次のようになります。

```
adarep dbid=20 summary
```

また、コントロールステートメントを入力ファイルからリダイレクトすることも可能です。例えば、`rep.in` というファイルが次の行で構成されているとします。

```
dbid=20
summary
```

この場合、次のように指定すれば、この2つのコントロールステートメントを指定してADAREPを実行できます。

```
adarep <rep.in
```

ユーティリティが生成する情報の出力先は、通常、`stdout` ですが、リダイレクトして変更することもできます。したがって、次のように指定すれば、サンプルのADAREP コールの出力は `rep.out` というファイルにリダイレクトされます。

```
adarep <rep.in >rep.out
```



注意: 対話型入力では、空白は改行と等価です。パラメータ値に空白が必要な場合は、二重引用符で囲みます。

例

```
adadcu fields 'NEW RECORD',AA,AB end_of_fields
```

このコントロールステートメントは、意味としては次のようになるため無効です。

```
adadcu
fields
'NEW
RECORD',AA,AB
end_of_fields
```

リテラルは 1 行内に指定する必要があります。

```
adadcu "fields 'NEW RECORD',AA,AB end_of_fields"
```

このコントロールステートメントは、意味としては次のようになるため無効です。

```
adadcu
fields 'NEW RECORD',AA,AB end_of_fields
```

fields および end_of_fields は別の行で指定する必要があります。

```
adadcu fields "'NEW RECORD',AA,AB end_of_fields
```

または

```
adadcu fields "'NEW RECORD',AA,AB" end_of_fields
```

これらの両方のコントロールステートメントは、意味としては次のようになるため有効です。

```
adadcu
fields
'NEW RECORD',AA,AB
end_of_fields
```

コマンド行から標準入力へのパラメータ入力の切り替え

ユーティリティのパラメータ入力をコマンド行から標準入力に切り替える場合は、コマンド行に指定した最後のパラメータの後に「+」（プラス記号）を入力します。

例：

```
adafdu dbid=35 file=36 + <employee.fdu
```

ユーティリティのリモート実行

オプション LOCAL_UTILITIES を Adabas データベースのアクティブニュークリアスに設定していない場合、ローカルおよびリモートマシンのアーキテクチャが同一（例えば、両方のマシンがバイトスワッピングを使用）であれば、そのデータベースは、Software AG の製品 Entire Net-Work や NET-WORK ACCESS を使用することにより、特定の Adabas ユーティリティでリモートアクセスすることができます。データベースにリモートアクセスできるユーティリティは次のとおりです。

- ADACMP
- ADADBM
- ADAFDU
- ADAREC
- ADAREP
- ADATST
- ADAULD

ADADBM と ADAULD については、ニュークリアスが実行しているときに許される機能だけがリモートで実行することができます。



注意: ADAM ファイルを定義しているか、または FORMAT パラメータを指定している場合、ADAFDU はリモートで実行することができません。

ユーティリティ構文

関数と引数

関数の引数はパラメータと呼ばれます。パラメータによって、持つ値はさまざまです。論理的に相互に切り替わる、2つの値のみを持ち得るパラメータはスイッチと呼ばれます。

関数は、予約キーワードで表現されます。使用される関数タイプは次の3つです。

- 引数を必要としない関数。
- 1つの引数を必要とする関数。
- 複数の引数を必要とする関数。

引数を必要としない関数はスイッチであり、LOCK や UNLOCK などのキーワードによって関数の有効または無効を制御できます。

引数は、次のタイプに分けられます。

■ 文字列

文字列とは、連続した ASCII 文字を指します。ただし、ASCII 文字のうち、コンマ (,)、改行文字 (<CR>)、セミコロン (;)、および丸カッコ ("("と")") は文字列としては扱われず、文字列のデリミタとして扱われます。また、空白やタブが含まれていれば削除されます。バックスラッシュ (\) は、エスケープ文字として使用できます。

ほとんどの文字列は大文字に変換されます。等号の代わりに、コロン「:」を指定することもできます。違いは、コロンの後の文字列は大文字に変換されないことです。ユーティリティの構文記述でコロンが許可されない他のパラメータにもコロンを指定できますが、コロンは上記の場合にのみ使用することをお勧めします。そうしないと、パラメータ指定の結果が未定義になることがあります。

例：

```
adatst: rb=test
adatst: rb
<<<<<< RECORD BUFFER >>>>>>

00000000      54455354 ..... TEST.....
adatst: rb:test
adatst: rb
<<<<<< RECORD BUFFER >>>>>>

00000000      74657374 ..... test.....
```

「rb=test」を指定した場合、文字列「test」は大文字に変換されますが、「rb:test」を指定した場合は変換されません。

■ キーワード

キーワードは、あらかじめ定義された文字列です。

■ 数値

数値は、符号なし整数を数の並びとして表現したものです。数値は、「0x」という 2 桁の文字列を付加して 16 進フォーマットで入力することもできます。

■ 数値範囲

数値範囲を指定するときは、範囲になる数値をハイフンでつなぎます。

■ キーワード、文字列、数値、数値範囲のリスト形式表記

キーワード、文字列、または数値をリスト形式で指定する場合は、それぞれの値をコンマで区切り、全体を丸カッコで囲みます。

引数は、次の規約に従って指定します。

■ 最初の引数は、機能キーワードに続く等号 (=) の後に指定します。

■ 後続の引数は、等号と値を伴うキーワードを用いて指定します。

例：

```
adadbm: recover
```

```
adadbm: renumber = (3,14)
```

```
adadbm: reuse = (isn,ds), file = 6
```

```
adarec: regenerate = *  
adarec: plog = 654  
adarec: checkpoint = synp  
adarec: block = 0x1A
```

引数のリストは後続の行に継続することもできます。多機能ユーティリティの場合には、このリストがすべて整い、正しい場合に機能が実行されます。

構文の一覧で使用される記号

引数が必須またはデフォルト値を持つ場合は、ユーティリティ記述の先頭にある引数のリストの中に文字 M または D があります。

現在設定されているパラメータの確認

設定ないしプリセットされた全パラメータの現在の値は、アスタリスク (*) を入力することによって表示されます。

```
adarep:*<cr>
```

```
%ADAREP-I-PARSET setting of COUNT=<set>  
%ADAREP-I-PARSET setting of DBID=5  
%ADAREP-I-PARSET setting of FDT=<set>  
%ADAREP-I-PARSET setting of FILES=(10)
```

スイッチが設定されている場合は、「<set>」と表示されます。

絶対時間

一部のユーティリティ（ADACLPL、ADADBM、ADAREP）の引数は、絶対時間（日付と時刻の両方を含む文字列）の形式で指定する必要があります。使用する文字列は、次の絶対形式と同じでなければなりません。

```
dd-mmm-yyyy[:hh:mm:ss]
```

絶対時間の指定方法の詳細については、該当するユーティリティのドキュメントを参照してください。

単一機能ユーティリティと多機能ユーティリティ

ユーティリティは、1回の実行で実行可能な機能の数に応じて、単一機能ユーティリティと多機能ユーティリティに分けられます。

単一機能ユーティリティでは、各キーワード値の正誤はそれが入力された後でチェックされます。引数の設定が整っているかどうかは、そのユーティリティのプロンプトでCtrlキーとDキーを同時に押したとき（UNIX）、またはCtrlキーとZキーを同時に押してから改行キーを押して（Windows）、ユーティリティが実行したときにチェックされます。単一機能ユーティリティは機能が実行されると終了します。

多機能ユーティリティでは、機能の指定またはパラメータの設定、またはその両方を使用できます。機能やパラメータの指定は一定の順序に従わなければなりません。機能はこの順序が完全で正しいときに実行されます。多機能ユーティリティの機能を実行した場合、正常終了したのかどうかかわるように、終了時にメッセージが出力されます。ユーティリティ自身は、明示的に終了しない限り、機能の実行が終了した後でも終了しません。

ユーティリティの終了

ユーティリティの終了には、制御ステートメント *QUIT* またはその省略形の「Q」を使用します。*QUIT* の同義語として *EXIT* も使用できます。



注意: ユーティリティ制御パラメータは常に大文字に変換されます。したがって、*QUIT* と *EXIT* は小文字でも指定できます。*QUIT* と *EXIT* は、FDT 解析モードでも指定できます。例えば *FIELDS* パラメータを指定した後などです。ただし、*LOWER_CASE_FIELD_NAMES* パラメータを指定している場合は、FDT 解析モードで大文字への変換は行われません。この場合は、*QUIT* または *EXIT* を大文字で指定する必要があります。

単一機能ユーティリティでは、*QUIT* により機能が中止され、ユーティリティが終了します。EOF（UNIX の場合は Ctrl キーと D キーを同時に押す、Windows の場合は Ctrl キーと Z キーを同時に押す）を入力すると、まず機能が実行した後でユーティリティが終了します。

多機能ユーティリティでは、*QUIT* でユーティリティが終了します。機能のパラメータを設定している途中で *QUIT* を入力した場合は、機能が中止され、ユーティリティが終了します。ただし、EOF はユーティリティの終了前に機能の実行が試行されます。必要とされるパラメータがいくつか指定されていない場合、ユーティリティは該当機能が実行される前にエントリの要求を行います。

終了時はすべての場合に、入出力ファイルに対する I/O 件数が示され、ユーティリティ名、日付、時刻、経過時間を記載した情報メッセージが発行されます。

```
%ADAFRM-I-IOCNT,      500  Ios on dataset WORK
%ADAFRM-I-IOCNT,      800  Ios on dataset DATA
%ADAFRM-I-IOCNT,      600  Ios on dataset ASSO
%ADAFRM-I-TERMINATED,  28-JUL-2005 11:39:50, elapsed time: 00:00:51
```

ユーティリティが正常終了すると、最終メッセージ **TERMINATED** が表示され、プログラム終了ステータスにゼロが設定されます。そうでない場合には異常終了メッセージ **ABORTED** が発行され、プログラム終了ステータスにはゼロ以外の値が設定されます。多機能ユーティリティは、入力のリダイレクトなしで対話型に使用される場合、常に正常終了します。

いくつかのケースでは、ユーティリティは **TERMINATED** メッセージを表示し、例外を示すためにゼロ以外の終了ステータスを返します。例えば、エラーが見つかったときのユーティリティ **ADAVFY**、またはレコードが棄却されたときのユーティリティ **ADACMP** などです。

エラー処理

ユーティリティコマンド内のエラーがある部分は、後続行内の上矢印によって示され、その後の行にエラーメッセージが表示されます。左から右に向かって評価され、その過程で検出されたエラーが順番に表示されます。

パラメータ設定のシーケンス内にエラーが発生した場合、エラーの前の設定部分は実行されません。

```
adadbm: refresh = 30
                ^
%ADADBM-E-NOTLOAD, file not loaded
```

入力行内のエラーの位置は、後続行内の上矢印によって示されます。

Adabas シーケンシャルファイル

概要

Adabas ニュークリアスまたはユーティリティは、通常、シーケンシャルファイルの入出力を処理します。これらのシーケンシャルファイルに加えて、所定の内部 Adabas フォーマットの内部ファイル（例えばプロテクションログ）、およびユーザー定義フォーマットの外部ファイル（例えば圧縮解除したデータ）もあります。

内部 Adabas シーケンシャルファイルの出力先は、ファイルシステム、RAW セクション、またはテープデバイスに設定することができます。シーケンシャルファイルの中には、出力先を名前付きパイプに設定できるものもあります。次の表は、内部 Adabas シーケンシャルファイル、RAW デバイスアクセスに対する **ADADEV** キーワード、およびディスクセクション（UNIX のみ）、テープデバイスまたはファイルシステムに設定可能な対応する環境変数をリストしていま

す。環境変数が定義されていない場合、Adabas は、現在のワーキングディレクトリのファイルを環境変数名付きで使用します。

シーケンシャルファイルタイプ	ADADEV キーワード	環境変数
プロテクションログ	PLG	DEVPLG _x 、NUCPLG _x (*)、PLPLEX、PLPPLG _x 、RECPLG _x
コマンドログ	CLG	CLPCLG _x 、DEVCLG _x 、NUCCLG _x
バックアップファイル	BCK	BCK00 _n _x 、DEV00 _n _x 、ULD00 _n _x
バックアップファイルコピー	BCKOUT	BCKOUT _x 、DEVOUT _x
圧縮データ	DTA	CMPDTA _x 、DCUDTA _x 、DEVDTA _x 、MUPDTA _x 、ULDDTA _x
ディスクリプタバリュートーブル	DVT	CMPDVT _x 、DEV DVT _x 、MUPDVT _x 、ULDDVT _x
ADAMUP テンポラリデータ	MUPTMP	MUPTMP _x
ADAMUP ログ (DTA/DVT)	MUPLOG	DEVLOG _x 、MUPLOG _x
リオーダファイル	ORDEXP	DEVEXP _x 、ORDEXP _x
ユーティリティエラーファイル	ERR	ERRIN _x 、CMPERR _x 、DCUERR _x 、DEVERR _x 、INVERR _x 、MUPERR _x 、RECERR _x

ただし、n = 1～9、x は省略または 2 以上
 (*) NUCPLG はテープ上には存在しません。

次の Adabas シーケンシャルファイルがファイルシステム内に存在する必要があります。

シーケンシャルファイルタイプ	環境変数
ADAMUP ISN リスト	MUPISN
LOB 処理用の ADAMUP 一時ワークスペース	MUPLOB、MUPLBI
フィールド定義データ	FDUFDT、CMPFDT、DCUFDT
圧縮解除した未加工データ	CMPIN、DCUOUT

ディスクスペース管理ユーティリティ ADADEV を使用して、RAW ディスクセクションのファイルを管理できます。詳細については、「ADADEV」を参照してください (UNIX のみ)。

プラットフォームによる依存性

Adabas シーケンシャルファイルは、上位優先と下位優先のプラットフォームでは内部フォーマットが異なりますが、バイト順が同じプラットフォームであれば、互換性が保証されます。

例

プラットフォーム 1	バイト順	プラットフォーム 2	バイト順	互換性あり
Solaris	上位優先	AIX	上位優先	○
LINUX (Intel)	下位優先	Windows	下位優先	○
Solaris	上位優先	Windows	下位優先	×

次のファイルのみがすべてのプラットフォームで互換性を持ちます。

- 非圧縮データファイル
- FDT ファイルなどの純粋なテキストファイル

上位優先と下位優先のプラットフォーム間でデータを交換する場合、次の手順に従います。

1. ADAULD を使用してアンロードします
2. ADADCUC を使用して圧縮解除します
3. ADACMP を使用して圧縮します
4. ADAMUP を使用してロードします

非圧縮レコードが大きすぎるため、この手順が不可能な場合、ソースデータベースからデータを読み取るプログラムを作成する必要があります。このプログラムを作成する際に、ファイルシステム内の中間ファイルにデータを書き込む方法があります。別のプログラムが、この中間ファイルを読み込んで、ターゲットデータベースにデータを書き込みます。また、Entire Net-Work を使用してデータを直接データベースに格納する方法もあります。

名前付きパイプの使用

名前付きパイプは、ユーティリティの出力を別のユーティリティやアプリケーションに処理させるとき、または、ユーティリティ入力が他のユーティリティやアプリケーションの出力だった場合、ストレージスペースを節約するために使用することができます。名前付きパイプは、MUPDTA、MUPDVT、MUPISN、MUPLBI、MUPLOB、MUPLOG、MUPTMP、NUCPLG、ORDEXP、PLPLEX、PLPPLG、ULD00n 以外のすべての Adabas シーケンシャルファイルに使用することができます。

Windows で動作する名前付きパイプは、一時的なオブジェクトです。環境変数に名前付きパイプの名前が指定されている場合（例えば \\.\pipe\パイプ名）、名前付きパイプに書き込みを試行するユーティリティは暗黙的に名前付きパイプを作成します。それから、このユーティリティは、ユーティリティの読み込み処理が発生するまで、前もって定められた一定時間を待ってから、

名前付きパイプから読み込みを開始します。読み込みユーティリティの処理無しで時間が経過した場合、書き込みユーティリティは I/O エラーで中断します。

Adabas シーケンシャルファイルの複数エクステンツ

シーケンシャルファイルに割り当てられたスペースが、ユーティリティがそれを作成している間に使い切ってしまった場合、Adabas は、別の空きスロットにファイルを継続しようとします。これは、割り当てたスペースまたは事前割り当てが行われたスペースがすべて使われた場合、またはシーケンシャルファイルが設置されたファイルシステムがいっぱいになった場合に、RAW デバイスに設置されたすべてのシーケンシャルファイルに対して適用されます。次のように、環境変数の名前にカウンタを付加すれば、環境変数を何回も使用することができます。

例 (C シェル)

```
setenv NUCPLG /dev/rdsk/c4d0s2
setenv NUCPLG2 /usr/adabas/PL0G_2
setenv NUCPLG3 /dev/rdsk/c7d0s2
```

上記の3つのデバイスは、ニュークリアセッション中にプロテクションログに使用されます。

次に示すように、1つの環境変数内でファイル/デバイス名のすべてを設定する方法もあります。



注意: デリミタは、UNIX プラットフォームでは空白で、Windows プラットフォームではセミコロン (「;」) です。

例 (C シェル)

```
setenv NUCPLG "/dev/rdsk/c4d0s2 /usr/adabas/PL0G_2 /dev/rdsk/c7d0s2"
```

Adabas の起動時に、最初の環境変数（この場合は NUCPLG）から処理されますが、環境変数に値が設定されていなければ、そのデフォルト（デフォルトは環境変数と同じ名前のカレントディレクトリのファイル）が処理されます。ファイルの末尾に達したとき、またはファイルシステムがいっぱいになったとき（ファイルシステムが使用された場合）、ファイルはクローズされ、次の環境変数（NUCLPG2）が解釈され、関連ファイルやディスクセクションがオープンされます。その後、次のエクステンツのために使用されるディスクセクションから空きスペースが割り当てられます。後続の環境変数（例えば NUCPLG4）が存在しない場合、Adabas はもう一度最初から処理し、最初のディスクセクション（NUCPLG）に戻ります。

1つの環境変数だけが定義されており、ディスクセクションに割り当てられている場合、後続のファイルエクステンツはすべて同じディスクセクションに作成されます。1つの環境変数だけが定義されてファイルシステムに割り当てられている場合、ファイルシステムがいっぱいになると、Adabas は継続することができません。

1つのファイルエクステンツから別のファイルエクステンツへの切り換えが発生した場合、クローズされたエクステンツは、通常その名前にエクステンツ番号が付加され名前が変わります。例えば、ディスクセクションの PLG.3 というファイルエクステンツは PLG.3(1) という名前に変

更されます。また、ファイルシステムの/tmp/BCK_DB というファイルエクステントは /tmp/BCK_DB(1) という名前に変更されます。新規ファイルエクステントの名前には新規ファイルエクステント番号が含まれます。したがって、ディスクセクションの新規ファイルエクステントは PLG.3(2) と命名され、ファイルシステムの新規ファイルエクステントは /tmp/BCK_DB(2) と命名されます。



注意: 環境変数の値のサイズは、オペレーティングシステムに応じて 512 バイトまたはそれ以上に制限されます。1つの環境変数で多数のエクステントを指定すると、環境変数の値が切り捨てられて、エラーが発生する場合があります。このような場合には、エクステントを個別の環境変数で指定する必要があります。

ユーティリティの呼び出し

複数のエクステントを持つファイルを指定してユーティリティを使用する場合には、重要な制限事項が1つ適用されます。その制限事項とは、複数のファイルエクステントが必要になり、1つの環境変数に1つの値だけが設定されている場合に、入力のリダイレクトができなくなることです。この場合、ユーティリティは次のメッセージを表示して異常終了します。

```
%ADABCK-E-SFN0TP0, subsequent file extents not possible with input redirection
```

次のように入力すると、それぞれのモードが正常に動作します。

対話モード (UNIX)

```
adabck <RETURN>
dbid=5, dump=4 <RETURN>
<CTRL/D>
```

対話モード (Windows)

```
adabck <RETURN>
dbid=5, dump=4 <RETURN>
<CTRL/Z><RETURN>
```

対話モード (コマンド行)

```
adabck dbid=5 dump=4
```

バッチジョブ

```
adabck dbid=5 dump=4 &
```

新しいテープが必要になると、ユーティリティの処理は停止します。停止したら、ジョブをフォアグラウンドに切り替えれば、有効なデバイス名を入力して、処理を続行できるようになります。

複数エクステントファイルの読み込み

複数エクステントに分割されたシーケンシャルファイルを読み込むとき、正しい環境変数が設定されている場合、Adabas は、それぞれのエクステントが終了しているかどうかチェックし、読み込みを継続する次のエクステントのファイル／デバイスがどこにあるのかパス名を探します。

例：

ADAREC は、2 セクションに分割されたプロテクションログを 1 つにまとめます。環境変数 RECPLG はプロテクションログ 5 (PLG.5(1)) の最初のエクステントをポイントする必要があります。その処理が終了すると、ADAREC は次のエクステントのファイル／デバイスのパス名を求めます。

```
Enter next file/device path name to continue PLG.5(2)
```

ファイルエクステントはディスクセクションまたはファイルシステムまたはテープデバイスに配置できます。後続のエクステントをオープンした後、Adabas は、指定エクステントが予期されたものであるかを確認するためのチェックを行います。これが正しいエクステントではない場合、エラーメッセージが表示され、正しいファイル／デバイス名に対するプロンプトが再表示されます。このようなプロンプトが表示された場合は、「quit」を入力して他の処理を中止できます。Adabas はプロテクションログエクステントを予期していますが、後続のプロテクションログエクステントが存在しない状況が発生することがあります。この場合は、「close」と入力して、エラーなしで現在のプロテクションログファイルの処理を終了できます。

counted 環境変数を使用する場合、プロンプトなしで、1 回の実行でファイルを処理することもできます。この場合、ファイルを作成したときと同じ方式が使用されます。ファイル（複数可）は RAW デバイス、ファイルシステムまたはテープデバイスに配置することができます。

例

C シェルの場合は、次のようになります。

```
setenv RECPLG /dev/rmt/0m
setenv RECPLG2 /dev/rdsk/c10d0s2
setenv RECPLG3 /dev/rdsk/c7d0s2
```

Bourne シェル、Korn シェルの場合は、次のようになります。

```
RECPLG = /dev/rmt/0m
export RECPLG
RECPLG2 = /dev/rdsk/c10d0s2
export RECPLG2
RECPLG3 = /dev/rdsk/c7d0s2
export RECPLG3
```

Windows：

```
set RECPLG=\\.\TAPE0
set RECPLG2=C:\BCK_2
set RECPLG3=D:\BCK_3
```

例

1つの環境変数内で、すべてのファイル／デバイス名を設定する方法としては、次の例のような方法もあります。

一括更新ユーティリティ ADAMUP が 2 エクス Tent に分割されたデータファイル (DTA) を使用する場合、次のように設定することができます。

C シェルの場合は、次のようになります。

```
setenv MUPDTA "/dev/rdisk/c5d0s2 /usr/adabas/DTA_2"
```

Bourne シェル、Korn シェルの場合は、次のようになります。

```
MUPDTA = "/dev/rdisk/c5d0s2 /usr/adabas/DTA_2"export MUPDTA
```

Windows :

```
set MUPDTA=\\.\TAPE0;C:\adabas\DTA_2
```

テープデバイスのサポート

テープの使用

Adabas ユーティリティはテープデバイスを直接読み書きすることができます。この機能を使用できるようにするために、対応する環境変数はテープデバイス名をポイント先として設定しなければなりません。ユーティリティはそれ自身のテープフォーマット (他のテープ操作ユーティリティの互換性がない) を使用します。

ユーティリティを開始する前に、テープデバイスの名前がポイント先になるように環境変数を設定する必要があります。例えば、バックアップユーティリティ ADABCK を HP-UX マシンで使用する場合、次のように入力すべきです。

C シェルの場合は、次のようになります。

```
setenv BCK001 /dev/rmt/0m
```

Bourne シェル、Korn シェルの場合は、次のようになります。

```
BCK001 = /dev/rmt/0m
export BCK001
```

例えば、テープデバイス \\.\TAPE0 に書き込むために、Windows でバックアップユーティリティ ADABCK を使用したい場合、次を入力する必要があります。

Windows :

```
set BCK001=\\.\TAPE0
```

テープドライブのデバイス名は \\.\TAPE<tape id> 形式を持っています (<tape id>は 0 から始まるテープデバイス番号です)。

複数テープサポート

非常に大規模なファイルは、何本かのテープに分けることが必要になる場合もあります。テープがいっぱいになった場合、ユーティリティはユーザーの交換が必要な旨のメッセージを表示し、入力を求めます。テープを交換した後、ユーザーは継続するために再度デバイス名を入力する必要があります。

例：

```
%ADABCK-I-TDEVFU, end of tape reached, rewinding
      Insert new tape and enter device name, [/dev/rmt/0m] (UNIX)
      Insert new tape and enter device name, [\\.\TAPE0] (Windows)
```

複数テープを読み込むとき、同様の方式が使用されます。ユーティリティは、テープが正しい順番でマウントされているかをチェックします。テープが間違った順番で挿入されている場合、ユーティリティはエラーメッセージを発行し、入力を促します。

例：

```
%ADABCK-F-ILLSUB, illegal subsequent tape detected,
      present 1, expected 3
```

継続するため、ユーザーは正しいテープをマウントし、デバイス名を入力しなければなりません (テープデバイス名は最初と同じであってはいけません)。標準ファイル名および RAW セクションは入力できません。

自動テープ交換

プロンプトなしの自動テープ交換をサポートするため、1つの環境変数に複数のテープデバイスを設定することができます。これは、例えば最初に選択されたテープの空きスペースがなくなった場合に、Adabas は自動的に次のテープに切り替えることを意味します。このメカニズムは、テープの読み込みと書き込みの両方でサポートされています。

例 (C シェル)：

```
setenv BCK001 "/dev/rmt/0m /dev/rmt/1m"
setenv RECPLG "/dev/rmt/0m /dev/rmt/1m"
```

または

```
setenv BCK001 /dev/rmt/0m
setenv BCK0012 /dev/rmt/1m
setenv RECPLG /dev/rmt/0m
setenv RECPLG2 /dev/rmt/1m
```

例（Windows）：

```
set BCK001=\\.\TAPE0;\\.\TAPE1
set RECPLG=\\.\TAPE0;\\.\TAPE1
```

大規模データベースを選択された2つのテープにダンプします。2つのテープでは十分でない場合、Adabas は3番目のテープに交換するよう促します。

ユーティリティの呼び出し

テープ処理に伴うユーティリティの使用には重要な制限事項が1つ適用されます。その制限事項とは、複数テープが必要になっても、自動テープ交換機能（上記参照）を使用していない場合には、入力のリダイレクトができなくなることです。この場合、ユーティリティは次のメッセージを出力して異常終了します。

```
%ADABCK-E-STNOTPO, subsequent tapes not possible with input redirection
```

次のように入力すると、それぞれのモードが正常に動作します。

対話モード（UNIX）

```
adabck <RETURN>
dbid=5, dump=4 <RETURN>
<CTRL/D>
```

対話モード（Windows）

```
adabck <RETURN>
dbid=5, dump=4 <RETURN>
<CTRL/Z><RETURN>
```

対話モード（コマンド行）

```
adabck dbid=5 dump=4
```

バッチジョブ（UNIX）

```
adabck dbid=5 dump=4 &
```

新しいテープが必要になると、ユーティリティの処理は停止します。停止したら、ジョブをフォアグラウンドに切り替えれば、有効なデバイス名を入力して、処理を続行できるようになります。

単一テープの複数ファイル

Adabas では、1 本のテープに保存された複数ファイルを扱うことができますが、これはお使いのオペレーティングシステムでサポートされている場合に限り（例えば、巻き戻しなしテープを備えた mt インターフェイスなど）。この機能を使用するため、テープでの必要な位置までスキップし、その後、ユーティリティを実行します。

例：

```
mt fsf 1
adabck db=5 dump=3
```

テープの最初のファイルをスキップし、データベースを2番目のファイルとしてテープにダンプします。

サポートされないテープフォーマット

Adabas がサポートしていないテープフォーマットのテープに、外部データ（非圧縮データなど）が入っていることがあります。この場合には、Adabas から I/O エラーが返されます。このようなテープは、UNIX コマンド dd を使用すれば、ファイルを名前付きパイプに読み込むことができます。その後、Adabas ユーティリティは名前付きパイプから読み込むことができます。

ADAMUP および ADAINV の実行の最適化

インデックスを作成するには、ユーティリティ ADAINV と ADAMUP でディスクリプタ値をソートする必要があります。ソート実行の効率は、ユーティリティパラメータ LWP のサイズの影響を受ける可能性があります。このパラメータは、ユーティリティに割り当てられた最小メモリに加えて使用できるメモリの量を指定しています。以下では、ソート処理の実行に対する LWP パラメータのサイズの影響について説明します。

- LWP パラメータのサイズに応じて、メモリ内の領域がメモリ内ソート用に提供されます。LWP の大きさが十分な場合は、すべてのディスクリプタ値をメモリ内でソートでき、インデックスを作成できます。また、SORT コンテナは必要ありません。ADAMUP および ADAINV SUMMARY 機能で、メモリ内ソートに必要な値が表示されます。
- 領域が小さすぎてメモリ内でソートできない場合は、ソート対象のデータが、メモリ内でソートできるサブセットに分割されます。サブセットの数は、LWP パラメータのサイズしだいです。LWP のサイズを大きくすると、サブセットの数を減らすことができます。これらのサブセットをメモリ内でソートすると、ソートされたサブセットが SORT コンテナに書き込まれます。
- すべてのサブセットをソートしたら、ソートしたサブセットをマージする必要があります。ただし、1 回のパスでは限られた数のサブセットしかマージできません。サブセットの数が制限値を超えない場合は、マージ後にすべてのディスクリプタ値がソートされ、インデックスが作

成されます。単一パスのマージに必要なLWP値とSORTコンテナの必要サイズは、ADAMUPおよびADAINV SUMMARY機能で表示されます。

- サブセットの数が多い場合、マージされたサブセットは再度ディスクに書き込まれ、別のマージパスが必要になります。つまり、1回のマージパスで十分な場合のSORTコンテナサイズと比較して、2倍のサイズが必要になります。ADAMUPおよびADAINV SUMMARY機能で、SORTコンテナの必要サイズが表示されます。
- SORTコンテナの必要サイズは、マージパスの回数にのみ依存し、LWPパラメータの値には依存しません。LWPパラメータが単一パスのマージに十分な大きさの場合、必要となるディスクスペースは、2回以上のパスでマージする際の半分になります。
- メモリ内ソートは、1回以上の追加のマージパスを使用するソートよりも必ずしも高速であるとは限りません。これは、提供されているメモリに、ソートアルゴリズムでランダムにアクセスすることが理由です。LWP=0の場合は、通常、メモリキャッシュミスは多くなりませんが、LWPパラメータが大きくなると、メモリキャッシュミスが多くなります。そのため、ソートされたサブセットのマージに必要な時間よりも、ソートに必要な時間が長くなる場合があります。ソートされたサブセットをマージする際には、多くのメモリキャッシュミスは発生しません。また、SORTコンテナのI/Oへの悪影響も限定的です。これは、I/Oが非同期的に実行されているためです。また、データがSORTコンテナのファイルシステムキャッシュに残っていることもあります。
- システムコンポーネントのパフォーマンスは、コンピュータによって大きく異なる場合があります。したがって、定期的にADAMUPやADAINVを実行する場合は、異なる方法を試してみることをお勧めします。これらのユーティリティを頻繁に実行しない場合は、上記の作業は労力に見合う価値がない可能性があります。
- ADAMUP SUMMARYの実行は非常に高速です。これは、MUPDTAおよびMUPDVTファイルが作成されたときに、すでにディスクリプタサマリが作成されていて、MUPDTAファイルに保存されているからです。これに対して、ADAINV SUMMARYは、ディスクリプタサマリを作成するために、ファイルのすべてのDATAブロックを読み取る必要があるので、比較的低速になります。
- SUMMARY機能は、推奨されるTEMPサイズも計算します。ユーティリティは一度に1つのディスクリプタしか処理できませんが、常にすべてのディスクリプタの値を確認します。ADAINVがDATAブロックを読み直すのを回避するために、他のディスクリプタの値は、ディスクリプタごとにグループ化されてTEMPに格納されます。TEMPは、最大のディスクリプタを格納できるようにする必要があります。すべてのディスクリプタを1回のパスで処理できる十分な大きさのTEMPが理想的です。ADAINVは、ディスクリプタサイズを事前に認識しないので、1回のパスに必要なTEMPサイズは比較的大きくなります。ADAINV SUMMARYは、2回のパスでディスクリプタを処理するためのTEMPサイズ也表示します。
- SUMMARY機能によって計算された値は正確な最小値ではありません。この値は切り上げられています。つまり、これらの値は十分ですが、より小さい値でも目的にかなう場合があります。

ニュークリアスとユーティリティの間の同期

ユーティリティコミュニケーションブロック (UCB)

ほとんどのユーティリティはデータベースコンテナに書き込むことができます。また、オンラインモード（つまりニュークリアスがアクティブであるとき）とオフラインモード（つまりニュークリアスがアクティブでないとき）の両方で実行できます。つまり、データベースの整合性を保証するには、ニュークリアスがアクティブかどうかに関係なく、ニュークリアスとユーティリティの同期メカニズムが必要です。

その目的で、各 Adabas データベースに、ユーティリティコミュニケーションブロック (UCB) が含まれています。UCB は、ASSO コンテナの RABN 3 に保存されています。1 つ以上のファイルに対する読み取りまたは書き込みアクセス権がユーティリティで必要になるたびに、以下の情報を含む UCB エントリを最初に作成する必要があります。

- 処理する Adabas ファイルのファイル番号。
- アクセスモード（読み取りまたは書き込み）。

さらに、データベースコンテナ内の一部のグローバルデータ構造へのアクセスは、ニュークリアスとユーティリティの間で同期される必要があります。

オンラインモードでは、ユーティリティとニュークリアスの同期は、常にニュークリアスを介して実行されます。ユーティリティは特別な Adabas コマンド (U コール) を発行し、UCB とその他のグローバルデータベースブロックを更新します。

オフラインモードでは、ユーティリティ自体によって UCB およびその他のグローバルデータベースブロックが更新されます。2 つのユーティリティプロセスがブロックを同時に更新する状況を回避するために、セマフォを介してアクセスが保護されます。セマフォでは、グローバルデータ構造へのアクセスのみが保護され、ファイル固有データへのアクセスは UCB を介して保護されます。

ユーティリティが終了すると、通常、UCB のエントリは再び削除されます。ただし、ユーティリティが異常終了した場合は、ユーティリティの UCB エントリが削除されない可能性があります。この場合は、ADADBMRESET=UCB を使用して、UCB を手動で削除する必要があります。



注意: ニュークリアスおよび他のユーティリティ (ADAREP および ADAPRI) と同期しない、読み取り専用ユーティリティがいくつかあります。このようなユーティリティには、いつでも実行できるという利点がありますが、まれにユーティリティが異常終了するというリスクがあります。

読み取り専用モードのデータベース

コンテナが書き込み保護されている場合、またはニュークリアスパラメータ `OPTION=READONLY` を指定した場合は、データベースを読み取り専用モードになります。読み取り専用モードのデータベースの場合は、UCB を書き込みませんが、それでも同期が必要です。これは、読み取り専用モードが、ニュークリアスパラメータ `OPTION=READONLY` のみによって生じていることがあり、その際にはコンテナが書き込み可能なことが理由です。ニュークリアスおよびユーティリティは、ジェネラルコントロールブロック (GCB) が保存されている共有メモリにアクセスします。これは、最初のニュークリアスまたはユーティリティプロセスが開始したときに作成される共有メモリです。最後に共有メモリを使用したニュークリアスおよびユーティリティプロセスが終了すると、再び削除されます。この共有メモリは、データベースを読み取り専用としてマークするために使用されます。これは、共有メモリが作成され、コンテナが読み取り専用の場合、またはニュークリアスが `OPTION=READONLY` で開始された場合に行われます。

次の規則が適用されます。

- 共有メモリが読み取り専用としてマークされている場合は、ニュークリアスを読み取り専用モードで起動すること、またはデータベースコンテナの読み取りのみを行うユーティリティ機能を起動することのみが可能です。書き込みを実行するニュークリアスやユーティリティは、共有メモリを削除した後にのみ開始できます。
- 共有メモリが読み取り専用としてマークされていない場合は、ニュークリアスを読み取り専用モードで起動することはできません。これは、共有メモリを作成したユーティリティがデータベースを変更する可能性があることが理由です。読み取り専用ニュークリアスは、この変更が行われないことを前提としています。

8 データのロードとアンロード

■ はじめに	130
■ 他のハードウェアアーキテクチャへのデータのコピー	131
■ 非圧縮データフォーマット	132
■ ADACMP の入力データ要件	136
■ ADACMP の処理に関する考慮	143
■ ADAMUP の処理に関する考慮	144
■ ADABCK の処理に関する考慮	148
■ ADAORD の処理に関する考慮	151
■ ファイルのスペースの見積り	154

はじめに

データベースにデータをロードするにはいくつかの方法があります。

- ADAMUP ユーティリティを使用：ADACMP、ADAMUP または ADAULD によって生成された圧縮データをロードできます。
- ADABCK ユーティリティを使用：ADABCK によって作成されたバックアップからファイルまたはデータベースをリストアできます。
- ADAORD ユーティリティの IMPORT 機能を使用：ADAORD の EXPORT 機能によって作成されたファイルをインポートできます。

圧縮データ

ADAMUP ユーティリティでデータベースにロードされるデータは、特別に圧縮したデータフォーマットで ADAMUP への入力にする必要があります。圧縮データはユーティリティ ADACMP（圧縮解除したデータを「非圧縮データフォーマット」セクションの説明のように、必要なフォーマットに変換する）で生成することができます。既存データをデータベースからアンロードするとき、ユーティリティ ADAMUP および ADAULD で圧縮データを生成することもできます。

データフォーマットとして柔軟性に富んだフォーマットです。データファイルを圧縮してから、そのファイルを必要に応じて別々のデータベースにロードすることも、1つのデータベースにロードすることもできます。また、レコードのサブセットだけをファイルにロードすることもできます。

このファイルのフォーマットの不便な点は、データをデータベースにロードするときに、Adabas が各ファイルにソートインデックスを構築する必要があるということです。ファイル数が多い場合、これは、大量の CPU タイムを必要とし、SORT および TEMP コンテナファイルを必要とします。

バックアップデータ

バックアップデータはユーティリティ ADABCK で生成します。このようなデータを使用して長期間データアーカイブを作成することができます。また、ファイルをデータベースにリストアしたり、データベース全体をリストアしたりするために使用することもできます。

バックアップおよびリストア処理は、データの保存やリストアの他の方法より高速に行われます。ただし、元のデータベースにバックアップファイルを常にコピーしなければなりません。また、異なるファイル番号でファイルをコピーすることもできません。

データのインポート／エクスポート

ADAORD EXPORT 機能を使用してデータベースからエクスポートされたデータは、ADAORD IMPORT 機能を使用してデータベースにインポートすることができます。エクスポートされたデータは前述の圧縮データとフォーマット上、非常に類似していますが、おもな違いは、エクスポートされたファイルのインデックス情報もエクスポートされることです。これは、データを続けてインポートしても、インデックスの再構築が不要なので、ADAMUPに相当する別の処理よりも、ロード手順が非常に高速であることを意味します。また、SORT および TEMP コンテナファイルは必要ありません。

圧縮データと同様に、これは柔軟なデータフォーマットです。データファイルを圧縮してから、そのファイルを必要に応じて別々のデータベースにロードすることも、1つのデータベースにロードすることもできます。ただし、インデックス情報はエクスポートファイルに格納されるので、レコードのサブセットだけをデータベースのファイルにインポートすることはできません。

他のハードウェアアーキテクチャへのデータのコピー

ある Adabas データベースから、異なるハードウェアアーキテクチャを使用しているコンピュータ上の別のデータベースに、データをコピーすることが必要になる場合があります。たとえば、UNIX プラットフォームから Windows プラットフォームにコピーする場合などです。

この目的のために、ユーティリティ ADABCK（バージョン 6.4 以降）を使用できます。これを使用すれば、あるハードウェアアーキテクチャ上で作成したバックアップを、別のアーキテクチャを使用しているコンピュータ上のデータベースに復元できます。



注意:

1. この方法は、バージョン 6.3 より前の Adabas で作成されたバックアップでは実行できません。
2. この方法を使用した、メインフレームとの間でのデータコピーはサポートされていません。
3. 異なるハードウェアアーキテクチャで作成された入力ファイルの処理は、ADABCKでのみ可能です。ユーティリティ ADAMUP と ADAORD では処理できません。
4. 以前の Adabas バージョンで必要とされていた古い方法を使用して、別のハードウェアアーキテクチャにデータをコピーすることもできます。その方法とは、ユーティリティ ADAULD を使用してファイルをアンロード、ユーティリティ ADADCU を使用してファイルを圧縮解除、ユーティリティ ADACMP を使用して新しいアーキテクチャ用にファイルを圧縮、ADAMUP を使用してファイルをロード、という手順を使用する方法です。これは、ブロックサイズやファイルの FDT を変更する場合にも便利です。

非圧縮データフォーマット

このセクションでは、ユーティリティ ADACMP への入力であるデータレコードと、ユーティリティ ADADCU からの出力であるデータレコードのフォーマットについて説明します。このフォーマットは非圧縮データフォーマットといいます（未加工データフォーマットともいう）。ユーティリティ ADACMP は、このフォーマットでデータを読み取り、後続の一括更新ユーティリティ ADAMUP への入力のためにそれを圧縮します。ユーティリティ ADADCU は反対の処理を行います。これは、ADACMP が生成した圧縮データを圧縮解除します。圧縮データは、データを削除またはデータベースからアンロードするとき、ユーティリティ ADAMUP および ADAULD によって生成できることに注意してください。

特に指示がない限り、記述されたデータフォーマットは ADACMP に対する入力データおよび ADADCU に対する出力データの両方に適用します。

非圧縮データレコードの構文

非圧縮データレコードは、次の構文要素の順で構成されています。

```
format_buffer_element  
field_references
```

`field_references`を除く構文要素は、『コマンドリファレンス』の「Adabasの呼び出し」のフォーマットバッファとレコードバッファで説明されているフォーマットバッファ要素と同じです。ここでは、`nX`、リテラル、または長さおよびフォーマット指定を含むフィールド定義が存在する場合は、フォーマットバッファ要素はこれらの要素になります。相違点は、次の構文要素の区切り方法です。

- 1つの構文要素は、1行に入力されなければなりません。
- 構文要素は、コンマまたは改行によって区切られます。
- 構文要素間にコメントを挿入できます。セミコロンは、後続の文字が改行されるまではそれがコメントであることを示します。
- 構文要素間に *FDT* を挿入できます。*FDT* は新規行で入力する必要があります。これにより、非圧縮データレコード構文が指定されたユーティリティは、*FDT* を表示するように指示されます。

次の特別な考慮事項は、非圧縮レコード構造指定で指定されたフォーマットバッファ要素に適用されます。

- 編集マスクは使用できません。
- N 要素は使用できません。
- 1-N 要素の同一行内の直前には、対応する C 要素を付加する必要があります。更新または格納コマンドのフォーマットバッファとは異なり、これらの要素は圧縮用のフォーマットバッファでも使用できます。

例

GB は、BA、BB および BC フィールドからなるピリオディックグループと仮定します。

GBC,GB1-N	ピリオディックグループのオカレンスの回数分だけ、ピリオディックグループのオカレンスが繰り返されて、GBC、BA1、BB1、BC1、BA2、BB2、BC2 というように展開されます。
-----------	--

- 1-N 要素は、PE 内のフィールドでは使用できません。

無効な指定の例

GB2-GB4	構文に誤りがあります。
GB4-2	範囲の下限の方が上限よりも大きくなっています。
GBC GB1-N	GBC と GB1-N は同じ行に指定する必要があります。
GBC,BA1-N	ピリオディックグループにフィールドを指定する場合は、1-N を指定することはできません。

field_references の構文

```
name R [mu_pe_index] [, length ]
```

name は LOB フィールドの名前です。非圧縮レコード指定にフィールド参照が含まれている場合、非圧縮レコードには LOB 値自身は含まれませんが、LOB 値が含まれているファイルの名前が含まれます。

length には、0 以上の数値または '*' を指定できます。length=* は、単一の LOB 値のレコードの終わりでのみ許可されます。length=0 が指定されると、2 バイトの包括長さバイトがファイル名の先頭に付加されます（LA フィールドと同様）。デフォルトは 0 です。

mu_pe_index の構文

```
{ i [-j] | N } [ ( m [-n] | N | 1-N ) ] | 1 - N
```

MU フィールドまたはピリオディックグループ内のフィールドが LOB フィールドである場合、フィールド値を指定するときの方法と同様に、フィールド参照に MU または PE インデックスを指定できます。



注意:

1. フィールド値の MU-PE インデックスとしての 1-N の使用に適用される上述の規則は、フィールド参照にも適用されます。
2. LOB フィールドの値が空白の場合、フィールド参照も同様に空白です。

非圧縮レコードのレコード構造の指定例

```
AA,AB      ; 2 fields specified in the same line
9X,'LITERAL' ; Compress: 16 bytes are ignored
            ; Decompress: "      LITERAL" in decompressed record
fdt        ; Display FDT before next field is specified
AT1-12C    ; Number of values of MU field in the first 12 elements of PE
            ; Only allowed for decompress
AT1-12(1-2),8,U ; Values of MU field in a PE
            ; Length is 8 bytes, Format is U
P1C,P11-N  ; Periodic group count and all groups
            ; Allowed for compress, too.
LMR1-4,20   ; File names of files containing values
```

レコード定義例

このセクションでは、レコード定義の例を示します。ここで記述する例はすべて、『Adabas コマンドリファレンスマニュアル』の「付録 A」のサンプル Adabas ファイルを転載したものです。

例 1：エレメンタリフィールドの定義（標準長および標準フォーマット）

```
Syntax      : AA,5X,AB.
Record      : AA value(8 bytes alphanumeric)
              5 spaces
              AB value(2 bytes packed)
```

例 2：エレメンタリフィールドの定義（長さとフォーマットを上書き）

```
Syntax      : AA,5X,AB,3,U.
Record      : AA value (8 bytes alphanumeric)
              5 spaces
              AB value (3 bytes unpacked)
```

例 3：ピリオディックグループの処理

```
Syntax      : GB1.
Record      : BA1 value (1 byte binary)
              BB1 value (5 bytes packed)
              BC1 value (10 bytes alphanumeric)
```

例 4：ピリオディックグループ **GB** の最初から 2 つのオカレンスを処理

Syntax : GB1-2.

Record : BA1 value (1 byte binary)
BB1 value (5 bytes packed) GB1
BC1 value (10 bytes alphanumeric)

BA2 value (1 byte binary)
BB2 value (5 bytes packed) GB2
BC2 value (10 bytes alphanumeric)

例 5：マルチプルバリューフィールド **MF** の 6 番目の値を処理

Syntax : MF6.

Record : MF value 6 (3 bytes alphanumeric)

例 6：マルチプルバリューフィールド **MF** の最初から 2 つの値を処理

Syntax : MF01-02.

Record : MF value 1 (3 bytes alphanumeric)
MF value 2 (3 bytes alphanumeric)

例7: ピリオディックグループ **GC** の最高オカレンス番号と、マルチプルバリューフィールド **MF** に存在する値の数を処理

Syntax : GCC,MFC.
Record : Highest occurrence count for GC (1 byte binary)
Value count for MF (1 byte binary)

出力レコード

ユーティリティ ADADCU は、要求されたフィールド値をレコード定義構文で指定された順に返します。長さまたはフォーマットのいずれか、または両方が指定されていない場合、値はフィールドに定義された標準長とフォーマットで返されます。値が空値である場合は、フィールドのフォーマットに応じて次のように返されます。

フォーマット	空値
英数字 (A)	空白 (ASCII の場合は 16 進 '20'、EBCDIC の場合 16 進 '40')
2 進数 (B)	バイナリのゼロ
固定小数点 (F)	バイナリのゼロ
浮動小数点 (G)	バイナリのゼロ
パック 10 進数 (P)	パック 10 進数の 0
アンパック 10 進数 (U)	アンパック 10 進数の 0 (ターゲットのアーキテクチャにより異なる)
Unicode (W)	WCHARSET に指定した空白



注意: パック 10 進数では、C は符号として使用されます。アンパック 10 進数では、ターゲットアーキテクチャが ASCII の場合は 3 が、ターゲットアーキテクチャが EBCDIC の場合は F が符号として使用されます。

Adabas は、要求されたすべてのフィールドの合計長（標準長または指定された長さ）に等しいバイト数を返します。

ADACMP の入力データ要件

ユーザーデータが ADACMP への入力になる場合、そのデータはシーケンシャルファイルのデータであることが前提条件になります。各レコードのフィールドの構造は、設定済みのフィールド定義テーブル、またはファイルのフィールドのサブセットのどちらかに合わせます。

ユーザーデータが ADACMP への入力になる場合、そのデータはシーケンシャルファイルのデータであることが前提条件になります。4つの方法で入力ファイルのレコードを区切ることができます。詳細については、『ユーティリティマニュアル』の「ADACMP」のパラメータ RECORD_STRUCTURE を参照してください。各レコードのフィールドは、設定済みのデータ定義ステートメントに従って構成する必要があります。

ユーザー出口ルーチンを使用する場合は、ユーザー出口処理後に、構造とデータ定義との間に不整合が発生しないようにする必要があります。対応するデータ定義ステートメントがない入力レコードがあると、それ以降の情報は処理されなくなり、ADACMPの出力結果からも除外されます。

UNPACKED と定義されたフィールドは、下位バイトの上位 4 ビットに有効な符号値を持っていない必要があります。符号はゾーン付き数値形式でなければなりません。Adabas は符号をゾーン形式で表現します。

PACKED と定義されたフィールドは、下位バイトの下位 4 ビットに有効な符号値を持っていない必要があります。有効な正の符号は A、C、E および F です。有効な負の符号は B および D です。Adabas は、正の値を C、負の値を D で表現します。

オーバーパンチ形式もサポートされています。詳細については、『VAX-11 Architecture Handbook』の「Data Representation」を参照してください。

入力ファイルにレコードがないと、警告メッセージが表示され、ユーティリティは異常終了します。しかし、FDT 情報を持つ CMPDTA 出力ファイルは作成されます。

マルチプルバリューフィールドカウント

非圧縮レコードの構造を FIELDS パラメータ以外の方法で記述する場合は、これから説明する点に注意してください。

マルチプルバリューフィールドの値の先頭には、レコード内にあるマルチプルバリューフィールドの値の数を示すために、ADACMP パラメータ MUPE_C_L の設定に応じた、1、2、または 4 バイトのバイナリカウントがなければなりません。指定できる値の最小数は 1 です。

値の数がどのレコードでも一定であれば、マルチプルバリューフィールドの定義に使用するフィールド定義テーブルにその数を指定することができます。この場合、入力レコード中にはカウントバイトを付けません。このオプションは、FDT キーワードを使用する場合だけ可能です。データベースから読み込まれる FDT のデフォルトのオカレンスカウントは、常に可変になります。このような可変のオカレンスカウントは、FIELDS キーワードを使用すれば値を変更できます。

ピリオディックグループ内のマルチプルフィールドは、ピリオディックグループのオカレンスカウントの指定が可変になっている場合には、オカレンスカウントを指定してはいけません。

例：

```
01,PG,PE
02,P1,4,A,NU
02,PM,4,A,NU,MU(4)
      ^
%ADACMP-E-FIX0CC, specification of occurrences not allowed at this position
```

ユーザーが指定するカウントは、そのフィールドに NU オプションが定義されていれば ADACMP で修正できます。空値は省略され、カウントフィールドはそれによって修正されます。

例：

Field Definition: 01,MF,4,A,MU,NU

各レコードの MF 値の数が可変です。

Input Records	Before ADACMP	After ADACMP
Input Record 1 (3 values)	MF count = 3 AAAA BBBB CCCC	MF count = 3 AAAA BBBB CCCC
Input Record 3 (3 values)	MF count = 3 AAAA <null value> CCCC	MF count = 2 AAAA CCCC
Input Record 4 (1 value)	MF count = 1 <null value>	MF count = 0

例：

Field Definition: 01,MF,4,A,MU(3),NU

各レコードの MF 値の数が 3 です。

Input Records	Before ADACMP	After ADACMP
Input Record 1	AAAA BBBB CCCC	MF count = 3 AAAA BBBB CCCC
Input Record 2	AAAA BBBB <null value>	MF count = 2 AAAA BBBB
Input Record 3	AAAA <null value> CCCC	MF count = 2 AAAA CCCC
Input Record 4	<null value> <null value> <null value>	MF count = 0

ピリオディックグループカウント

非圧縮レコードの構造を FIELDS パラメータ以外の方法で記述する場合は、これから説明する点に注意してください。

ピリオディックグループの最初のオカレンスの先頭には、レコードのピリオディックグループのオカレンス数を示す、1、2、または 4 バイト（ADACMP パラメータ MUPE_C_L の設定に依存する）のバイナリカウントがなければなりません。指定できるオカレンスの最小数は 1 です。

オカレンス数がどのレコードでも一定であれば、ピリオディックグループの定義に使用するフィールド定義テーブルにその数を指定することができます。この場合、入力レコード中にはカウントバイトを付けません。

このオプションは、FDT キーワードを使用する場合だけ可能です。データベースから読み込まれる FDT のデフォルトのオカレンスカウントは、常に可変になります。このような可変のオカレンスカウントは、FIELD キーワードを使用すれば値を変更できます。

設定済みのオカレンスカウントは、ピリオディックグループ内のすべてのフィールドが NU オプションで定義されている場合のみに、ADACMP で修正できます。あるオカレンスのすべてのフィールドが空値を持ち、空でない値を持つ後続のオカレンスがなければ、そのオカレンスは省略され、ピリオディックグループのオカレンスカウントはそれに従って修正されます。

例（NU オプションを指定した PE）

```
Field Definitions:  01,GA,PE
                   02,A1,4,A,NU
                   02,A2,4,A,NU
```

入力レコードには GA に対する可変数のオカレンスが含まれています。

Input Records	Before ADACMP	After ADACMP
Input Record 1	GA count = 2	GA count = 2
	GA (1st occ.)	
	A1 = AAAA	A1 = AAAA
	A2 = BBBB	A2 = BBBB
	GA (2nd occ.)	
	A1 = CCCC	A1 = CCCC
	A2 = DDDD	A2 = DDDD
Input Record 2	GA count = 1	GA count = 0
	GA (1st occ.)	
	A1 = <null value>	suppressed *
	A2 = <null value>	suppressed *
Input Record 3	GA count = 3	GA count = 3
	GA (1st occ.)	
	A1 = AAAA	A1 = AAAA
	A2 = <null value>	A2 = suppressed
	GA (2nd occ.)	
	A1 = BBBB	A1 = BBBB
	A2 = <null value>	A2 = suppressed

GA (3rd occ.)	
A1 = CCCC	A1 = CCCC
A2 = <null value>	A2 = suppressed

*これは2という値を持つ空フィールドカウントによって示されます。63までの連続する空フィールドが、1つの空フィールドカウントによって示されます。

例 (NU オプションを指定した PE)

Field Definitions: 01,GA,PE(3)
02,A1,4,A,NU
02,A2,4,A,NU

入力レコードの GA のオカレンスの数はどれも 3 つです。

Input Records	Before ADACMP	After ADACMP
Input Record 1	GA (1st occ.) A1 = AAAA A2 = <null value> GA (2nd occ.) A1 = BBBB A2 = <null value> GA (3rd occ.) A1 = CCCC A2 = <null value>	GA count = 3 A1 = AAAA A2 suppressed A1 = BBBB A2 suppressed A1 = CCCC A2 suppressed
Input Record 2	GA (1st occ.) A1 = <null value> A2 = <null value> GA (2nd occ.) A1 = BBBB A2 = <null value> GA (3rd occ.) A1 = <null value> A2 = <null value>	GA count = 2* A1 = suppressed A2 = suppressed A1 = BBBB A2 = suppressed A1 = suppressed A2 = suppressed
Input Record 3	All occ. contain null value	GA count = 0 All occurrences are suppressed **

* 空でない値を持つオカレンスが後に続くので、最初のオカレンスはカウントに含まれます。3 番目のオカレンスは、空でない値を持つオカレンスが後に続かないのでカウントには含まれません。

** これは 2 という値を持つ空フィールドカウントによって示されます。

例 (NU オプションの指定がない PE)

```
Field Definitions:  01,GA,PE(3)
                   02,A1,4,A
                   02,A2,4,A
```

入力レコードの GA のオカレンスの数はどれも 3 つです。

Input Records	Before ADACMP	After ADACMP
Input Record 1	GA (1st occ.) A1 = <null value> A2 = <null value>	GA count = 3 A1 = <null value> A2 = <null value>
	GA (2nd occ.) A1 = <null value> A2 = <null value>	A1 = <null value> A2 = <null value>
	GA (3rd occ.) A1 = CCCC A2 = <null value>	A1 = CCCC A2 = <null value>
Input Record 2	GA (1st occ.) A1 = <null value> A2 = AAAA	GA count = 3 A1 = <null value> A2 = AAAA
	GA (2nd occ.) A1 = <null value> A2 = <null value>	A1 = <null value> A2 = <null value>
	GA (3rd occ.) A1 = <null value> A2 = <null value>	A1 = <null value> A2 = <null value>

可変長の場合の長さインジケータ

可変長フィールド（フィールド定義に長さが0で指定されている）の各値の先頭には、（長さインジケータを含めた）値の長さを示す長さインジケータ（バイナリ形式）がなければなりません。

長さインジケータの長さは次のようになります。

- 4 バイト (L4 オプションを持つフィールドの場合)
- 2 バイト (LA オプションを持つフィールドの場合)
- 1 バイト (どちらのオプションも持たないフィールドの場合)

例：

Field Definitions:

```
01,AA,8,A,DE
01,V1,0,A
01,V2,0,A,LA
01,V4,0,A,L4
```

入力レコード（上位優先）

```
"FIELD AA\x09FIELD V1\x00\x0aFIELD V2\x00\x00\x00\x0cFIELD V4"
```

```
"FIELD AA\x09FIELD V1\x07\xd2 (2000 data bytes)\x00\x00\x07\xd2 (2000 data bytes)"
```

入力レコード（下位優先）

```
"FIELD AA\x09FIELD V1\x0a\x00FIELD V2\x0c\x00\x00\x00FIELD V4"
```

```
"FIELD AA\x09FIELD V1\xd2\x07 (2000 data bytes)\xd2\x07\x00\x00 (2000 data bytes)"
```

NC オプションのインジケータ

入力レコードの構造の規定に FDT を使用する場合、NC オプション付きのフィールドの値は、インジケータなしで定義します。

例：

Field Definitions:

```
01,AA,5,A,NC
01,AB,5,A,NC
```

Input Record

```
Field AA      Field AB
```

```
(5 bytes)    (5 bytes)
```

入力レコードの値が NC オプション付きの値に相当する場合、NULL_VALUE パラメータを設定するか、または FILEDS パラメータを使ってレコードの構造を規定しなければなりません。

ADACMP の処理に関する考慮

データ変更

ADACMP は、すべての入力レコードを次のように変更します。

フォーマット U または P で定義されたフィールドは、そのフィールド値が数値であり、正しいフォーマットであるかどうかチェックされます。

値が空の場合は、フィールドに指定したフォーマットに対応する文字を持っていなければいけません。

フォーマット	空値
英数字 (A)	空白 (ASCII の場合は 16 進 '20'、EBCDIC の場合 16 進 '40')
2 進数 (B)	バイナリのゼロ
固定小数点 (F)	バイナリのゼロ
浮動小数点 (G)	バイナリのゼロ
パック 10 進数 (P)	パック 10 進数の 0
アンパック 10 進数 (U)	アンパック 10 進数の 0 (ソースのアーキテクチャにより異なる)
Unicode (W)	WCHARSET に指定した空白

パックまたはアンパックの英数字フィールドの場合には、-0 は +0 に変換されます。

不当なデータを持つすべてのレコードは、ADACMP エラーファイルに書き込まれ、圧縮ファイルには書き込まれません。

データ圧縮

各フィールドの値は (FI オプションの指定がない限り) 次のように圧縮されます。

- A フォーマットで定義されたフィールドの末尾の空白は削除されます。
- B、P または U フォーマットで定義されたフィールドの場合、先頭のゼロは除去されます。
- フィールドが NU オプションを指定して定義され、かつ値が空値である場合は、1 バイトのインジケータが格納されます。16 進 'C1' は 1 つの空フィールドが続くことを示し、'C2' は 2 つの空フィールドが続くことを示します。以下同様です。
- レコードの最後に位置する空フィールドは、格納されません。

例：

次のデータ定義と対応する値は、ADACMP によって図に示すように処理されます。

```
01,ID,4,B,DE      ; ID
01,BD,6,U,DE,NU   ; BIRTHDATE
01,SA,5,P         ; SALARY
01,DI,2,P,NU      ; DAYS ILL
01,FN,8,A,NU      ; FIRST_NAME
01,LN,9,A,NU      ; LAST_NAME
01,SE,1,A,FI      ; SEX
01,H0,7,A,NU      ; HOBBY
```

Field	Format	Before compression	After compression
ID	B	67 12 00 00	03 67 12
BD	U	31 36 30 35 35 39	07 31 36 30 35 35 39
SA	P	00 00 05 00 0C	04 05 00 0C
DI	P	00 0C	))C2 (two empty fields)
FN	A	20 20 20 20 20 20 20 20	)
LN	A	4E 41 4D 45 20 20 20 20 20	05 4E 41 4D 45
SE	A	4D	4D
H0	A	20 20 20 20 20 20 20	C1 (one empty field)

ADAMUP の処理に関する考慮

Adabas データベースファイルのレコード追加または削除時は、アドレスコンバータ (AC)、データストレージ (DS) およびインデックスに、エントリの挿入／削除が必要になります。データストレージスペーステーブル (DSST) は、それにしたがって修正される必要があります。

レコードの追加

ISN の割り当て

USERISN オプションを設定すると、入力データを特定する ISN が使用されます。この ISN がファイルに対する現在の制限 (MAXISN) を超える場合、または同じ ISN がすでに別のレコードに割り当てられている場合は、ADAMUP は実行を終了してエラーメッセージを返します。Adabas N2 コマンドを使用したときと同じように、ファイルのアドレスコンバータの自動拡張はありません。ファイルの最初の空き ISN は、ファイルの現在の最初の空き ISN より大きいか等しい USERISN があれば、設定済みの最高 USERISN より 1 大きい値に設定されます。

USERISN オプションが省略されるか、NOUSERISN が指定されると、各レコードの ISN は ADAMUP によって割り当てられます。ISN は昇順に割り当てられます。ISN 再使用が有効な場

合は、ADAMUP は未使用の ISN があるかファイルのアドレスコンバータを探します。すべての ISN が再使用されているか、ISN 再使用が無効な場合は、ADAMUP はファイルの最初の空き ISN で始まる新しい ISN を割り当てます。新しいアドレスコンバータブロックが必要になると、現在使用可能なエクステントから割り当てられます。これらのブロックを使いつくすと、この章で説明しているルールに従って、自動拡張が実行されます。エクステントが割り当てられた場合には、処理は続行しますが、割り当てられなかった場合には、ADAMUP がエラーメッセージを出力して終了します。

並行に動作している大量削除によって削除される ISN は、追加されるレコードに対して直ちに再使用できます。

データストレージ内のスペースの発見

DS 再使用が有効になっている場合、ADAMUP は、現在のデータレコードの格納に十分なスペースを持つ DS RABN があるか DSST 内を探します。Adabas ニュークリアスが行うのと全く同じように、1 回につき 1 つずつ DSST RABN が調べられ、DSST にスペースが見つからなければ、最初の空き DSRABN が使用されます。大量削除が並行に実行されている場合は、レコードが削除された DS RABN が最初に再使用されます。これは、Adabas ニュークリアスの場合の手順とは異なりますが、DSST 内の検索を省き、データストレージへの入出力回数を最小化しています。こうするのは、いずれの場合にも削除ルーチンがこのような RABN を読み取り、書き込む必要があるからです。

DS 再使用が無効な場合、または DSST にスペースが見つからない場合、ADAMUP は最初の空き DS RABN で始まる新しい DS ブロックを割り当てます。

新しいレコードがデータストレージブロックに追加される場合は、ファイルに対して指定されているパディングファクタが考慮されます。新しいデータストレージブロックが必要な場合は、現在使用可能なエクステントから割り当てられます。これらのブロックを使いつくすと、自動拡張が実行されます。エクステントが割り当てられた場合には、処理は続行しますが、割り当てられなかった場合には、ADAMUP がエラーメッセージを出力して終了します。

レコードの削除

最初のステップで、削除対象の ISN を含むファイルの入力レコードがすべて読み込まれ、検証されます。正しくないレコードが発見されると、行番号とオフセットが表示され、入力ファイルの解析が完了すると、ADAMUP は実行を終了してエラーステータスを返します。

このステップの最後に、ADAMUP は削除対象の ISN のテーブルを仮想メモリ上に作成します。このテーブルは、ファイルのアドレスコンバータ、データストレージおよびインデックスに対して、必要な更新を行う次のステップで使用されます。このテーブルに必要なスペース（エントリごとに 1 ビット）は、入力ファイルに指定された最小および最大 ISN に依存します。十分なスペースがなければ、ADAMUP は実行を終了してエラーメッセージを返します。

2 番目のステップでは、ファイルのアドレスコンバータが処理されます。削除対象の ISN はあらかじめソートされているため、このステップのアドレスコンバータの入出力回数は最小限に抑えられます。

指定された各 ISN に対応するアドレスコンバータのエントリが検索されます。未使用の ISN については、NOT_PRESENT=IGNORE が指定されていれば（デフォルト）、エントリはエラーログに書き込まれ、処理は続行します。指定されていない場合は、ADAMUP は終了してエラーメッセージが返されます。使用されている ISN については、対応するデータストレージ RABN が SORT に入れられ、アドレスコンバータのエントリが削除されます。同じデータストレージ RABN への連続する参照はスキップされます。ファイルコントロールブロック（FCB）での位置がわかるように、SORT に入れられた各データストレージ RABN の前にはエクステント番号が先頭に付けられます。こうすることによって、次のステップでは、データストレージのエクステントが割り当てられた順番に従って、ファイルのデータストレージを処理できます。

このステップの最後で、ファイルの最初の空き ISN は、ISN の再使用が有効であれば、削除対象の ISN の最大範囲の最初の ISN に再設定され、削除対象レコードの範囲の最大 ISN は、ファイル上の最後に使用された ISN と同一になります。

3 番目のステップでは、ファイルのデータストレージとデータストレージスペーステーブルが処理されます。変更対象のデータストレージ RABN はその時点でソート済みなので、このステップではデータストレージとデータストレージスペーステーブルの入出力回数は最小限に抑えられます。

関連するデータストレージブロックが、SORT によって返される値を使用して読み込まれます。各ブロック内で、削除対象の ISN のテーブル内にある ISN と一致するレコードは削除され、（大量追加が同時に実行され、DS 再使用が有効な場合は）ブロックは追加対象のレコードの割り当てに再使用され、データストレージスペーステーブルがそれにしたがって変更されます。このステップの最後で、最初の空きデータストレージ RABN は、DS 再使用が有効であれば、すべてのデータが削除されたデータストレージ RABN の最終範囲の開始 RABN に再設定され、最終 RABN はファイル上の最後に使用されたデータストレージ RABN と同一になります。

インデックスの更新

アドレスコンバータ、データストレージおよびデータストレージスペーステーブルが変更されると、ADAMUP はファイルのノーマルインデックス（NI）を中間ファイルにコピーし、ファイルのインデックスエクステントをリセットします。削除レコードに対応するインデックスエントリは、このステップで省略されます。

ノーマルインデックスとメインインデックスのロード

ノーマルインデックスとメインインデックスを構築するため、入力ファイルに含まれるディスクリプタバリュートーブル（DVT）エントリを読み取り、昇順のディスクリプタ値と ISN に従ってソートする必要があります。このソートの出力は、中間ファイルに保存されたノーマルインデックスエントリとマージされ、後で新しいノーマルインデックスとメインインデックスの構築に使用されます。

ユニークオプションで定義されたディスクリプタは、新しいノーマルインデックスがディスクリプタ値ごとに 1 つのみの ISN を持っているかどうかチェックされます。複数の ISN があると、矛盾する ISN はエラーログに書き込まれ、UQ_CONFLICT=RESET が指定されていると、FDT

のユニークフラグがリセットされて処理は続行します。指定されていない場合は、ADAMUPはエラーメッセージを表示して終了します。

ディסקリプタ値のソートに加えて、ディスクリプタバリュートーブルの読み込みは、シーケンシャル入力ファイルへの大量の入出力の結果として非常に時間を要します。そのため、大量のディスクリプタがある場合は、ADAMUPは、ディスクリプタスペースサマリ (DSS) に含まれている情報を利用することによって、ディスクリプタバリュートーブルを介した読み込みの回数の最小化を試みます。ディスクリプタバリュートーブルを毎回、読み込むごとに、ディスクリプタの値が1つずつ、直接 SORT に渡されます。追加ディスクリプタがある場合は、それらは TEMP データセットに書き込まれます。各読み込みの間に並行して TEMP を使用するディスクリプタの数が多くなるほど、このステップも速くなります。ADAMUPは実行の最後で、必要となった読み込みの合計数を表示します。

ファイルのロード時に指定したパディングファクタに従って、すべてのインデックスブロックに値が設定されます。新しいインデックスブロックは必要に応じて、（このステップの開始時にリセットされている）既存のエクステントから割り当てられます。これらのブロックを使いつくすと、自動拡張が実行されます。エクステントが割り当てられた場合には、処理は続行しますが、割り当てられなかった場合には、ADAMUP がエラーメッセージを出力して終了します。

アッパーインデックスのロード

ノーマルインデックスとメインインデックスがディスクリプタ単位に構成されるのに対し、インデックスレベル3以上であるアッパーインデックスは、すべてのディスクリプタを含みます。新しいメインインデックスにリンクするため、新しいメインインデックスブロックごとに、1エントリがアッパーインデックスに作成されます。この時アッパーインデックス全体が再構築され、ファイルのロード時に指定したパディングファクタに従って、再度、値が設定されます。あらかじめ割り当てられているすべてのブロックは、追加ブロックが割り当てられる前に使用されます。追加ブロックが必要になると、ノーマルインデックスとメインインデックスのロードについて説明したのと同じ手順が使用されます。

エラーデータ

すべてのエラーデータは ADAMUP エラーファイルに書き込まれます。このエラーファイルの内容は、ADAERR ユーティリティを使用して表示します。レコードの中に印刷できない文字が含まれているため、オペレーティングシステム標準の印刷ユーティリティでエラーファイルを印刷しないでください。

詳細については、『ユーティリティマニュアル』の「ADAERR」を参照してください。

ADABCK の処理に関する考慮

DUMP/EXU_DUMP 機能

データベース全体のダンプ時 (DUMP=*) は、データベースのグローバル情報とロードされている全ファイルが Adabas バックアップコピーにダンプされます。したがって、データベースをデータベースバックアップコピーからリストアできます。そのような Adabas バックアップコピーに含まれている単一ファイルのリストアもできます。

選択ファイルのみのダンプ機能は、データベース全体のバックアップが不必要な場合に、データベースの一定部分のバックアップを制御可能とします。

DUMP/EXU_DUMP 機能は、ニュークリアスがアクティブでも非アクティブのときにでも使用できます。DUMP 機能の実行中にニュークリアスがアクティブの場合は、すべての更新がバックアップコピーにダンプされます。

AUTORESTART が保留になっているときに、DUMP/EXU_DUMP 機能は使用できません。このため、まずニュークリアスを起動して AUTORESTART の保留状態を解消しなければなりません。

DUMP 機能の終了直前には、すべてのトランザクションを ET ステータスに同期する必要があります。アクティブなニュークリアスは ADABCK の要求時にこれを自動的行います。同期の間、ニュークリアスは次のコマンドだけをスケジュールします。

- ET ユーザーが ET ステータスに達することができるようにするコマンド
- すべてのアクティブな更新コマンドを完了させるコマンド
- 読み取り／検索コマンド

DUMP/EXU_DUMP の実行中にニュークリアスが異常終了した場合、ニュークリアスの起動を要求するメッセージがデータベースのオペレータに送信されます。

DUMP 機能が実行しているときニュークリアスを起動することができます。この場合、ニュークリアスと DUMP 機能は相互に同期をとります。DUMP 機能が稼動しているとき、ニュークリアスは ADAOPR CANCEL でシャットダウンすることができます。ニュークリアスが異常終了すると、ADABCK はニュークリアス開始要求のメッセージを表示します。正常終了した後、ニュークリアスが自動再スタートを実行するまで待機状態になります。

並行バックアップ

複数の ADABCK ジョブを使用して、単一ファイルを並行してダンプすると便利なこともあります。これは、EXU_DUMP で一般的に有効ですが、ニュークリアスが稼動しているときは、1 つの DUMP 機能だけが実行できます。



注意: 並行バックアップは、Windows 環境で使用できません。

RESTORE/OVERLAY 機能

バックアップコピーは、単一ファイル、またはデータベースのグローバル情報が破壊されている場合に、指定ファイルまたはデータベースのリストア／オーバーレイに使用できます。

ADABCKDUMP=* により作成されたバックアップコピーを使用して、データベース全体をリストアできます。

ADABCK DUMP=(2, 5-8, 100-130) のように、ADABCK DUMP=(ファイルの列挙) という形式でバックアップコピーを作成すると、指定に従ってデータベースから抽出されたファイルだけがバックアップコピーとして作成されます。この種のバックアップは、通常、指定したファイルのリストアが必要な場合に使用します。また、この種のバックアップを使用して、ADABCK RESTORE=* または ADABCK OVERLAY=* でデータベース全体のリストアまたはオーバーレイを実行することもできます。ただし、これを実行した場合、ADABCKDUMP コマンドによってバックアップされたファイルのみがデータベースにリストアされ、その他すべてのファイルは削除されます。この場合、ADABCK は、リストア処理の結果、データベース内のファイル数合計が変化したことを示すメッセージ GCBAFL を出力します。

ファイルのリストアとオーバーレイは、ニュークリアスがアクティブかどうかに関係なく実行できます。リストア／オーバーレイ対象のファイルによって要求されるすべての RABN がチェックされます。すべての RABN が使用可能な場合、そのファイルは以前と同じ場所に置かれます。データベース内の要求した RABN のうち、1 つでも使用不可能なものがあれば、完全に新しい RABN の組み合わせが割り当てられます。

ファイルのリストアとオーバーレイは、ニュークリアスがアクティブかどうかに関係なく実行できます。リストア／オーバーレイの対象ファイルに必要な RABN が存在している必要があります。

データベースのリストア／オーバーレイ時は、ニュークリアスはアクティブであってはけません。それは、データベースコンテナファイルの排他制御が必要であるからです。

データベース全体のリストア／オーバーレイ時は、元のデータベースの方が大きいことがあります。これは、基準データベースがバックアップセーブセットよりも多くのブロック数またはコンテナ数を含むためですが、セーブセットで扱うブロックサイズは同一でなければなりません。基準データベースの未使用ブロックは保持され、そのスペースはフリースペーステーブルに戻されます。

ファイルをリストア／オーバーレイするとき、基準データベースはバックアップコピーより小さくしたり、または大きくしたりすることができます。

ファイルをリストア／オーバーレイするとき、ADABCKはデフォルトで元のブロック番号にブロックをリストアしようとします。このスペースが他のファイルに使用されているために利用できない場合、ファイルは完全に他のブロック番号にリストアされ、複数のファイルエクステンントを1エクステンントに結合しようとします。

並行リストア

複数のADABCKジョブを使用して、単一ファイルを並行してリストアすると便利なこともあります。これは、ニュークリアスが稼動しているかどうかに関係なく、RESTOREおよびOVERLAY機能の両方で有効です。

セキュリティファイルの考慮

セキュリティファイルのリストア／オーバーレイ時は、パスワードおよび関連パーミッションレベルのみが再設定され、ロードされるファイルの保護レベルは再設定されません。したがって、新たにフォーマット化されたデータベースにファイルをリストアする場合は、ADASCR セキュリティユーティリティを使用して保護レベルを再度使用可能にしなければなりません。

データベースをリストア／オーバーレイすると、全ファイルの保護レベルの再設定のみが行われます。

ADABCK の再スタートに関する考慮

ADABCK には再スタート機能は備えられていません。ADABCK が異常終了した場合、最初から再実行する必要があります。

1つ以上のファイルに対してRESTORE/OVERLAY 機能を実行しているときに処理が中断されると、RABN の割り当ては不完全になりますが、ADADBМ ユーティリティの RECOVER 機能を実行すれば完全な状態に戻すことができます。データベースに対して RESTORE/OVERLAY 機能を実行しているときに処理が中断されると、そのデータベースにはアクセスできなくなります。

ADAORD の処理に関する考慮

ファイルのエクスポート

1つ以上のファイルをエクスポートするとき、ADAORD は各ファイルのデータストレージの内容を、そのインデックスを再構築するのに必要な情報と一緒にシーケンシャル出力ファイル（ORDEXP）にコピーします。ファイルのデータレコードのエクスポートは、アンロードと同じで、ADAORD は ADAULD ユーティリティと同様の処理順序をサポートします。しかし、ファイルのインデックスを再構築するのに必要な情報が提供される方法が異なります。ADAORD は、（ADAULDのように）データレコードに基づいたディスクリプタバリュートーブル（DVT）エントリを生成しませんが、ファイルのインバーテッドリストを回復し、エクスポートします。ADAORD は、有効なインデックスにアクセスする必要があるため、I/O 数が増える結果となりますが、一方で CPU 時間を節約できます。

処理対象のファイルすべては、ファイル番号の小さい方から順番に単一シーケンシャル出力ファイル（ORDEXP）に書き出されます。後でファイルを再度インポートするときに、デフォルト値でない割り当て量または配置を使用したいのであれば、エクスポートを複数回の実行に分けて、シーケンシャル出力ファイルを複数作成することも考えられます。デフォルト以外の値や配置を使用する場合、各ファイルは別々に実行する必要があるため、エクスポートの手順を分けると、再度インポートするときに長く時間のかかる位置づけをしなくても済みます。

ファイルのインポート

1つ以上のファイルをインポートするとき、各ファイルのデータストレージ、アドレスコンバータおよびインデックスを再構築するために、ADAORD は、シーケンシャル入力ファイル（ORDEXP）に含まれる情報を検索します。ファイルのデータレコードをインポートし、アドレスコンバータを構築することは、（USERISN オプション付き）ADAMUP ユーティリティを使用してそれらをロードすることと同じです。しかし、ディスクリプタ値と ISN は正しい順序で提供されるため、ファイルのインデックスを構築する処理は ADAORD の方が速くなります。このため、ソートを実行（SORT ファイルおよび TEMP ファイルを使用）する必要がなくなり、EXPORT 段階でのインデックスの読み込みにかかる時間を十分に相殺できます。

シーケンシャル入力ファイル（ORDEXP）のフォーマットは、データベースのデバイスタイプとは無関係です。このため、エクスポートとその後のインポートの処理を使用して、異なるデバイスタイプに存在するデータベース間でファイルを移動できます。

セキュリティファイルをインポートするとき、パスワードと関連する許可レベルだけが再構築されます。インポートされたファイルの保護レベルは再構築されません。このため、新規にフォーマットされたデータベースにファイルをインポートする場合、ADASCR ユーティリティを使用して保護レベルを再度有効にする必要があります（詳細については、『ユーティリティマニュアル』を参照）。

セキュリティファイルをインポートするとき、パスワードと関連する許可レベルだけが再構築されます。インポートされるファイルがデータベース内でセキュリティ保護されていない場合、バックアップコピーに保存されている保護レベルがリストアされます。

スペースの割り当て

ファイルをインポートするとき、その配置および初期割り当て量はユーザーが制御することも、ADAORD に制御を任せることもできます。

開始 RABN または KEEP_LAYOUT オプションを指定して強制的に位置指定しない限り、ADAORD は、次の順序でファイルのエクステントの初期割り当てを行います。アドレスコンバータ (AC)、アップパーインデックス (UI)、ノーマルインデックス (NI)、およびデータストレージ (DS)。

開始 RABN を指定して強制的に位置指定しない限り、ADAORD は、次の順序でファイルのエクステントの初期割り当てを行います。アドレスコンバータ (AC)、アップパーインデックス (UI)、ノーマルインデックス (NI)、およびデータストレージ (DS)。

これにより、使い尽くす可能性の最も高い2つのエクステントタイプ (NI と DS) が他のエクステントに割り込まずに拡張できます。

割り当てられるブロックまたはシリンダの数が省略され、KEEP_LAYOUT オプションが指定されていない場合、ADAORD は次のように割り当て量を計算します。

$$ALQN = ALQO * (1 + (PFACN - PFACO) / 100)$$

上記式内の各項目の意味は次のとおりです。

ALQN	新しく割り当てられるブロック数
ALQO	今まで割り当てられていたブロック数
PFACN	新しく使用されるパディングファクタ
PFACO	今まで使用されてきたパディングファクタ

デフォルトでは、最初から最後まで一貫して、できるだけ物理的に連続した領域に割り当てようとする方式が適用されます。CONTIGUOUS パラメータを指定すると、指定した各タイプの最初の論理エクステントのみが使用されます。ただし、連続したスペースが十分に取れないと、ADAORD が異常終了する可能性があります。

割り当てるブロック数またはメガバイト数を指定しないと、ADAORD は次のように割り当て量を計算します。

$$ALQN = ALQO * (100 - PFACO) / (100 - PFACN)$$

上記式内の各項目の意味は次のとおりです。

ALQN	新しい割り当て量（ブロック単位またはメガバイト）
ALQO	今までの割り当て量（ブロック単位またはメガバイト）
PFACN	新しく使用されるパディングファクタ
PFACO	今まで使用されてきたパディングファクタ

デフォルトでは、最初から最後まで一貫して、できるだけ物理的に連続した領域に割り当てようとする方式が適用されます。

ISN の割り当て

データレコード単位に設定された（および、インバーテッドリストに含まれる）ISN が使用されます。ファイルの制限値（MAXISN）をファイルの最初の空き ISN 値よりも小さく設定した場合、その新しい制限値を超える ISN があると、ADAORD は実行を終了し、エラーメッセージを返します。ファイルの最初の空き ISN は、データレコード内の最大 ISN よりも 1 大きい値に設定されます。

ISN 割り当てを変更するには、ADAULD を使用して該当ファイルをアンロードし、ADAMUP を使用して再ロードしなければなりません。

データベースの再構築

この処理を行うということは、EXPORT 機能と IMPORT 機能を使用することを意味しています。

データベースレベルで再構築するとき、データベースのすべてのファイルを第 1 段階でエクスポートしなければなりません。単一バージョンの ORDEXP が、物理的に存在する場所とは別に作成されます。

第 2 段階では、データベースの FCB 領域と FDT 領域を再配置し、DSST を再度割り当てます。

最終段階では、ファイルを再インポートします。各ファイルは再配置され、複数の論理エクステン트는単一の論理エクステン트에凝縮されて、パディングファクタは再設定されます。

作成されたシーケンシャルファイル（ORDEXP）はこの機能終了時に削除されません。

ファイルレベルでリオーダして、ディスクを中間ストレージとして使用するとき、ADAORD は、指定されたファイルの処理前にシーケンシャルファイル ORDEXP を作成し、処理後にシーケンシャルファイル ORDEXP を削除して、一時スペース要件を最小化します。これにより、EXPORT 機能の後に IMPORT 機能を何回も続けることができます。データベース内のファイルの位置が変わらず、ファイルの割り当て領域内のみで上書きが発生するため、これが可能になっています。ただし、NODELETE オプションが指定されているか、ORDEXP がテープ上に存在する場合、データベースレベルでのリオーダで説明したように、ADAORD は単一のファイルを作成し処理します。テープ上では、データベースレベルで処理することにより、複数のバージョンの ORDEXP が存在することによる問題を回避し、テープ処理を効率化します。

一貫性のないインデックスの修復

新しいインデックスは（ファイルのデータレコードではなく）旧インデックスの内容に基づくものであるため、ファイルのエクスポートと再インポートによって、論理的に一貫性のないインデックスを修復することはできません。また、物理的に破壊されたインデックスがあると、ADAORD の EXPORT 機能がループしたり、異常終了したりすることがあります。

インデックスを修復するには、ADAINVを使用して再インバートするか、ADAULDとADAMUPを使用してファイルをアンロードしてから再ロードします。

ファイルのスペースの見積り

ここでは、ファイルに必要なアソシエータおよびデータストレージのスペース要件を求める計算式について説明します。

最初の見積もりの取得

以降では、データをロードする前にファイルやデータベースに必要なディスクスペースの正確な見積もりを取得するための適切な方法について説明します。最初の概算を取得する単純な方法は少量のデータ（例えば1～2%）をデータベースにロードし、ADAREPユーティリティを実行して、割り当て済みブロックと未使用ブロックとして出力された数字をチェックします。その後、100%のデータに必要なスペースの大きさを求めるためにこれらの数字を推定します。これは、スペース要件を求めるために熟練したデータベース管理者が使用するアプローチです。

アソシエータのスペースの見積り

ファイルに必要なアソシエータスペースは次のアソシエータエレメントに必要なスペースの合計です。

1. ノーマルインデックス

ノーマルインデックスは、インデックス構造の最小レベルです。これはインバーテッドリストを持ちます。各インバーテッドリストは、ディスクリプタ値とこのディスクリプタ値を参照するすべてのレコードのISNリストから構成されます。

2. アッパーインデックス

アッパーインデックスはメインインデックスおよび他のアッパーインデックスレベルで構成されます。メインインデックスはノーマルインデックスの後のインデックス構造の次に高いレベルです。これを使用してノーマルインデックスを管理します。このレベルまでの各インデックスブロックには1ディスクリプタだけのエントリが含まれりことがあります。

アッパーインデックス（インデックスレベル3以上）は、存在している全ディスクリプタに対するエントリを持ちます。レベル3は、メインインデックスの管理に使用されます。現在

のレベルのアップーインデックスブロックが複数あれば、さらにレベルが追加され、各レベルが下位レベルを管理します。

3. アドレスコンバータ

アドレスコンバータは、RABN のテーブルで構成され、それぞれの RABN は、ISN で識別されるレコードがデータストレージ内のどの場所にあるのかを示します。

ノーマルインデックスのスペースの見積り

ノーマルインデックスに必要なスペースは、ファイルにあるディスクリプタの数とその特性によって異なります。

各ディスクリプタに必要なノーマルインデックスの容量は、次の計算式によって求められます。

$$\text{NIBY} = (\text{IL} * \text{UV} * \text{MAXISN}) + \text{DV} * (\text{L} + 2)$$

上記式内の各項目の意味は次のとおりです。

NIBY

ノーマルインデックスのスペース要件（バイト単位）です。

UV

各レコードにあるディスクリプタのユニーク値の平均個数です。

ディスクリプタが MU オプションで定義されていない場合、UV は 1 以下です。

ディスクリプタが NU オプション指定で定義されている場合、UV はレコード当たりの値の平均個数から空値を持つレコードの比率を差し引いた値です。例えば、各レコードに存在する値の平均個数が 1 で、値の 20 % が空値の場合、UV は $1 - 0.2 = 0.8$ になります。

MAXISN

ファイル内の最大レコード数（ユーティリティ ADAFDU の MAXISN パラメータを参照）です。

DV

ファイル内のディスクリプタの異なる値の個数です。

L

ディスクリプタの値の平均長。FI オプションを指定しないでディスクリプタを定義した場合、L は平均長になります。FI オプションを指定してディスクリプタを定義した場合は、L はディスクリプタの標準長になります。

IL

ISN 自体の長さ。可能な値は 2 バイトまたは 4 バイトです。

$\text{IL} * \text{UV} * \text{MAXISN}$ の項は、ISN を格納するのに必要なスペースを表し、 $\text{DV} * \text{L}$ の項はディスクリプタ値を格納するのに必要なスペースを表しています。

重複値が多いディスクリプタについては、 $\text{IL} * \text{UV} * \text{MAXISN}$ の項は重要です。ユニーク値が大きな割合を占めるディスクリプタについては、 $\text{DV} * \text{L}$ の項が重要な要素となります。

一括更新ユーティリティ ADAMUP を使用してデータをロードする場合やインバーテッドリストユーティリティ ADAINV でインデックスを作成する場合に限り有効です。S1 コールを使用してデータをロードするときは、（最悪の場合）2 倍のスペースが必要となり、ブロックの一部が埋まらなくなります。この場合、新しい値をソート順に従ってブロックに追加しなければなりません。ブロック内の空きスペースが足りない場合、インデックスブロックは分割されます。

例 1：バイトの計算

ディスクリプタ AA は、各レコードに平均 1 個の値を持っています。ファイルには、50 個の異なる AA 値が存在します。AA に空値はありません。値の平均長は 3 バイトです。ファイルに設定した MAXISN は 20000 で、ISN サイズは 2 バイトです。

```
Field Definition: 01,AA,5,U,DE
NI = (2 * 1 * 20,000) + 50*(3 + 2)
NI = 40,000 + 250
NI = 40,250 bytes
```

例 2：バイトの計算

ディスクリプタ BB は、各レコードに平均 1 個の値を持っています。ファイル内での BB は 20000 個の異なる値をもっています。BB に空値はありません。値の平均長は 10 バイトです。ファイルの MAXISN 設定が 20000、ISN のサイズが 4 バイトだとすると、次のようになります。

```
Field Definition: 01,BB,15,A,DE
NI = (4 * 1 * 20,000) + 20,000*(10 + 2)
NI = 80,000 + 240,000
NI = 320,000 bytes
```

例 3：バイトの計算

ディスクリプタ CC は、マルチプルバリューフィールドで各レコードに平均 10 個の値を持っています。ファイルには、およそ 300 個の異なる CC 値が存在します。値の平均長は 4 バイトです。各レコードに平均 3 個の空値が存在します。ファイルの MAXISN 設定が 20000、ISN のサイズが 4 バイトだとすると、次のようになります。

```
Field Definition: 01,CC,12,A,DE,MU,NU
NI = (4 * 7 * 20,000) + 300*(4 + 2)
NI = 560,000 + 1,800
NI = 561,800 bytes
```

例 4：バイトの計算

ディスクリプタ DD は、ピリオディックグループ内のフィールドです。各レコードには DD の値が平均で 5 個あります。ファイルには、10 個の異なる DD 値が存在します。各レコードに平均 3 個の空値が存在します。ファイルの MAXISN 設定は 20000 です。値の平均長は 5 バイトで、ISN サイズは 2 バイトです。

```
Field Definition: 01,PX
                  02,DD,8,A,NU
NI = (2 * 2 * 20,000) + 10*(5 + 2)
NI = 80,000 + 70
NI = 80,070 bytes
```

ノーマルインデックスに必要なバイト数が決まれば、必要なブロック数は次の計算式で求めることができます。

$$\text{NIBL} = \text{NIBY} / (\text{BL} * (1 - p / 100) - 3)$$

上記式内の各項目の意味は次のとおりです。

NIBL

ノーマルインデックスのスペース要件（ブロック数）

NIBY

ノーマルインデックスのスペース要件（バイト）

BL

アソシエータブロック長

p

アソシエータブロックのパディングファクタ

割り算の結果は、整数に切り上げます。

例

```
NI requirement in bytes = 60,250
Device type RA92
Associator block padding factor = 10 percent
NIBL = 60,250 / (2044 * (1 - 10 / 100) - 3)
NIBL = 32+ = 33 blocks
```

例 5：ブロックの計算

```
NI requirement in bytes = 60,250
Device type 2 KB
Associator block padding factor = 10 percent
NIBL = 60,250 / (2048 * (1 - 10 / 100))
NIBL = 32+ = 33 blocks
```

アッパーインデックスのスペースの見積り

アッパーインデックスは、メインインデックスおよび他のアッパーインデックスで構成されます。メインインデックスでは、各ノーマルインデックスが9バイトの固定部分とディスクリプタ値で構成されます。各ディスクリプタに必要なメインインデックスのスペースは、次の計算式で求めることができます。

$$\text{MIBY} = \text{NIBL} * (\text{L} + 9)$$

上記式内の各項目の意味は次のとおりです。

MIBY

メインインデックスのスペース要件（バイト単位）

NIBL

ノーマルインデックスのスペース要件（ブロック数）

L

ディスクリプタの値の平均長。FI オプションを指定しないでディスクリプタを定義した場合、L は平均長になります。FI オプションを指定してディスクリプタを定義した場合は、L はディスクリプタの標準長になります。A および W フォーマットのフィールドでは、ディスクリプタ値がどの場所で切り捨てられるのかを意識する必要があります。ディスクリプタ値は、前のディスクリプタ値と比べて違っている部分の最初のバイトで切り捨てられます。

例 1：バイトの計算

$$\begin{aligned}\text{NI Block Requirement} &= 45 \text{ blocks} \\ \text{MI} &= 45 * (3 + 9) \\ \text{MI} &= 540 \text{ bytes}\end{aligned}$$

次の計算式を使用してメインインデックスの容量をバイト数からブロック数に変換することができます。

$$\text{MIBL} = \text{MIBY} / (\text{BL} * (1 - \text{P} / 100))$$

上記式内の各項目の意味は次のとおりです。

MIBL

メインインデックスのスペース要件（ブロック数）

MIBY

メインインデックスのスペース要件（バイト単位）

BL

アソシエータブロック長

P

アソシエータブロックのパディングファクタ

除算の結果は、整数に切り上げられます。

例 2：ブロックの計算

```

MI byte requirement = 540 bytes
Device type 2 KB
Associator block padding factor = 5 percent
MIBL = 540 / (2048 * (1 - 5 / 100))
MIBL = 0+ = 1 block

```

スペース全体の要件

最上位のアップパーインデックスレベル（レベル3以上）には、ファイルの全ディスクリプタに対するエントリが含まれています。アップパーインデックスに必要なスペース全体は、次の計算式で求めることができます。

$$UIBL = M * (1 + C + C^{**2} + C^{**3} + \dots + C^{**13})$$

上記式内の各項目の意味は次のとおりです。

UIBL

アップパーインデックスのスペース要件（ブロック）

M

ファイルの全ディスクリプタに必要なメインインデックスのスペースの合計

C は次の計算式で求められます。

$$C = (L + 13) / (BL * (1 - P/100))$$

上記式内の各項目の意味は次のとおりです。

L

ファイルの全ディスクリプタのすべての値の平均長

BL

アソシエータブロック長

P

アソシエータブロックのパディングファクタ

アドレスコンバータのスペースの見積り

ファイルのアドレスコンバータは、相対 Adabas ブロック番号（RABN）のリストから構成され、それぞれの RABN は、レコードの保存先に当たるデータストレージのブロック番号を表します。ブロック番号は ISN 順に格納されます。この際、ISN *n* にはデータストレージ RABN を持つ *n* 番目のエントリが対応します。各エントリに対して 3 バイト必要になります。

アドレスコンバータに必要なスペースは、次の計算式で求めることができます。

$$AC = MAXISN * 3 / BL$$

上記式内の各項目の意味は次のとおりです。

AC

アドレスコンバータのスペース要件（ブロック数）

MAXISN

ファイルの MAXISN 設定

BL

アソシエータブロックサイズ

除算の結果は、整数に切り上げられます。

例：

```
MAXISN = 2,000,000
Device type 2 KB
AC      = 2,000,000 * 3 / 2048
AC      = 6,000,000 / 2048
AC      = 2929+ = 2930 blocks
```

データストレージのスペース要件

データストレージのスペース要件は、次の計算式で求めます。

$$DS = N / (BW / L) + 1$$

上記式内の各項目の意味は次のとおりです。

DS

データストレージのスペース要件（ブロック数）

X

ファイルにロードされるレコード数

B

データストレージのブロックサイズ

P

データストレージのブロックパディングファクタ

BW

計算式 $B * (1 - p / 100)$ による実際のスペース容量

L

平均レコード長

例：

```
Number of records = 1,000,000
Average compressed record length = 50
Device type = 4 KB
Data Storage block padding factor = 5 percent
BW =  $4096 * (1 - 5/100) = 3891$ 
DS =  $1,000,000 / (3891/50) + 1 = 12,988$  blocks
```


9 ユーザー出口とハイパー出口

■ ユーザー出口の概要	164
■ ユーザー出口の説明	165
■ ハイパー出口の概要	191
■ ハイパー出口コントロールブロックとバッファ	193
■ ハイパー出口インターフェイス	202
■ ユーザー出口およびハイパー出口の作成と定義	206

この章では Adabas でサポートされるユーザー出口とハイパー出口について説明します。

ユーザー出口の概要

ユーザー出口とは、ユーザーが作成するルーチンであり、ユーザーは、これにより Adabas ニュークリアスまたは Adabas ユーティリティが実行する処理に独自の処理を追加することができます。ユーザー出口は、ニュークリアスまたはユーティリティ入力パラメータで設定します。ユーザー作成ルーチンは、ニュークリアスまたはユーティリティの起動時に動的にロードされ、ニュークリアスまたはユーティリティの処理においてあらかじめ定義された過程で呼び出されます。

ルーチンは、C プログラミング言語で記述する必要があります。

ユーザー出口の実装形態は、共有ライブラリ（UNIX）、またはダイナミックリンクライブラリ（Windows）です。このため、ユーザー出口やハイパー出口のコンパイルとリンクには、対応するオプションを指定する必要があります共有ライブラリは、位置に依存しないコードにする必要があるため、コンパイラは PIC オプションを使用してコールする必要があります。ダイナミックリンクライブラリ（DLL）は、マルチスレッド用にコンパイルする必要があります。

Adabas はユーザー出口の場所を示す環境変数／論理名を使用します。詳細については、「ユーザー出口およびハイパー出口の作成と定義」を参照してください。

Adabas では、次のユーザー出口が使用できます。

ユーザー出口	説明
ニュークリアスユーザー出口 1	Adabas ダイレクトコール時のユーザー処理。ニュークリアス処理の前に実行されます（場所の指定には環境変数／論理名 ADAUEX_1 を使用）。
ニュークリアスユーザー出口 2	プロテクションログまたはコマンドログファイルのクローズ時のユーザー処理（場所の指定には環境変数／論理名 ADAUEX_2 を使用）。
ニュークリアスユーザー出口 4	CLOG 出力レコードに対するユーザー処理。CLOG ファイルに書き込まれる前に実行されます（場所の指定には環境変数／論理名 ADAUEX_4 を使用）。
ADACMP ユーザー出口 6	ADACMP 入力レコードに対するユーザー処理。ADACMP が圧縮を行う前に実行されます（場所の指定には環境変数／論理名 ADAUEX_6 を使用）。
ADAULD ユーザー出口 7	Adabas 圧縮レコードに対するユーザー処理。ADAULD 処理の前に実行されます（場所の指定には環境変数／論理名 ADAUEX_7 を使用）。
ニュークリアスユーザー出口 11	Adabas ダイレクトコール時のユーザー処理。ニュークリアス処理の前に実行されます（場所の指定には環境変数／論理名 ADAUEX_11 を使用）。
ニュークリアスユーザー出口 14	CLOG V6 出力レコードに対するユーザー処理。CLOG ファイルに書き込まれる前に実行されます（場所の指定には環境変数／論理名 ADAUEX_14 を使用）。

ユーザー出口	説明
ニュークリアスユーザー出口 21	認証資格情報の設定。Adabas サーバー API 機能を使用します。
ADALNK ユーザー出口 0	ユーザー処理。Adabas コール実行の前に実行されます（場所の指定には環境変数／論理名 LNKUEX_0 を使用）。
ADALNK ユーザー出口 1	ユーザー処理。Adabas コール実行の後に実行されます（場所の指定には環境変数 LNKUEX_1 を使用）。
ADALNKX ユーザー出口 0	ユーザー処理。Adabas コール実行の前に実行されます（場所の指定には環境変数／論理名 LNKUEX_ACBX_0 を使用）。
ADALNKX ユーザー出口 1	ユーザー処理。Adabas コール実行の後に実行されます（場所の指定には環境変数 LNKUEX_ACBX_1 を使用）。
XA のユーザー出口	詳細については、このマニュアルの「XA サポート」を参照してください（場所の指定には環境変数／論理名 XAUEX_0 を使用）。

ニュークリアス起動時に、ユーザー出口は以下の形式の初期化コールで最初に呼び出されなくてはなりません。

```
uex_X (0, UEX_INIT*)
```

X はユーザー出口の番号を指し、1、2、4 のいずれかです。

構造体 UEX_INIT (adauex.h で定義されている) は Adabas とユーザー出口間の入出力値に使用します。入力パラメータはデータベース ID と現在のバージョンです。uex_type に UEX_REENTRANT または UEX_N_REENTRANT を設定することにより、ユーザー出口のコードをリエントラントにするかどうかを Adabas に通知する必要があります。ストレージクラスを自動にする変数だけを使用する場合は、そのコードをリエントラントにします。コードがリエントラントでない場合、そのユーザー出口がコールされると、順番に処理されますが、リエントラントの場合には並行して処理されます。

ユーザー出口の説明

ニュークリアスユーザー出口 1

説明

Adabas ニュークリアスユーザー出口 1 は、Adabas ダイレクト コール上のユーザー処理を実行するユーザー出口です。ルーチンは、コマンド処理の開始時にコールされます。指定された入力パラメータによって、ユーザー出口による Adabas コールのパラメータの変更やコールの拒否が可能になります。コールが拒否された場合は、そのコールを発行したユーザーに Adabas レスポンスコード 22（無効なコマンド）が返されます。

ユーザー出口では、コマンドコードを変更したり、Adabas コントロールブロックで指定されたバッファ長を変更したりすることはできません。これらの値を変更すると、コマンドを発行したユーザーに Adabas レスポンスコード 22（無効なコマンド）が返されます。

ニュークリアスユーザー出口 1 は、ADANUC で USEREXITS=1 と設定することによりアクティブになります。

入力パラメータ

```
Format: uex_1 ( pcb, pfb, prb, psb, pvb, pib, pcq )
```

pcb : 用途 : Adabas コントロールブロック
型 : unsigned char *
アクセスの種類 : 読み込み／書き込み
値の渡し方 : 参照渡し

Adabas コントロールブロックの構造については、『Adabas コマンドリファレンスマニュアル』を参照してください。

pfb : 用途 : Adabas フォーマットバッファ
型 : unsigned char *
アクセスの種類 : 読み込み／書き込み
値の渡し方 : 参照渡し

pcb が 0 ポインタの場合、pfb は UEX_INIT 構造をポイントします（初回コール時）。

prb : 用途 : Adabas レコードバッファ
型 : unsigned char *
アクセスの種類 : 読み込み／書き込み
値の渡し方 : 参照渡し

psb : 用途 : Adabas サーチバッファ
型 : unsigned char *
アクセスの種類 : 読み込み／書き込み
値の渡し方 : 参照渡し

pvb： 用途： Adabas バリユーバッファ
 型： unsigned char *
 アクセスの種類： 読み込み／書き込み
 値の渡し方： 参照渡し

pib： 用途： Adabas ISN バッファ
 型： unsigned char *
 アクセスの種類： 読み込み／書き込み
 値の渡し方： 参照渡し

pcq： 用途： Adabas コマンドキューエレメント
 型： struct cq_entry *
 アクセスの種類： 読み込み
 値の渡し方： 参照渡し

Adabas コマンドキューエレメントの詳細については、ヘッダーファイル `adauex.h` を参照してください。

戻り値

ユーザー出口戻り値は、基本的に Adabas レスポンスコードです。

ADA_NORMAL： 成功。

ニュークリアスは、コントロールブロック内で無効な変更が行われていないかどうかチェックします。無効な変更が検出されなかった場合、このコールは処理されます。

その他： 失敗。

Adabas コールは拒否され、レスポンスコード 22（無効なコマンド）が返されます。

ニュークリアスユーザー出口2

説明

Adabasニュークリアスユーザー出口2は、コマンドログファイル（CLOG）またはプロテクションログファイル（PLOG）を閉じるときにユーザー処理を実行するユーザー出口です。このユーザー出口は、次の状況でファイルが閉じられた後にコールされます。

- ニュークリアスがシャットダウンした（ADAOPR SHUTDOWN または CANCEL 機能）
- PLOG/CLOG が強制変更された（ADAOPR FEOF 機能）
- PLOG/CLOG が自動変更された（新しいエクステンツ）
- オンラインダンプ後に新しい PLOG が作成された（ADABCK NEW_PLOG 機能）
- 自動再スタート中

このユーザー出口は、（ADADEV などによる）PLOG のアーカイブにも使用されます。

ニュークリアスユーザー出口2は、ADANUC で USEREXITS=2 と設定することによりアクティブになります。

入力パラメータ

```
Format: uex_2 ( sess_num, dbid, env_var_cnt, logname, status )
```

sess_num : 用途 : PLOG セッション番号
型 : int *
アクセスの種類 : 読み込み
値の渡し方 : 参照渡し

sess_num は、ニュークリアスセッションの番号です。値がゼロの場合は、現在の PLOG がクローズされたことを示します。ポインタが 0 の場合、これは INIT-call となります。この場合、次のパラメータは UEX_INIT 構造をポイントします。

dbid : 用途 : データベース ID
型 : int *
アクセスの種類 : 読み込み
値の渡し方 : 参照渡し

dbid は、データベース ID のポインタです。sess_num が 0 ポインタの場合、UEX_IMT 構造体を指します（初期コール時）。

env_var_cnt : 用途 : 環境変数カウンタ
型 : int
アクセスの種類 : 読み込み
値の渡し方 : 値渡し

env_var_cnt は、現在の環境変数カウンタです。複数の環境変数を使用する場合に指定します。PLOG の場合、NUCPLG が 1、NUCPLG2 が 2、NUCPLG3 が 3 のようになります。ユーザー出口では、PLOG のパス名を取得するために環境変数を変換することが可能です。

logname : 用途 : セクション内のファイル名
型 : char *
アクセスの種類 : 読み込み
値の渡し方 : 参照渡し

logname は、ディスクセクション内の PLOG/CLOG の名前です (PLG.15 または PLG.11(3) など)。logname は、セクションのパス名とともに ADADEV により直接使用され、PLOG/CLOG が閉じた状態でもそれらを保存できます。

status : 用途 : コールのステータス
 型 : int
 アクセスの種類 : 読み込み
 値の渡し方 : 値渡し

status は、ユーザー出口がコールされたタイミングを示します。PLOG/CLOG の変更時には UEX2_SWITCH、ニュークリアスセッションの終了時には UEX2_SHUTDOWN、自動再スタート中のクラッシュ後にニュークリアスが起動したときは、UEX2_AUTORESTART となります。

戻り値

ユーザー出口の戻り値は、基本的に Adabas レスポンスコードです。

ADA_NORMAL : 正常終了。

その他 : 失敗。
 Adabas ニュークリアスはユーザー出口リターンコードを含むメッセージを出力します。ユーザー出口は無効になり、以降はコールされなくなります。

ニュークリアスユーザー出口 4

説明

Adabas ニュークリアスユーザー出口 4 は、CLOG 出力レコードに対するユーザー処理を、レコードが CLOG ファイルに書き込まれる前に実行します。このユーザー出口では、レコードの短縮、拡張、および完全な変更が可能です。レコードの長さまたは構造が変更された場合、CLOG を出力するために使用する ADACLP ユーティリティは、作成されたレコードを出力できなくなる場合があります。CLOG レコードが書き込まれたときにユーザー出口がアクティブな場合、警告メッセージが発行されます。

ニュークリアスユーザー出口 4 は、ADANUC で USEREXITS=4 と設定することによりアクティブになります。

入力パラメータ

Format: uex_4 (pcl, pcq, pdsc)

pcl： 用途： CLOG 出力レコード
型： struct cl_entry *
アクセスの種類： 読み込み／書き込み
値の渡し方： 参照渡し

CLOG レコードの構造については、adauex.h に含まれる CL_ENTRY の構造体定義を参照してください。

CLOG レコードの先頭 2 バイトにはレコード長が含まれます。ADACL P の情報が含まれる 3 バイト目および 4 バイト目は、変更しないでください。この 2 バイトは、ユーザー出口ルーチンへのコール後に、ニュークリアスによって以前の状態に戻されます。

ユーザー出口入力レコードの長さは、拡張しないでください。ユーザー出口から追加のデータを返すには、追加のディスクリプタを使用しなければなりません。返される CLOG レコードの長さに追加バッファの長さを加えた値が、32763 バイト以内である必要があります。これは、CLOG に書き出すことができる最大レコード長です。Adabas ニュークリアスはユーザー出口リターンコード 0 を含むメッセージを出力します。ユーザー出口は無効になり、以降はコールされなくなります。

CLOG レコードが省略される場合、ユーザー出口は、長さを含むレコードの先頭 2 バイトをゼロにセットする必要があります。

pcq： 用途： Adabas コマンドキューエレメント
型： struct cq_entry *
アクセスの種類： 読み込み
値の渡し方： 参照渡し

pcl が 0 ポインタの場合、pcl は UEX_INIT 構造をポイントします（初回コール時）。

Adabas コマンドキューエレメントの構造については、ユーザー出口 1 の説明を参照してください。

pdsc： 用途： バッファディスクリプタ
型： struct cl_dcs *
アクセスの種類： 書き込み
値の渡し方： 参照渡し

pdsc は、構造のアドレスを示します。この構造には、CLOG レコードの追加データが含まれるバッファが記述されます。バッファは長さとポインタで指定されます。このバッファに含まれる情報は、pcl パラメータで指定された CLOG 出力レコードに付加されます。

戻り値

ユーザー出口の戻り値は、基本的に Adabas レスポンスコードです。

ADA_NORMAL： 正常終了。

その他： 失敗。
Adabas ニュークリアスはユーザー出口リターンコードを含むメッセージを出力します。ユーザー出口は無効になり、以降はコールされなくなります。

ADACMP ユーティリティユーザー出口 6

説明

ADACMP ユーティリティユーザー出口 6 は、ADACMP 入力レコード上で圧縮前にユーザー処理を実行するユーザー出口です。このユーザー出口では、レコードの変更やスキップ、追加レコードの挿入、圧縮処理の終了が可能です。実行される処理はユーザー出口の戻り値で示され、この値が ADACMP ユーティリティによって解釈されます。

ユーティリティユーザー出口 6 は、ADACMP の USEREXIT オプションを設定することによりアクティブになります。

入力パラメータ

```
Format: uex_6 ( in_st , out_st )
```

in_st： 用途： ユーザー出口 6 の入力構造
型： struct ue6_in *
アクセスの種類： 読み込み
値の渡し方： 参照渡し

out_sc： 用途： ユーザー出口 6 の出力構造
型： struct ue6_out *
アクセスの種類： 書き込み
値の渡し方： 参照渡し

戻り値

ユーザー出口の戻り値は、出力構造内に返されます。

UE6_O_PROCESS： このレコードを圧縮します。

UE6_O_SKIP： このレコードを圧縮せずにスキップします。

UE6_O_TERM： レコードの圧縮を直ちに終了します。

UE6_O_REPEAT： 同じ入力レコードを使用して再度ユーザー出口をコールします。

上記の定数の定義、構造体 ue6_in、構造体 ue6_out については、インクルードファイル adauex.h を参照してください。

その他の情報

ユーザー出口ルーチンは C で記述する必要があります。ルーチンはダイナミックにロードされます。

ADACMP は、各入力レコードを読み込んだ直後にユーザー出口ルーチンに制御を渡します。ユーザールーチンでは、レコードの修正、拡張、短縮が可能です。また、ADACMP に対して処理されないレコードを示します。ユーザー出口内で作成された追加レコードを ADACMP に渡すこともできます。

入力パラメータブロックへのポインタおよび出力パラメータブロックへのポインタは、各コールで渡されます（詳細については、ヘッダーファイル `adauex.h` を参照してください）。各コールに対する入力レコードエリアの長さおよびアドレスは、ADACMP によって指定されます。入力ファイル内にエンドオブファイル条件が検出された場合、ADACMP は入力ステータスを `UE6_I_EOF` に、入力レコード長を 0 に設定します。ユーザー出口では、出力エリアのアドレスおよび出力レコード長を ADACMP に返す前に、これらを出力パラメータブロックに配置する必要があります。デフォルトでは、ADACMP は入力長と入力エリアおよび出力長と出力エリアを同じ値に設定します。レコードが変更されないように、ユーザールーチンは戻り命令のみを実行する必要があります。レコードが ADACMP による処理対象ではない場合は、出力ステータスを `UE6_O_SKIP` に設定する必要があります。

ユーザー出口では、ADACMP が現在のレコードを処理したら（次のレコードを読み込まずに）、直ちにユーザー出口に制御を渡すように設定することができます。これは、ADACMP に制御が戻る前に、出力ステータスを `UE6_O_REPEAT` に設定することによって行われます。この方法は、ユーザー出口内で作成されたレコードを ADACMP に渡すために使用できます。

出力ステータスでユーザー出口が `UE6_O_TERM` を返した場合、これ以上レコードは処理されません。

```
typedef struct ue6_in
{
    unsigned long        ue6_i_status;
#define UE6_I_NORMAL    1          /* standard call          */
#define UE6_I_EOF       2          /* call after EOF on input */
#define UE6_I_REPEAT    3          /* repeat call on same record */
                                /* because of previous output */
                                /* status UE6_O_REPEAT      */

    unsigned long        ue6_i_len;    /* length of input record  */
    unsigned char*       ue6_i_ptr;    /* pointer to input record  */
} UE6_IN;
```

```
typedef struct ue6_out
{
    unsigned long        ue6_o_status;
#define UE6_O_PROCESS    1          /* process (compress) record */
#define UE6_O_SKIP       2          /* skip this record */
#define UE6_O_TERM       3          /* terminate compression */
#define UE6_O_REPEAT     4          /* call again before reading */
                                   /* next record from input file */

    unsigned long        ue6_o_len;   /* length of output record */
    unsigned char*       ue6_o_rec;   /* pointer to output record */
} UE6_OUT;
```

例

```
#include <adabas.h>
#include <adaux.h>
```

```
#define PERS_ID_OFFSET    0
#define SEX_OFFSET        69
#define FULL_ADDRESS_OFFSET 76
#define ADDRESS_LENGTH    20
#define CITY_COUNTRY_DISPLACEMENT 30
```

```
/*+
**      NAME:
**          uex_6 - adabas user exit 6 example
**
**      SYNOPSIS:
**          int      uex_6 ( )
**
**      DESCRIPTION:
**          This USEREXIT requires uncompressed records
**          of the example file EMPLOYEES as input.
**
**          It changes the personnel-id for all employees
**          coming from Denmark (DK) and United States (USA).
**          The personnel-id is changed in the way that for
**          all female employees from Denmark the personnel-id
**          starts with "40", for all male employees with "41".
**          For all female employees in the United States the
**          personnel-id starts with "20", for all male
**          employees with "21".
**          Additionally all employees from Spain are rejected
**          and therefore those records are not compressed.
**
**      RETURN VALUES:
**          always 0
**
**      FUNCTIONS USED:
**          none.
```

```
/**
- */

#ifdef __STDC__

int uex_6 (struct ue6_in* ue6_in_ptr, struct ue6_out* ue6_out_ptr)

#else

int uex_6 (ue6_in_ptr, ue6_out_ptr)

UE6_IN  *ue6_in_ptr;
UE6_OUT *ue6_out_ptr;

#endif
{

    register unsigned char *country_ptr;
    register unsigned char *field_ptr;
    register unsigned char  mu_field_count;

    if (ue6_in_ptr->ue6_i_status == UE6_I_NORMAL)
    {

        /*
        ** calculate address of country in input record
        */

        field_ptr      = ue6_in_ptr->ue6_i_ptr + FULL_ADDRESS_OFFSET;
        mu_field_count = *field_ptr;
        country_ptr     = field_ptr + 1 + mu_field_count * ADDRESS_LENGTH +
                        CITY_COUNTRY_DISPLACEMENT;

        if (memcmp(country_ptr, "E  ", 3) == 0)
        {
            /*
            ** mark records of spanish employees to be skipped
            */

            ue6_out_ptr->ue6_o_status = UE6_O_SKIP;
        }
        else if ((memcmp(country_ptr, "USA", 3) == 0) ||
                 (memcmp(country_ptr, "DK ", 3) == 0))
        {
            /*
            ** modify personnel id for employees from denmark and USA
            */

            field_ptr = ue6_out_ptr->ue6_o_ptr + PERS_ID_OFFSET;

            if (memcmp(country_ptr, "USA", 3) == 0)
            {
```

```

        *field_ptr = '2';
    }
    else
    {
        *field_ptr = '4';
    }

    if (*(ue6_out_ptr->ue6_o_ptr + SEX_OFFSET) == 'F')
    {
        *(field_ptr + 1) = '0';
    }
    else
    {
        *(field_ptr + 1) = '1';
    }
    }
}
else
{

    /*
    ** signal termination of processing caused by EOF of input file
    */

    ue6_out_ptr->ue6_o_status = UE6_0_TERM;
}
return(0);
}

```

ADAULD ユーティリティユーザー出口7

説明

ADAULD ユーティリティユーザー出口7は、Adabasの圧縮レコードに対するユーザー処理を、ADAULD ユーティリティでレコードがアンロードされる前に実行します。ユーザー出口では、レコードの変更やスキップ、アンロード処理の終了が可能です。実行される処理はユーザー出口の戻り値で示され、この値が ADAULD ユーティリティによって解釈されます。

ユーティリティユーザー出口7は、ADAULD の USEREXIT オプションを設定することによりアクティブになります。

入力パラメータ

Format: uex_7 (in_st, out_st)

in_st： 用途： ユーザー出口 7 の入力構造
型： `struct ue7_in *`
アクセスの種類： 読み込み／書き込み
値の渡し方： 参照渡し

out_st： 用途： ユーザー出口 7 の出力構造
型： `struct ue7_out *`
アクセスの種類： 書き込み
値の渡し方： 参照渡し

戻り値

ユーザー出口の戻り値は、出力ステータスフィールド内に返されます。

UE7_O_PROCESS： このレコードをアンロードします。

UE7_O_SKIP： このレコードをアンロードせずにスキップします。

UE7_O_TERM： レコードのアンロードを直ちに終了します。

上記の定数の定義、構造体 `ue7_in`、構造体 `ue7_out` については、インクルードファイル `adauex.h` を参照してください。

ユーザー出口 7 のパラメータブロック

```
typedef struct ue7_in
{
    unsigned long    ue7_i_len;        /* length of input record */
    unsigned char*   ue7_i_ptr;        /* pointer to input record */
} UE7_IN;

typedef struct ue7_out
{
    unsigned long    ue7_o_status;
#define UE7_O_PROCESS 1                /* process (unload) this record */
#define UE7_O_SKIP    2                /* skip this record */
#define UE7_O_TERM    3                /* terminate unload */
} UE7_OUT;
```

例

```

#include <adabas.h>
#include <adauex.h>

#define FEMALE      'F'
#define MALE        'M'

#define REC_LNG      2      /* Record starts with two byte length field */
#define ISN_LNG      4      /* Next four bytes represent ISN */
#define PERS_ID_LNG  9      /* Next nine bytes represent personnel id */

#define FULL_NAME_OFFSET (REC_LNG + ISN_LNG + PERS_ID_LNG)
                                /* Offset to FULL-NAME group */
#define EMPTY_FIELD_IND 0xC0 /* Indicator for empty NU-field */

/*+
**      NAME:
**          uex_7 - adabas user exit 7 example
**
**      SYNOPSIS:
**          int      uex_7 ( )
**
**      DESCRIPTION:
**          This USEREXIT requires compressed records
**          of the example file EMPLOYEES as input.
**
**          It unloads all records of female employees and skips
**          all other records of male employees. If a value different
**          from 'F' or 'M' is found ADAULD is terminated.

**
**      RETURN VALUES:
**          always 0
**
**      FUNCTIONS USED:
**          none.
**
-*/

#ifdef __STDC__
int uex_7 (struct ue7_in* ue7_in_ptr, struct ue7_out* ue7_out_ptr)
#else
int uex_7 (ue7_in_ptr, ue7_out_ptr)

UE7_IN  *ue7_in_ptr;
UE7_OUT *ue7_out_ptr;

```

```
#endif
{
    register unsigned char *field_ptr;

    /*
    ** skip to first field of FULL-NAME group
    */

    field_ptr = ue7_in_ptr->ue7_i_ptr + FULL_NAME_OFFSET;

    if (*field_ptr & EMPTY_FIELD_IND)
    {
        /*
        ** one empty NU-field (see FDT), skip to NAME-field
        */

        field_ptr += 1;
    }
    else
    {
        /*
        ** it's a length indicator, skip to NAME-field
        */

        field_ptr += *field_ptr;
    }

    field_ptr += *field_ptr;
        /* Add length byte of NAME-field, skip to MIDDLE-NAME-field */

    if (*field_ptr & EMPTY_FIELD_IND)
    {
        /*
        ** one empty NU-field (see FDT), skip to MARRIAGE-STATE-field
        */

        field_ptr += 1;
    }
    else
    {
        /*
        ** it's a length indicator, skip to MARRIAGE-STATE-field
        */

        field_ptr += *field_ptr;
    }

    field_ptr += 1;                                /* Skip to SEX-field */

    if (field_ptr < (ue7_in_ptr->ue7_i_ptr + ue7_in_ptr->ue7_i_len))
    {
        if (*field_ptr == FEMALE)
```

```

    {
        return(0);                                /* Female employee, unload record */
    }
    else if (*field_ptr == MALE)
    {
        ue7_out_ptr->ue7_o_status = UE7_O_SKIP;
        return(0);                                /* Male employee, skip this record */
    }
}

/*
** something is wrong, terminate ADAULD
*/

ue7_out_ptr->ue7_o_status = UE7_O_TERM;
return(0);
}

```

ニュークリアスユーザー出口 11

説明

Adabas ニュークリアスユーザー出口 11 は、Adabas ダイレクトコール上のユーザー処理を実行するユーザー出口です。ルーチンは、コマンド処理の開始時にコールされます。指定された入力パラメータによって、ユーザー出口による Adabas コールのパラメータの変更やコールの拒否が可能になります。コールが拒否された場合は、そのコールを発行したユーザーに Adabas レスポンスコード 22（無効なコマンド）が返されます。

このユーザー出口の機能は、ユーザー出口 1 の機能と同じです。ただし、ユーザー出口 11 は新しいバージョン 6 の Adabas 構造を使用しています。

ユーザー出口では、コマンドコードを変更したり、Adabas コントロールブロックで指定されたバッファ長を変更したりすることはできません。これらの値を変更すると、コマンドを発行したユーザーに Adabas レスポンスコード 22（無効なコマンド）が返されます。

ニュークリアスユーザー出口 11 は、ADANUC で USEREXITS=11 と設定することによりアクティブになります。

入力パラメータ

Format: uex_11 (pacbx, pcb, pcq6, num_abd, patb_abd)

pacbx：用途：新しい Adabas コントロールブロック
型：struct adacbx *
アクセスの種類：読み込み／書き込み
値の渡し方：参照渡し

Adabas コントロールブロックの構造については、『*Adabas コマンドリファレンスマニュアル*』を参照してください。

pcb：用途：古い Adabas コントロールブロック
型：struct cb_par *
アクセスの種類：読み込み／書き込み
値の渡し方：参照渡し

Adabas コントロールブロックの構造については、『*Adabas コマンドリファレンスマニュアル*』を参照してください。

pcq6：用途：Adabas コマンドキューエレメント V6
型：struct v6_cq_entry *
アクセスの種類：読み込み
値の渡し方：参照渡し

Adabas コマンドキューエレメントの詳細については、ヘッダーファイル `adauex.h` を参照してください。

num_abd：用途：Adabas バッファディスクリプタ (ABD) の数
型：int
アクセスの種類：読み込み／書き込み
値の渡し方：値渡し

patb_abd：用途：ABD リストへのポインタ
型：char *
アクセスの種類：読み込み／書き込み
値の渡し方：参照渡し

ACBX、ABD および ABD リストの構造については、『*Adabas コマンドリファレンスマニュアル*』を参照してください。

戻り値

ユーザー出口戻り値は、基本的に Adabas レスポンスコードです。

ADA_NORMAL : 成功。

ニュークリアスは、コントロールブロック内で無効な変更が行われていないかどうかチェックします。無効な変更が検出されなかった場合、このコールは処理されます。

その他 :

失敗。

Adabas コールは拒否され、レスポンスコード 22 (無効なコマンド) が返されます。

ニュークリアスユーザー出口 14

説明

Adabas ニュークリアスユーザー出口 14 は、バージョン 6 レイアウトの CLOG 出力レコードに対するユーザー処理を、レコードが CLOG ファイルに書き込まれる前に実行します。このユーザー出口では、レコードの短縮、拡張、および完全な変更が可能です。レコードの長さまたは構造が変更された場合、新しいタイプの CLOG を出力するために使用する PRILOGC ユーティリティは、作成されたレコードを出力できなくなる場合があります。CLOG レコードが書き込まれたときにユーザー出口がアクティブな場合、警告メッセージが発行されます。

ニュークリアスユーザー出口 14 は、ADANUC で USEREXITS=14 と設定することによりアクティブになります。

入力パラメータ

Format: uex_14 (pclx, pbuf, pcq6)

pclx : 用途 : CLOG レイアウト 6 出力レコード

型 : struct clx_entry *

アクセスの種類 : 読み込み／書き込み

値の渡し方 : 参照渡し

CLOG レイアウト 6 レコードの構造については、adauex.h に含まれる CLX_ENTRY の構造体定義を参照してください。

CLOG レコードの先頭 2 バイトにはレコード長が含まれます。PRILOGC の情報が含まれる 3 バイト目および 4 バイト目は、変更しないでください。この 2 バイトは、ユーザー出口ルーチンへのコール後に、ニュークリアスによって以前の状態に戻されます。

ユーザー出口入力レコードの長さは、拡張しないでください。ユーザー出口から追加のデータを返すには、追加のディスクリプタを使用しなければなりません。返される CLOG レコードの長さに追加バッファの長さを加えた値が、32763 バイト以内である必要があります。これは、CLOG に書き出すことができる最大レコード長です。Adabas ニュークリアスはユーザー出口リターンコード 0 を含むメッセージを出力します。ユーザー出口は無効になり、以降はコールされなくなります。

CLOG レコードが省略される場合、ユーザー出口は、長さを含むレコードの先頭 2 バイトをゼロにセットする必要があります。

pclx が 0 ポインタの場合、pclx は UEX_INIT 構造体をポイントします（初回コール時）。

pbuf：用途： バッファポインタ
型： char *
アクセスの種類： 書き込み
値の渡し方： 参照渡し

pdsc は、構造のアドレスを示します。この構造には、CLOG レコードの追加データが含まれるバッファが記述されます。バッファは長さで指定されます。このバッファに含まれる情報は、pcl パラメータで指定された CLOG 出力レコードに付加されます。

pcq6：用途： Adabas コマンドキューエレメント V6
型： struct v6_cq_entry *
アクセスの種類： 読み込み
値の渡し方： 参照渡し

Adabas コマンドキューエレメントの詳細については、ヘッダーファイル adauex.h を参照してください。

戻り値

ユーザー出口戻り値は、基本的に Adabas レスポンスコードです。

ADA_NORMAL： 正常終了。

その他： 失敗。
Adabas ニュークリアスはユーザー出口リターンコードを含むメッセージを出力します。ユーザー出口は無効になり、以降はコールされなくなります。

ニュークリアスユーザー出口 21

説明

Adabas ニュークリアスユーザー出口 21 を使用し、ADABAS サーバー API 機能を介して認証資格情報を設定することができます。ルーチンは、セッション処理の開始時にコールされます。

このルーチンはできる限り短期間で使用する必要があります。これは、すべてのアプリケーションが Adabas セキュリティの認証機能を使用およびサポートするまでの、移行期間中の使用を目的としています。

指定された入力パラメータによって、ユーザー出口は、Adabas コールパラメータを分析してコール元のアプリケーションを識別し、適切な資格情報を設定するかコールを拒否します。コールが拒否された場合は、そのコールを発行したユーザーに Adabas レスポンスコード 200（セキュリティ違反）が返されます。

ニュークリアスユーザー出口 21 は、ADANUC で USEREXITS=21 と設定し、環境変数 ADAUEX_21 を設定すれば、アクティブになります。

入力パラメータ

```
Format: int uex_21 (uex21, uex_init)
```

uex21： 用途： オブジェクトの処理
 型： struct SECUEXStruct *
 アクセスの種類： 読み込み
 値の渡し方： 参照渡し

Adabas コントロールブロックの構造については、『Adabas コマンドリファレンスマニュアル』を参照してください。

uex_init： 用途： 初期化インジケータ
 型： struct uex_init *
 アクセスの種類： 読み込み／書き込み
 値の渡し方： 参照渡し

戻り値

ユーザー出口の戻り値で、認証処理の進行方法が決まります。

SECUEX_SUCCESS: 成功。
 認証処理は、提供された資格情報で続行されます。

SECUEX_FAILED: 失敗。
 Adabas コールは拒否され、レスポンスコード 200（セキュリティ違反）が返されます。

ユーザー出口 **21** のパラメータブロック

```
struct SECUEXStruct {
    SECUEXPrivate * privatedata; /* For Internal Use Only */

    unsigned int  secdbid; /* database id */
    FNR          secfnr; /* file number */
    UQID         secuqid; /* s-node, s-user, s-tid */

    int (*set_uid_psw) (SECUEX * su, char * uid, char * psw);
    int (*get_acbx)    (SECUEX * su, ACBX * acbx);
    int (*is_natural)  (SECUEX * su);
    int (*is_sql_gateway)(SECUEX * su);
};
```

関数／メソッド

ユーザー出口には、Adabas セッションの識別を可能にする次の機能があります。

- `*set_uid_psw` セッションの資格情報を設定します。
- `*get_acbx` セッションの ACBX コントロールブロックを取得します。
- `*is_natural` コールが Natural アプリケーションから発行されたかどうかを判断します。
- `*is_sql_gateway` コールが SQL ゲートウェイアプリケーションによって発行されたかどうかを判断します。

機能	<code>*set_uid_psw</code>	セッションの資格情報を設定します。
パラメータ	<code>SECUEX * su</code> <code>char * uid</code> <code>char * psw</code>	オブジェクトの処理 ユーザー ID の参照 パスワードの参照
戻り値	<code>SECUEX_SUCCESS</code> <code>SECUEX_INVALID_PARAM</code> <code>SECUEX_INVALID_INTERNAL</code> <code>SECUEX_INVALID_HEADER</code> <code>SECUEX_BUFFER_OVERFLOW</code>	機能は正常に終了しました。 オブジェクトの処理が無効または欠落しています。 内部エラー – 予期しない値 内部エラー - 無効なセキュリティバッファ 内部エラー - セキュリティバッファオーバーフロー

機能	<code>*get_acbx</code>	セッションの ACBX コントロールブロックを取得します。
パラメータ	<code>SECUEX * su</code> <code>char * acbx</code>	オブジェクトの処理 <code>struct ACBX</code> の参照
戻り値	<code>SECUEX_SUCCESS</code> <code>SECUEX_INVALID_PARAM</code> <code>SECUEX_INVALID_INTERNAL</code>	機能は正常に終了しました。 オブジェクトの処理が無効または欠落しています。 内部エラー – 予期しない値

Adabas コントロールブロックの構造については、『コマンドリファレンス』ドキュメントを参照してください。

機能	<code>*is_natural</code>	コールが Natural アプリケーションから発行されたかどうかを判断します。
パラメータ	<code>SECUEX * su</code>	オブジェクトの処理
戻り値	<code>SECUEX_TRUE</code> <code>SECUEX_FALSE</code> <code>SECUEX_INVALID_PARAM</code> <code>SECUEX_INVALID_INTERNAL</code>	コール元のアプリケーションは Natural です。 コール元のアプリケーションは Natural ではありません。 オブジェクトの処理が無効または欠落しています。 内部エラー – 予期しない値

機能	*is_sql_gateway	コールが SQL ゲートウェイアプリケーションによって発行されたかどうかを判断します。
パラメータ	SECUEX * su	オブジェクトの処理
戻り値	SECUEX_TRUE	コール元のアプリケーションは SQL ゲートウェイです。
	SECUEX_FALSE	コール元のアプリケーションは SQL ゲートウェイではありません。
	SECUEX_INVALID_PARAM	オブジェクトの処理が無効または欠落しています。
	SECUEX_INVALID_INTERNAL	内部エラー – 予想しない値

例

```
#include <adauex.h>

/*
** NAME:
**     uex_21 - Adabas Security Exit Example
**
** SYNOPSIS:
**     int uex_21 (
**         SECUEX *   Pointer to SECUEX Data and Methods
**         UEX_INIT *   Pointer to UEX Initialization Mode
**
** DESCRIPTION:
**     This user exit is called before an Adabas call is processed.
**
**     It provides the Security credentials,
**     which are to be used for authentication in this session.
**
**     Because no global variables are used, this exit flags
**     in init call to inform the Adabas nucleus that reentrant
**     code is being used.
**
**     To create a shared library that includes this user exit,
**     the makefile can be used.
**
**     Call:          make -f makefile uex21    (Unix)
**                   nmake -f makefile uex21    (Windows)
**
** RETURN VALUES:
**
**     SECUEX_SUCCESS
**     SECUEX_FAILED
**/

#ifdef __STDC__

int uex_21 (SECUEX * uex21,
            UEX_INIT * uex_init)
```

```
#else

int uex_21 (uex21,
            uex_init)
SECUEX      * uex21;
UEX_INIT    * uex_init;

#endif

{
    int rc;
    if (uex21 == 0)
    {
        /*
         ** User-Exit Initialization during Nucleus Startup
         **
         ** Indicate whether the user-exit is reentrant or not
         ** - UEX_REENTRANT
         ** - UEX_N_REENTRANT
         */
        uex_init->uex_type = UEX_REENTRANT;

        return ( SECUEX_SUCCESS );
    }

    else

    {
        /*
         ** AuthN Processing - during Session Initialization
         **
         ** Provide security credentials or not
         ** This can be based on the values in ACBX and UQID values
         ** which can be retrieved as needed (optional)
         **
         ** Note:
         ** Not providing security credentials
         ** will result a security violation "unable to authenticate".
         **
         */
        ACBX  acbx;
        char uid[32];
        char psw[32];

        /*
         ** Retrieve Session-Specific information
         */

        /* ACBX values */
        rc = uex21->get_acbx (uex21, &acbx);
        if (rc != SECUEX_SUCCESS)
        {
```

```
    return( SECUEX_FAILED );
}

/*
** Reject commands from users with names starting with 'h'
*/
if ( uex21->secuqid.tid[0] == 'h' )
{
    return( SECUEX_FAILED );
}

/*
** Reject access to dbid=200
** where users names start with 'h'
*/
if ((acbx.acbxdbid == 200 ) && (uex21->secuqid.tid[0] == 'h'))
{
    return( SECUEX_FAILED );
}

/*
** Set Application-Specific credentials
*/
strcpy (uid,"uexuid");
strcpy (psw,"uexpsw");

/* NATURAL */
rc = uex21->is_natural (uex21);
if (rc == SECUEX_TRUE)
{
    strcpy (uid,"NATuid");
    strcpy (psw,"NATpsw");
}

/* SQL Gateway */
rc = uex21->is_sql_gateway (uex21);
if (rc == SECUEX_TRUE)
{
    strcpy (uid,"SQLuid");
    strcpy (psw,"SQLpsw");
}

/*
** Set Security Credentials: userid, password
*/
rc = uex21->set_uid_psw (uex21, uid, psw);
if (rc != SECUEX_SUCCESS)
{
    return( SECUEX_FAILED );
}
}
```

```
return ( SECUEX_SUCCESS );          /* provided Security Credentials */  
}
```

ADALNK 指定ユーザー出口

概要

ADALNKのユーザー出口は、実行時に制御を要求したり、コントロールブロックなどのAdabasパラメータを変更したりするなどのアプリケーションにも使用できます。

このユーザー出口は、ニュークリアスユーザー出口1と同様に、パラメータとしてAdabasバッファを指定して呼び出します。ユーザー出口は、Adabas コールがニュークリアスに渡される前とAdabas コールが実行された後に有効になります。ADALNK 指定ユーザー出口は、ユーザー処理のコンテキストで実行する点がニュークリアスユーザー出口1と異なります。

ADALNK 固有のユーザー出口の実装形態は、共有ライブラリ（UNIX）またはダイナミックリンクライブラリ（Windows）です。ADALNK固有のユーザー出口のコンパイル方法とリンク方法については、「ユーザー出口およびハイパー出口の作成と定義」を参照してください。

シグナルハンドラについての注意（UNIX）

Adabas コールがアプリケーション定義のシグナルハンドラで使用され、ADALNK ユーザー出口が確立された場合に、シグナルハンドラからのコールに対するユーザー出口が開始されない場合があります。この状況は、シグナルハンドラからのコールの発生時にユーザー出口の別のコールがアクティブになっている場合に発生します。

Adabas コールがアクティブな状態でシグナルハンドラがユーザーアプリケーションを終了した場合、次の状況が発生する可能性があります。

1. ユーザー出口0がすでにコールされているが、Adabas コールがニュークリアスに渡されていない。
2. ニュークリアスは Adabas コールの結果を返すが、ユーザー出口1はコールされていない。

例外ハンドラについての注意（Windows）

Adabas コールがアプリケーション定義の例外ハンドラで使用され、ADALNK ユーザー出口が確立された場合に、例外ハンドラからのコールに対するユーザー出口が開始されない場合があります。この状況は、例外ハンドラからのコールの発生時にユーザー出口の別のコールがアクティブになっている場合に発生します。

Adabas コールがアクティブな状態で例外ハンドラがユーザーアプリケーションを終了した場合、次の状況が発生する可能性があります。

1. ユーザー出口0がすでにコールされたが、Adabas コールがニュークリアスに渡されない。
2. ニュークリアスは Adabas コールの結果を返したが、ユーザー出口1がコールされない。

入力パラメータ

```
Format: int lnkuex_{0 | 1} (cb,fb,rb,sb,vb,ib)
```

入力パラメータについては、[ニュークリアスユーザー出口 1](#) を参照してください。

戻り値

ユーザー出口の戻り値は、基本的に Adabas レスポンスコードです。

ADA_NORMAL： 正常終了。

その他： 失敗。
レスポンスコードは Adabas コントロールブロックに格納されます。lnkuex_0 の場合、コールが直ちにユーザーに返される。RC コールの場合、レスポンスコードが 0 に設定される。

ADALNK 指定ユーザー出口と ADALNK 間のリンクの作成

ADALNK に相当するユーザー出口と ADALNK 自体をリンクするには、次の環境変数が設定されている必要があります。

■ データベースアクセス前の ADALNK 内のユーザー出口

```
setenv LNKUEX_0 lnk_uex.sl (UNIX)
```

または

```
set LNKUEX_0=lnk_uex.dll (Windows)
```



注意: デフォルトのエントリ関数名は 'lnkuex_0(...)' です。

■ データベースアクセス後の ADALNK 内のユーザー出口

```
setenv LNKUEX_1 lnk_uex.sl (UNIX)
```

または

```
set LNKUEX_1=lnk_uex.dll (Windows)
```



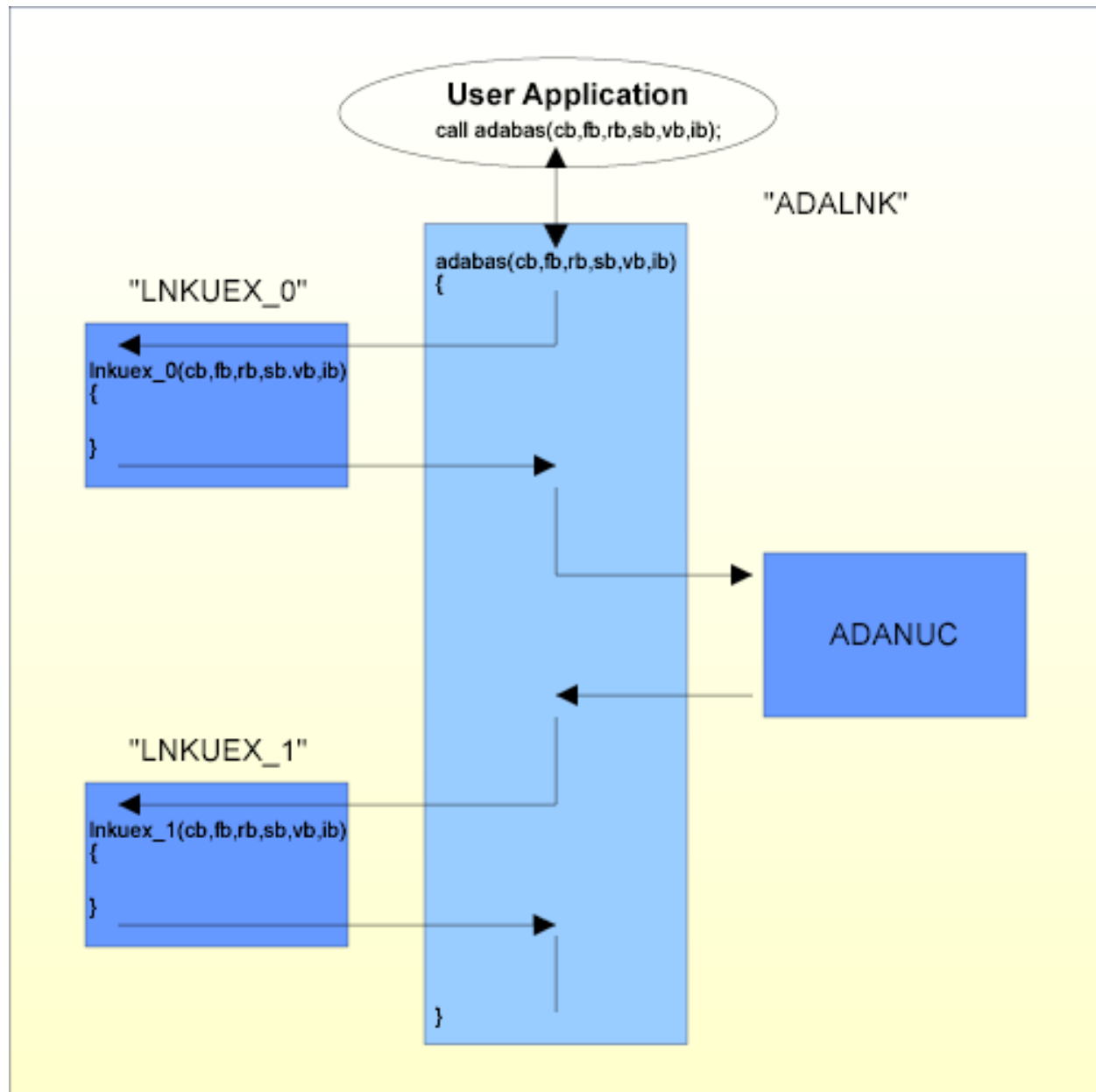
注意: デフォルトのエントリ関数名は 'lnkuex_1(...)' です。

次のように、各 ADALNK 指定ユーザー出口に個別のエントリ関数を定義することもできます。

```
setenv LNKUEX_0 "lnk_uex.sl xx_uex_0" (UNIX)
setenv LNKUEX_1 "lnk_uex.sl yy_uex_1"
```

または

```
set LNKUEX_0=lnk_uex.dll xx_uex_0 (Windows)
set LNKUEX_1=lnk_uex.dll yy_uex_1
```



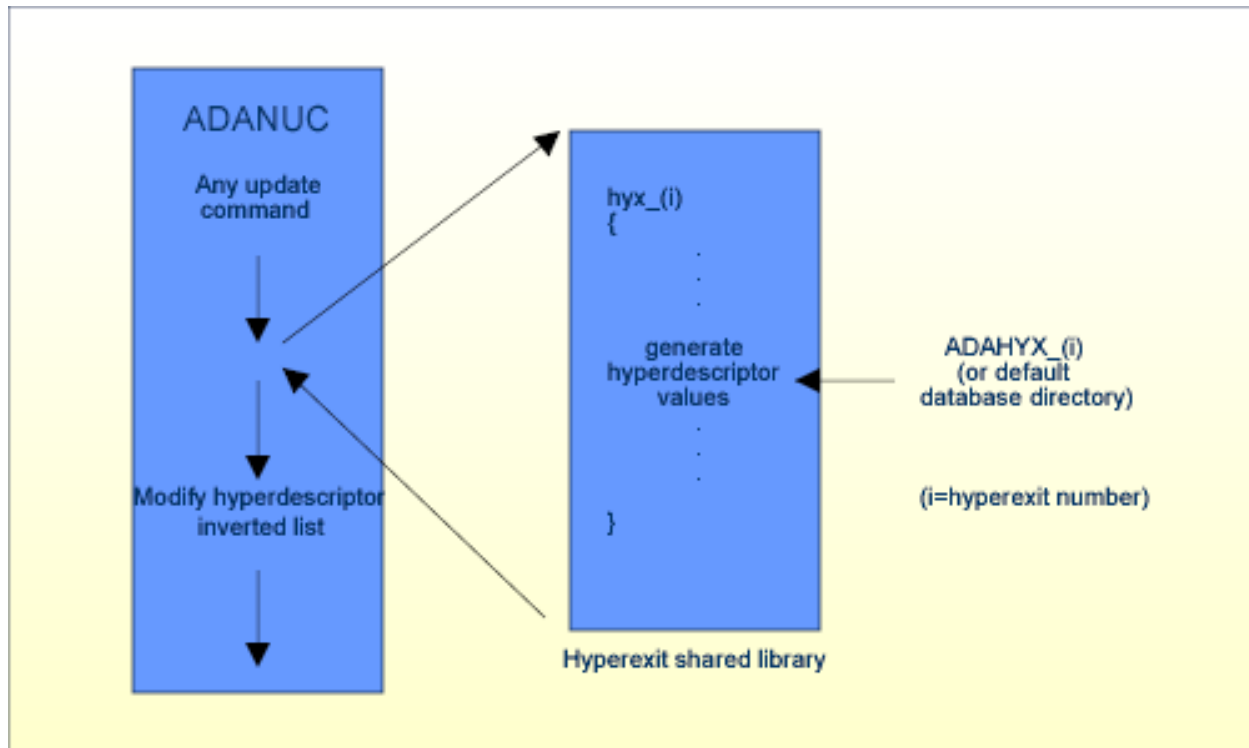
ハイパー出口の概要

サブディスクリプタやスーパーディスクリプタと同様に、ハイパーディスクリプタはファイルの FDT 内の親フィールドに基づいて作成されます。ただし、ハイパーディスクリプタのディスクリプタ値は、ユーザー定義のアルゴリズムを使用して作成されます。検索条件では、生成されたハイパーディスクリプタ値を指定します。ただし、ハイパーディスクリプタが HE オプションで定義されている場合を除きます。この場合は、親フィールドの値を指定します。

ハイパー出口は、ハイパーディスクリプタ値の ISN を変更できます。別のファイルに属する ISN も指定できます。このようにすれば、フォーマットバッファを使用せずに、ハイパーディスクリプタを含むファイルで検索を実行できます。結果のレコードを読み取るには、ファイル番号を、ハイパーディスクリプタに関連付けられた ISN が含まれているファイルのファイル番号に変更する必要があります。この機能を使用する際に、そのようなファイルをロードするときに、ADACMP と ADAMUP に USERISN パラメータを指定することが重要です。指定しないと、ハイパー出口で生成された ISN が適合しなくなります。

このアルゴリズムはハイパー出口と呼ばれるユーザー定義の共有ライブラリまたは DLL に実装されています。C プログラミング言語で記述する必要があります。ハイパー出口機能の基本出力は、1 つ以上のハイパーディスクリプタ値です。

ハイパー出口は、ユーティリティの起動時またはニュークリアスで初めてアクセスされるときに動的にロードされます。ハイパーディスクリプタ値が生成、削除、更新されるたびに呼び出されます。また、HE オプションが指定されたハイパーディスクリプタが、検索コマンドで使用された場合にも呼び出されます。



ハイパー出口は、次の処置を行うためにコールされます。

- ハイパー出口の初期化。下記にリストされた処理の1つのためにニュークリアスによって初めて使用される前、またはハイパーディスクリプタがファイルのFDTに存在する場合のユーティリティの実行中に行われます。
- レコード挿入コマンド：ハイパー出口は「アフターイメージコール」としてコールされ、インバーテッドリストに挿入されるハイパーディスクリプタ値を生成します。FDTに定義されているすべてのハイパーディスクリプタに対して、適切な出口関数がコールされます。

 **注意:** ハイパーディスクリプタのすべての親フィールドがNU/NCオプションで定義されている場合やこれらのフィールドに値が存在しない場合、出口関数はコールされません。

- レコード削除コマンド：ハイパー出口は「ビフォーイメージコール」としてコールされ、インバーテッドリストから削除されるハイパーディスクリプタ値を生成します。FDTに定義されているすべてのハイパーディスクリプタに対して、適切な出口関数がコールされます。
- レコード更新コマンド：1つ以上のハイパーディスクリプタの親フィールドが変更された場合、インバーテッドリスト内のハイパーディスクリプタ値を変更するために、関連するハイパー出口が2回コールされます（「ビフォーイメージコール」として1回、「アフターイメージコール」として1回）。
- 検索コマンド用のハイパーディスクリプタ値の生成（1つの親フィールドを持ち、HEオプションで定義されたハイパーディスクリプタのみ）。

ハイパーディスクリプタは、次のユーティリティで使用されます。

- ADAINV で次の処理を実行する場合：
 - 新しいハイパーディスクリプタのインバート
 - ハイパーディスクリプタの再インバート／確認
- ADACMP：FDTにハイパーディスクリプタの指定が含まれる場合のハイパーディスクリプタ値の生成
- ADAULD：FDTにハイパーディスクリプタの指定が含まれる場合のハイパーディスクリプタ値の生成

ハイパー出口は、"アフターイメージ" コールと "ビフォーイメージ" コールに同じ値を生成することが重要です。そうしないと、Adabasはハイパーディスクリプタ値を削除できません。つまり、ハイパーディスクリプタ値が変化するような変更をハイパー出口に加えた後は、影響を受けるハイパーディスクリプタを再インバートする必要があります。

ハイパー出口は、リエントラントインターフェイスまたは非リエントラントインターフェイスを使用して操作できます。非リエントラントインターフェイスの場合は、親フィールド値および生成されたハイパーディスクリプタ値のハイパー出口バッファが、ユーザーのハイパー出口モジュール内に配置されます。この場合、ハイパー出口へのコールは連続します。

リエントラントインターフェイスの場合、バッファはニュークリアスまたはユーティリティ内に配置されます。したがって、複数のコマンドが同じハイパー出口にアクセスしようとしたときに、リエントラントハイパー出口を並行して呼び出すことができます。これにより、非リエントラントバージョンと比較してパフォーマンスが向上します。リエントラントハイパー出口の実装では、ストレージクラススタティックの変数は、初期化中にのみ更新できます。



ヒント: Adabas キットには、ハイパー出口用のサンプル C ファイルが含まれています。これらは、インストールディレクトリのサブディレクトリ「Adabas/examples/server」にあります。

FDT でハイパーディスクリプタを指定する方法の詳細については、[「FDT のレコード構造」の「ハイパーディスクリプタ」](#)を参照してください。

ハイパー出口コントロールブロックとバッファ

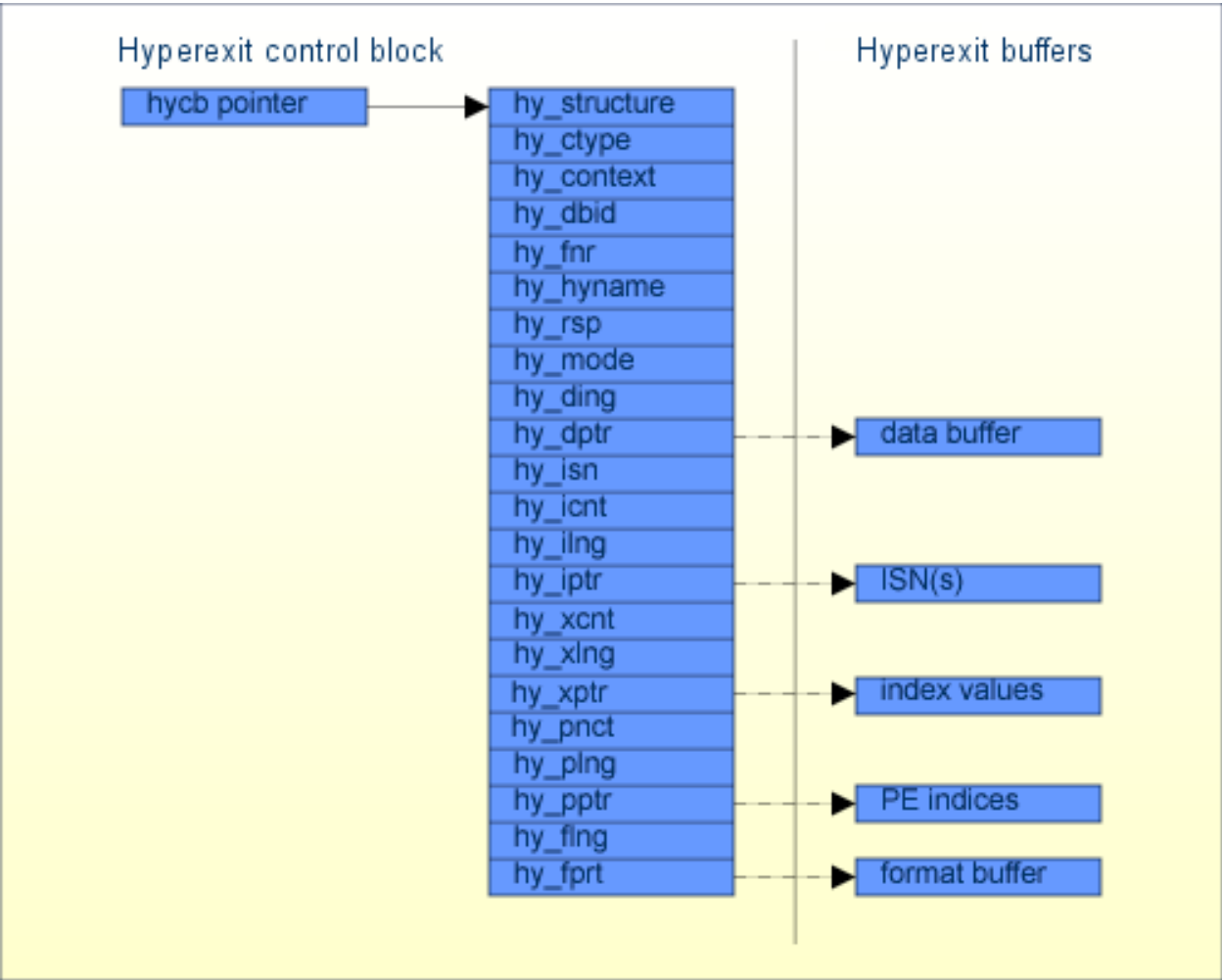
ハイパー出口機能の概要を次に示します。

```
#include <adahyx.h>

void hyx_<number> ( HYCB_ENTRY* hy_cb )
```

hy_cb は、ハイパー出口コントロールブロックへのポインタです。この C 定義は、インストールディレクトリのサブディレクトリ「Adabas/inc」にあるヘッダーファイル adahyx.h 内にあります。

ハイパー出口コントロールブロックには、ハイパー出口およびニュークリアスがハイパーディスクリプタを扱うために必要な情報がすべて含まれています。次の図にその構造を示します。



ハイパー出口コントロールブロック

変数	型	説明
hy_structure	符号なしの char	ハイパーディスクリプタインターフェイスの構造レベルを定義します。 将来的に構造が変更される可能性があります。そのため、Adabasは、サポートされているバージョンのハイパーディスクリプタインターフェイスが使用されているかどうかを確認できます。実際の構造レベルは2です。現在サポートされている他の構造レベルはありません。
hy_ctype	符号なしの short	実行されるコールのタイプを定義します。 0 値の生成コール 1 値の生成コール繰り返し 2 初期化コール 3 初期化コールの繰り返し

変数	型	説明
		ユーティリティの初期化中、またはニュークリアスで初めてハイパーディスクリプタが処理されるときに、初期化コールが実行されます。 ハイパー出口で親フィールドとハイパーディスクリプタの定義に関する詳細情報が必要な場合に、ハイパー出口レスポンスコード <code>hy_rsp</code> を 1 に設定できます。Adabas が初期化コールを繰り返し実行します。 Adabas が親値をハイパーディスクリプタ値に変換する必要がある場合は、値生成コールを実行します。複数のハイパーディスクリプタ値を生成する場合、ハイパー出口は複数の呼び出しで値を提供できます。追加の値を生成する場合は、ハイパー出口で <code>hy_rsp</code> を 1 に設定する必要があります。このようにすれば、Adabas は値生成コールを繰り返し実行します。
<code>hy_context</code>	符号なしの short	値生成コールのコンテキストを定義します。 0 ビフォーイメージ値 1 アフターイメージ値 2 サーチバッファ値 ビフォーイメージ値は、ディスクリプタ値をインデックスから削除する必要がある場合に使用されます。 アフターイメージ値は、ディスクリプタ値をインデックスに挿入する必要がある場合に使用されます。 サーチバッファ値は、検索処理の検索条件で HE オプションを使用するハイパーディスクリプタが指定されている場合に使用されます。
<code>hy_dbid</code>	符号なしの short	ハイパー出口をコールしたデータベースを定義します。
<code>hy_fnr</code>	符号なしの short	ハイパーディスクリプタのファイル番号を定義します。
<code>hy_hyname</code>	char[2]	ハイパーディスクリプタの名前を定義します。
<code>hy_rsp</code>	符号なしの short	このレスポンスによって、ハイパーディスクリプタのハイパー出口は、正常終了コードやエラーレスポンスコードを返すことができます。エラーは Adabas ニュークリアスのレスポンスコード 86 に変換され、ハイパー出口の値はコントロールブロックのアディション 2 フィールドに返されます。初期化コールの間、ユーティリティやニュークリアスはエラーで終了します。 レスポンスコードとその意味は次のとおりです。 0 正常終了 1 繰り返しコールが必要です 2～255 Adabas 用に予約済み >=256 ユーザーエラー

変数	型	説明
hy_mode	符号なしの short	これは、初期化コールにより設定されます。出口ルーチンがリエントラントか非リエントラントかを定義します（詳細については、「リエントラント／非リエントラントインターフェイス」を参照してください）。 0 (HY_NREENTRANT)：非リエントラント 1 (HY_REENTRANT)：リエントラント
hy_dptra	unsigned char *	データバッファポインタ。
hy_dlng	符号なしの int	データバッファ長サポートされる最大長は、65535 です。
hy_isn		データレコードの ISN です。ハイパーディスクリプタ値の ISN は変更可能です。Adabas は、ISN がファイルに指定された MAXISN の値以下であるかどうかについてはチェックしません。
hy_iptra	符号なしの int *	ISN バッファポインタ。
hy_ilng	符号なしの int	ISN バッファ長。サポートされる最大長は、65535 です。
hy_icnt	符号なしの int	ISN 値の個数。この数の ISN が ISN バッファに収まる必要があります。 4 * hy_icnt は hy_ilng 以下にする必要があります。
hy_xptr	unsigned char *	インデックスバリュースタックポインタ。
hy_xcnt	符号なしの int	返されたインデックス値の数。
hy_xlng	符号なしの int	インデックスバリュースタック長。
hy_pcna	符号なしの int	PE インデックスバッファで返される PE インデックスの数です。ハイパーディスクリプタがオプション PE で定義されている場合、この値は hy_xcnt と等しくなる必要があります。ただし、サーチバッファ値のコールでは、PE インデックスバッファは無視されます。
hy_plng	符号なしの int	PE インデックスバッファ長。
hy_pptra	unsigned char *	PE インデックスバッファポインタ。
hy_fpra	unsigned char *	フォーマットバッファポインタ。 初期化コールの間、ハイパー出口はフォーマットバッファを指定することができます。このフォーマットバッファは、ハイパー出口がコールされる前に、親フィールドのデータ値を圧縮解除してバッファに格納するために使用されます。ユーザーは、フォーマットバッファを使用してデフォルトのフォーマットバッファをオーバーライドすることができます。
hy_flg	符号なしの int	フォーマットバッファ長。

初期化コールの繰り返し用のデータバッファ

初期化コールの繰り返しでは、Adabas はデータバッファで親フィールドとハイパーディスクリプタに関する情報を提供します。この情報のレイアウトは、コマンドオプション S を使用した LF コマンドのレコードバッファと同じです（詳細については『コマンドリファレンス』の「LF コマンド」を参照してください）。

バイト数	説明
1～2	情報の合計長
3～4	エントリ数
5～n	各親フィールドの 1 つの F エlement
(n+1) ～m	ハイパーディスクリプタの H エlement

値生成コール用のデータバッファ

値生成コール／値生成コールの繰り返し用のデータバッファでは、Adabas は、初期化コール／初期化コールの繰り返しのフォーマットバッファ仕様に従って、ハイパーディスクリプタを挿入します。

ISN バッファ

値生成コール／値生成コールの繰り返しでは、ハイパーディスクリプタ値に関連付けられている ISN を定義する方法が 2 つあります。

1. このコールで生成されたすべてのハイパーディスクリプタ値に、同じ ISN が使用されます。この場合、ISN は hy_isn で指定する必要がある、hy_icnt を 0 にする必要があります。
2. この呼び出しで生成されたハイパーディスクリプタ値には、異なる ISN が使用されます。この場合、コールによって生成された各ディスクリプタ値について、ISN バッファに ISN が挿入されている必要があります。hy_icnt が hy_xcnt と等しくなっている必要があります。

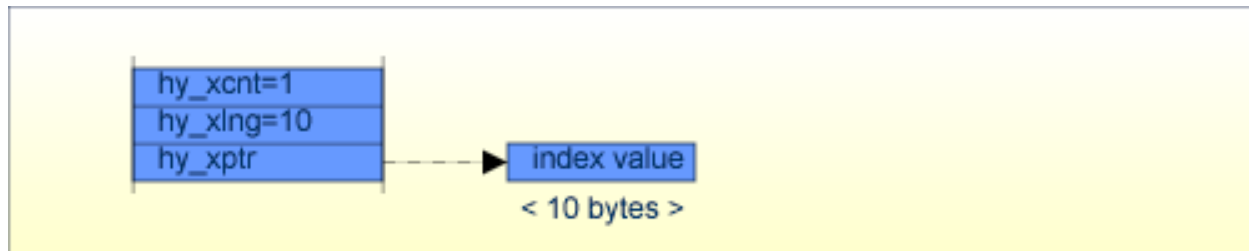
インデックスバリュースバッファ

値生成コール／値生成コールの繰り返しでは、ハイパー出口がハイパーディスクリプタ値を生成する必要があります。hy_xcnt には、生成されたハイパーディスクリプタ値の数を設定する必要があります。ハイパーディスクリプタ値は、ハイパーディスクリプタ定義で指定されたフォーマットで生成される必要があります。

以下の例に、インデックスバッファの生成方法を示します。

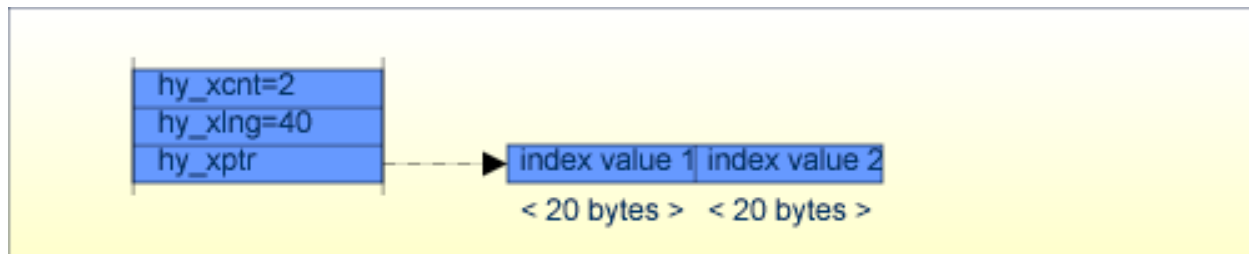
例 1：

```
H1,10,A=HYPER(1,AA)
```



例 2：

```
H2,20,A,MU=HYPER(1,MU)
```



2つのハイパーディスクリプタ値を生成するには、インデックスバリュースタックの長さを少なくとも 40 バイトにする必要があります。生成されるハイパーディスクリプタ値の最大数がより大きい場合は、それに応じて大きいインデックスバリュースタックを定義する必要があります。

PE インデックスバッファ

ハイパーディスクリプタがオプション PE で定義されている場合、ハイパーディスクリプタは、値生成コール／値生成コールの繰り返しの PE インデックスバッファにあるハイパーディスクリプタ値に対応する、PE インデックスを生成する必要があります。PE インデックスの数は、生成されるハイパーディスクリプタ値の数と同じになっている必要があります。PE インデックスは 1 バイトのバイナリとして保存されます。255 より大きい PE インデックスはサポートされません。

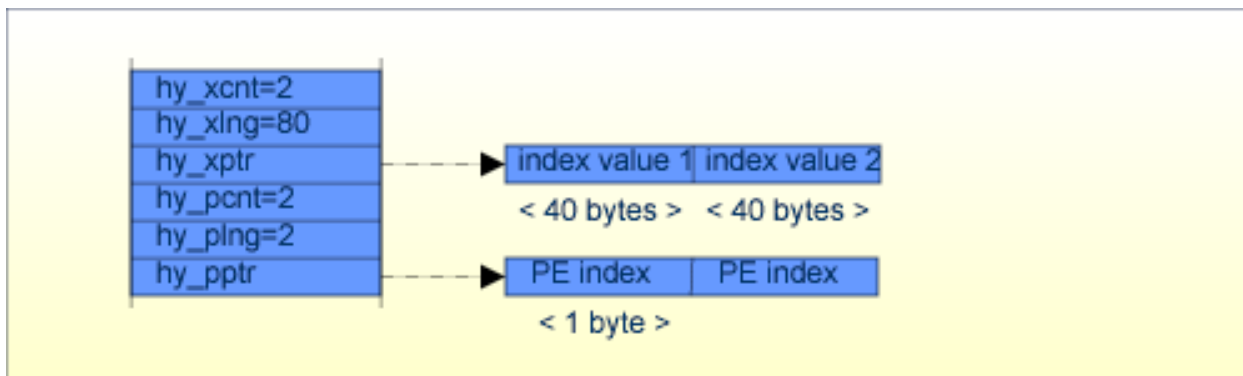
ただし、サーチバッファ値コールに対しては、ハイパー出口は PE インデックスを提供できません。このような場合は、ユーザーのサーチバッファの PE インデックスが付加されます。

パック値およびアンパック値の整合性チェックが行われます。可変長の値については、この値のフォーマットで指定できる最大の長さ以内であるかどうかチェックされます。

次の例に、オプション PE で定義されたハイパーディスクリプタに対して、どのようにインデックスバリュースタックと PE インデックスバッファが生成される必要があるかを示します。

例：

H3,40,A,PE=HYPER(1,P1)



2つのハイパーディスクリプタ値を生成するには、インデックスバリュースバッファの長さを少なくとも80バイトにし、PEインデックスバッファの長さを少なくとも2にする必要があります。生成されるハイパーディスクリプタ値の最大数がより大きい場合は、それに応じて大きいインデックスバリュースバッファおよびPEインデックスバッファを定義する必要があります。

フォーマットバッファ

初期化コール／初期化コールの繰り返しでは、ハイパー出口の値生成コールの前に親フィールド値の圧縮解除に使用する、フォーマットバッファを提供できます。フォーマットバッファの構文は、「コマンドリファレンス」の「Adabasの呼び出し」、「フォーマットバッファとレコードバッファ」のフォーマットバッファ構文で説明されているものと同じですが、許可されるフィールド定義には次の制限があります。

- 親フィールドのみ指定できます。
- 親フィールドがピリオディックグループに属している場合は、そのピリオディックグループのピリオディックグループカウントを指定できます。

フォーマットバッファが指定されていない場合、Adabasは、各親フィールドに次のフォーマットバッファ要素が含まれる、フォーマットバッファを想定します。

- ピリオディックグループに属していない非 MU 親フィールド：name
- ピリオディックグループに属していない MU 親フィールド：nameC,name1-N
- ピリオディックグループ PG に属している非 MU 親フィールド PA：PGC,PA1-N
- ピリオディックグループに PG に属している MU 親フィールド PA：PGC,PA1C,PA1(1-N),PA2C,PA2(1-N),...

次に、デフォルトのバッファ要素の使用例を示します。これらの例では、次に示す FDT が使用されています。

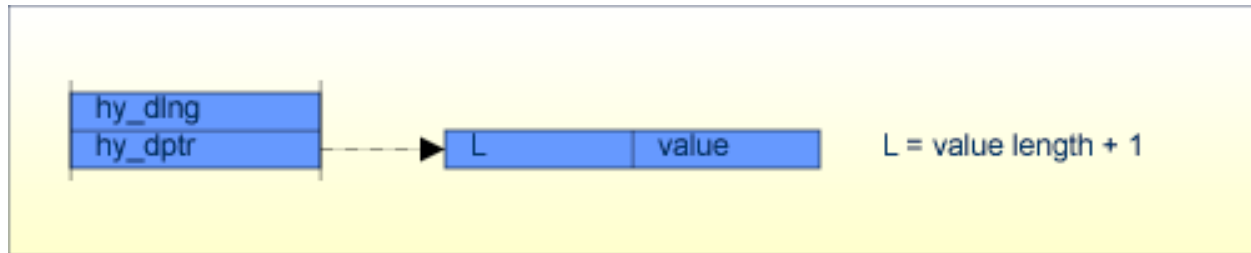
```
01,AA,0,A
01,MU,10,B,MU
01,PG,PE
  02,PP,40,A
  02,PM,20,B,MU
  .
  .
  .
```

例 1：

```
H1,10,A=HYPER(1,AA)
```

親フィールド AA は MU フィールドではなく、ピリオディックグループ内にはありません。そのため、デフォルトのフォーマットバッファは次のようになります。

```
AA.
```



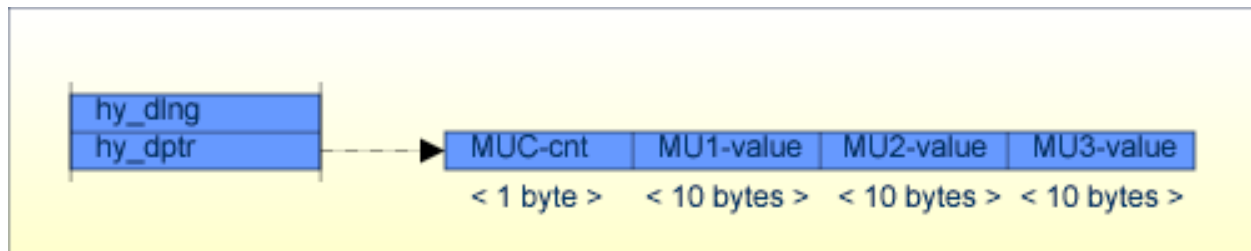
必要な長さ = 予想される最大値の長さ + 1。

例 2：

```
H2,20,A,MU=HYPER(1,MU)
```

MU は、ピリオディックグループ内の MU フィールドではありません。そのため、デフォルトのフォーマットバッファは次のようになります。

```
MUC,MU1-N.
```



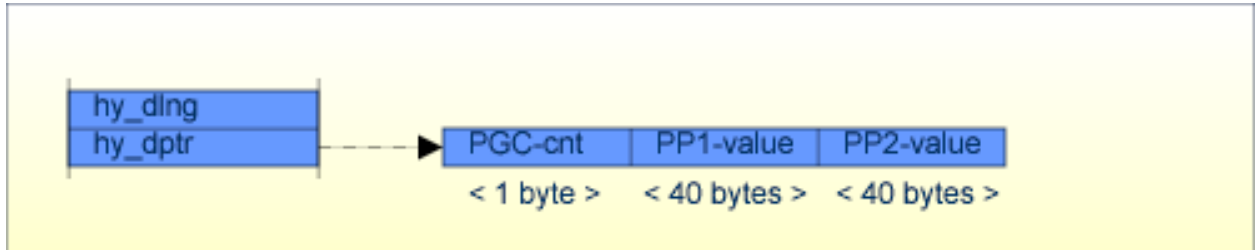
必要な長さは、予想される最大MUカウントに依存します。この例では、MUカウントが3で、必要なバイト数は 31 です。

例 3 :

H3,40,A,PE=HYPER(1,PP)

PP は PE 内の非 MU フィールドです。そのため、デフォルトのフォーマットバッファは次のようになります。

PGC,PP1-N.



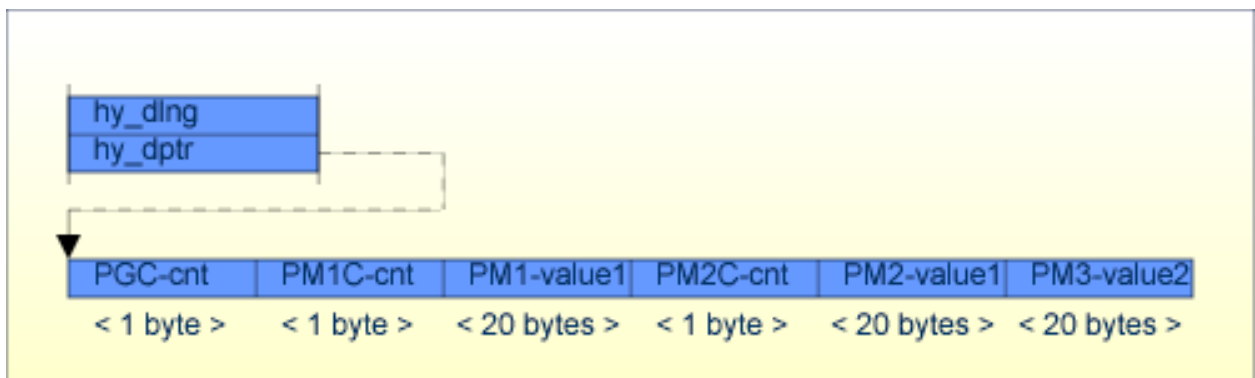
必要な長さは、予想される最大ピリオディックグループカウントに依存します。この例では、ピリオディックグループカウントが 2 で、必要なバイト数は 81 です。

例 4 :

H4,40,A,PE,MU=HYPER(1,PM)

親フィールドはピリオディックグループ内の MU フィールドです。そのため、デフォルトのフォーマットバッファはピリオディックグループカウントに依存します。この例では、ピリオディックグループカウントが 2 で、フォーマットバッファは次のようになります。

PGC,PM1C,PM1(1-N),PM2C,PM2(1-N).



データバッファの必要な長さは、予想される最大ピリオディックグループカウントと予想される MU カウントによって異なります。PGC=2、PM1C=1、PM2C=2 のサンプルデータでの必要な長さは 63 です。

ハイパー出口インターフェイス

リエントラント／非リエントラントインターフェイス

ハイパー出口は、リエントラントインターフェイスまたは非リエントラントインターフェイスを使用して実装できます。

コンピュータールーチンは、安全に同時に実行できる場合、リエントラントと呼ばれます。つまり、ルーチンがすでに別のスレッドで実行されていても、再入力できます。リエントラントハイパー出口の実装では、ストレージクラススタティックの変数は、初期化中にのみ更新できます。そうしないと、1つのスレッドが別のスレッドによって作成されたデータを上書きする可能性があります。これは、コール元のニュークリアスまたはユーティリティがハイパー出口バッファを提供する必要があることも意味します。ハイパー出口を呼び出す各スレッドは、専用ハイパー出口バッファを提供する必要があります。例外はフォーマットバッファです。常にハイパー出口によって常に提供されます。これは初期化中にのみ使用されるので、リエントラントインターフェイス用にも使用されます。

非リエントラントインターフェイスの場合、Adabas がハイパー出口コールをシリアル化します。このハイパー出口コールのシリアル化によって、パフォーマンスが低下する可能性があります。この場合、使用するハイパー出口バッファは、ハイパー出口によって提供される必要があります。

ハイパー出口バッファの処理が異なる場合は、リエントラントまたは非リエントラントのハイパー出口のために、異なるコールを異なる方法で実装する必要があります。

初期化コールでは、リエントラントモードまたは非リエントラントモードのどちらで実行しているかを、ハイパー出口が通知します。リエントラントモードでは、フィールド「hy_mode」を HY_REENTRANT（リエントラントバージョン用）に設定する必要があります。非リエントラントモードの場合、hy_mode は HY_N_REENTRANT と等しくなっている必要があります。これがデフォルトです。

ハイパー出口コール

以下で出力パラメータとして記載されていない、ハイパー出口コントロールブロックのフィールドは、ハイパー出口コールの最中に変更しないでください。

初期化コール

入力パラメータ（リエントラントおよび非リエントラント）

hy_ctype	初期化コールの場合、この値は 2 です。
hy_dbid	データベースの番号。
hy_fnr	ハイパーディスクリプタの番号。
hy_hyname	ハイパーディスクリプタの名前。

出力パラメータ（非リエントラント）

hy_structure	実際の構造番号は 2 です。
hy_rsp	成功コードまたはエラーのレスポンスコード。 特に、初期化コール hy_rsp=1 で使用して、「初期化コールの繰り返し」を開始することができます。 初期化コールの間に hy_rsp>1 の場合、ユーティリティやニュークリアスはエラーで終了します。（レスポンス 79 - ハイパー出口は使用不可）。ニュークリアスログが次のエラーメッセージを取得します：%ADANUC-W-HYERR, HYPER userexit, descriptor HY, file ..., reason=....
hy_mode	HY_N_REENTRANT
hy_dlng	データバッファ長親フィールドに十分な大きさのデータバッファを定義する必要があります。
hy_dptr	親フィールド値のデータバッファポインタ。
hy_flg	フォーマットバッファが指定されていない場合、独自のフォーマットバッファ長または 0。
hy_fptr	フォーマットバッファが指定されていない場合、独自のフォーマットバッファポインタまたは NULL。
フォーマットバッファ	独自のフォーマットバッファ（オプション）。

出力パラメータ（リエントラント）

hy_structure	実際の構造番号は 2 です。
hy_rsp	成功コードまたはエラーのレスポンスコード。 特に、初期化コール hy_rsp=1 で使用して、「初期化コールの繰り返し」を開始することができます。 初期化コールの間に hy_rsp>1 の場合、ユーティリティやニュークリアスはエラーで終了します。（レスポンス 79 - ハイパー出口は使用不可）。ニュークリアスログが次のエラーメッセージを取得します：%ADANUC-W-HYERR, HYPER userexit, descriptor HY, file ..., reason=....

ユーザー出口とハイパー出口

hy_mode	HY_REENTRANT
hy_dlng	データ転送バッファ長。初期化コールで、データバッファ長を設定する必要があります。
hy_xlng	インデックスバリュースバッファ長
hy_ilng	ISN バリュースバッファ長 (0 = 使用しない)。
hy_plng	ピリオディックグループインデックスバッファ長 (0 = 使用しない)。
hy_flg	フォーマットバッファが指定されていない場合、独自のフォーマットバッファ長または 0。
hy_fptr	フォーマットバッファが指定されていない場合、独自のフォーマットバッファポインタまたは NULL。
フォーマットバッファ	独自のフォーマットバッファ (オプション)。

初期化コール (繰り返し)

初期化コールで hy_rsp が 1 に設定されている場合、Adabas は hy_ctype=3 でハイパー出口コールを再度呼び出します。上記で説明したように、Adabas は、データバッファを使用して、ハイパーディスクリプタおよび親フィールドの情報を返します。

値生成コール

入力パラメータ (非リエントラント)

hy_ctype	0 は値生成コールを示します。
hy_dbid	データベース ID です。
hy_fnr	ハイパーディスクリプタが含まれるファイルのファイル番号。
hy_hyname	ハイパーディスクリプタの名前。
hy_dlng	データバッファ長
hy_dptra	データバッファへのポインタ。
hy_isn	レコードの ISN。
データバッファ	親フィールド値。

出力パラメータ (非リエントラント)

hy_rsp	成功コードまたはエラーのレスポンスコード。 特に、値生成コール hy_rsp=1 で使用して、「値生成コールの繰り返し」を開始することができます。 hy_rsp>1 の場合、ハイパー出口はエラーのレスポンスコードを返します。エラーは Adabas ニュークリアスのレスポンスコード 86 に変換され、ハイパー出口の値はコントロールブロックのアディション2フィールドに返されます。
hy_cnt	インデックス値の数。

hy_xlng	インデックスバリュースバッファの長さ。
hy_xptr	インデックスバッファへのポインタ。
hy_icnt	ISN 値の数 (0 = 使用しない)。
hy_ilng	ISN バッファ長 (0 = 使用しない)。
hy_iptr	ISN バッファへのポインタ
hy_pcmt	PE インデックスの数 (0 = 使用しない)。
hy_plng	PE インデックスバッファの長さ (0 = 使用しない)。
hy_pptr	PE インデックスバッファへのポインタ (空値 = 使用しない)。
インデックスバリュースバッファ	上で説明されているハイパーディスクリプタ値。
ISN バッファ	各ディスクリプタ値に関連付けられている ISN (オプション)。
PE インデックスバッファ	PE インデックス。オプション PE を指定してハイパーディスクリプタが定義されていて、かつサーチバッファ値コールがない場合にのみ使用します。

入力パラメータ (リエントラント)

hy_structure	実際の構造番号は 2 です。
hy_ctype	値生成コールの後は 0 です。
hy_dbid	データベースの番号。
hy_fnr	ハイパーディスクリプタが含まれるファイルのファイル番号。
hy_hyname	ハイパーディスクリプタの名前。
hy_dptra	親フィールド値へのポインタ。
hy_dlng	データ転送バッファ長。
hy_isn	レコードの ISN。
hy_ilng	ISN バッファの長さ。
hy_iptr	ISN バッファへのポインタ
hy_xlng	インデックスバリュースバッファの長さ。
hy_xptr	インデックスバリュースバッファへのポインタ。
hy_plng	ピリオディックカウントバッファ長。
hy_pptr	ピリオディックカウントバッファへのポインタ。

出力パラメータ（リエントラント）

hy_rsp	成功コードまたはエラーのレスポンスコード。 特に、値生成コール hy_rsp=1 で使用して、「値生成コールの繰り返し」を開始することができます。 hy_rsp>1 の場合、ハイパー出口はエラーのレスポンスコードを返します。エラーは Adabas ニュークリアスのレスポンスコード 86 に変換され、ハイパー出口の値はコントロールブロックのアディション2フィールドに返されます。
hy_icnt	ISN 値の数。
hy_xcnt	インデックス値の数。
hy_pcnt	ピリオディックカウント値の数。
インデックスバリュースバッファ	上で説明されているハイパーディスクリプタ値。
ISN バッファ	各ディスクリプタ値に関連付けられている ISN（オプション）。
PE インデックスバッファ	各ディスクリプタ値に関連付けられている ISN（オプション）。

値生成コールの繰り返し

値生成コール中に hy_rsp が 1 に設定されている場合、Adabas はインデックス値を処理してから、hy_ctype=1 を使用して、同じデータバッファ内容（リエントラントおよび非リエントラント）でハイパー出口を再び呼び出します。これで、ハイパー出口ではより多くのハイパーディスクリプタ値を生成したり、異なる ISN のハイパーディスクリプタ値を生成したりできるようになります。

ユーザー出口およびハイパー出口の作成と定義

ニュークリアスおよびユーティリティは、USEREXIT パラメータ（ADACMP または ADADCU ユーティリティ）を使用してユーザー出口をアクティブにします。このパラメータが設定されると、ユーザー出口は処理の特定のポイントでコールされます。

ハイパー出口は、ハイパーディスクリプタが定義されているファイルが更新されるとアクティブになります。

ユーザー出口およびハイパー出口は、次の手順を実行して定義できます。

1. ユーザー出口またはハイパー出口を、C プログラミング言語で記述します。ヘッダーファイル adaux.h（ユーザー出口用）および adahyx.h（ハイパー出口用）を使用する必要があります。

これらの出口は、デフォルトの関数名を使用して記述することもできます。表記規則として、"uex_"または"hyx_"の後にユーザー出口番号かハイパー出口番号を付加します。次に例を示します。

```
uex_1()      /* Default name user exit 1 */
{}

hyx_4()      /* Default name hyperexit 4 */
{}
```

環境変数／論理名を設定すれば、他の名前も使用できます（手順 4 を参照）。

2. ユーザー出口またはハイパー出口のソースファイルをコンパイルします。位置非依存コードのオプションを使用する必要があります。
3. ユーザー出口またはハイパー出口を共有ライブラリとしてリンクします。共有ライブラリの作成に使用するリンクオプションは、マシン依存です。
4. 共有ライブラリを環境変数／論理名に関連づけて、ユーザー出口またはハイパー出口をAdabasで使用可能にします。共有ライブラリを含んでいるディレクトリがLD_LIBRARY_PATH (UNIX) または PATH (Windows) に含まれている場合、共有ライブラリのメンバ名を指定するか、または共有ライブラリの絶対パスを指定することができます。環境変数は、「ADAUEX_」(ユーザー出口用) または「ADAHYX_」(ハイパー出口用) です。それぞれユーザー出口番号またはハイパー出口番号を付加します。次に例を示します。

csh の場合：

```
setenv ADAUEX_1 userex1.sl  # connect user exit 1 (see note below)
setenv ADAHYX_4 $ADADATADIR/exits/hypex4.sl  # connect hyperexit 4 (see note below)
```

注：UNIX系を使用している場合は、共有ライブラリの拡張子が"sl"または"so"になります。

ksh の場合：

```
ADAUEX_1=userex1.sl
export ADAUEX_1
```

Windows の場合：

```
set ADAUEX_1=userex1.dll  # connect user exit 1
set ADAHYX_4=%ADADATADIR%\exits\hypex4.dll  # connect hyperexit 4
```

便宜上、ユーザー出口およびハイパー出口のデフォルトの関数名は次のように変更できます。

```
my_uex_1 ()
{}

nga_hyx_4 ()
{}

setenv ADAUEX_1 "userex1.sl my_uex_1" (UNIX platforms)
setenv ADAHYX_4 "$ADADATADIR/exits/hypex4.sl nga_hyx_4"

set ADAUEX_1=userex1.dll;my_uex_1 (Windows)
set ADAHYX_4=%ADADATADIR%\exits\hypex4.dll;nga_hyx_4
```

ニュークリアスが、環境変数に指定された出口を見つけられない場合、次の手順で示されている検索を続行します。

5. ハイパー出口の共有ライブラリがデフォルトのデータベースディレクトリに配置され、次の命名規則が使用されている場合は、ADAHYX 環境変数を省略できます。

UNIX プラットフォーム：

```
$ADADATADIR/db<xxx>/adahyx_<i>.<ext>
```

xxx はデータベース ID、i はハイパー出口番号、ext は共有ライブラリの拡張子 ("sl" または "so") です。

Windows：

```
%ADADATADIR%\db<xxx>\adahyx_<i>.dll
```

xxx はデータベース ID、i はハイパー出口の番号です。

例 (**UNIX** プラットフォーム)：

```
$ADADATADIR/db076/adahyx_1.sl
```

例 (**Windows**)：

```
%ADADATADIR%\db076\adahyx_1.dll
```

共有ライブラリをロードするために、Adabas は最初に対応する ADAHYX 環境変数を取得します。環境変数が存在しない場合は、デフォルトのデータベースディレクトリ (\$ADADATADIR/db<xxx> または %ADADATADIR%\db<xxx>) 内を探します。

Adabas キットには、必要とされる C ヘッダーファイル、ユーザー出口とハイパー出口のためのサンプルソース、および対応する makefile が含まれています。ADALNK ユーザー出口は、サーバープラットフォームと異なるプラットフォーム上で使用できるので、ADALNK ユーザー出口、対応する C ヘッダーファイル、および makefile は、Entire Net-Work でも提供されます (詳細については Entire Net-Work ドキュメントを参照)。

UNIX プラットフォーム

必要なCヘッダーファイルは、ディレクトリ \$ADAPROGDIR/inc および \$ACLDIR/\$ACLVERS/inc に置かれます。ニュークリアスおよびユーティリティのユーザー出口にはインクルードファイル adaux.h、ADALNK ユーザー出口には lnuex.h、ハイパー出口には adahyx.h がそれぞれ必要です。

ニュークリアスユーザー出口 1、ニュークリアスユーザー出口 2、ニュークリアスユーザー出口 4、ADACMP ユーザー出口 6、ADAULD ユーザー出口 7、ハイパー出口 1 のサンプルソースファイル、およびそれらに対応する makefile は、\$ADAPROGDIR/examples に置かれます。

これらのユーザー出口例の 1 つを構築するには、次のように入力します。

```
cd $ADAPROGDIR/examples
make target
```

target には次のいずれかが入ります。

ユーザー出口／ハイパー出口	サンプルソースファイル名	<i>target</i>
ニュークリアスユーザー出口 1	adaux1.c	uex1
ニュークリアスユーザー出口 2	adaux2.c	uex2
ニュークリアスユーザー出口 4	adaux4.c	uex4
ADACMP ユーザー出口 6	adaux6.c	uex6
ADAULD ユーザー出口 7	adaux7.c	uex7
ハイパー出口 1	adahyx1.c	hyx1

ユーザー出口やハイパー出口のための共有ライブラリは \$ADAPROGDIR/examples/server に作成されます。

ADALNK ユーザー出口 0 と ADALNK ユーザー出口 1 のサンプルソースファイル（ファイル名は lnuex.c）および対応する makefile は \$ACLDIR/examples に置かれます。

このユーザー出口例を構築するには、次のように入力します。

```
cd $ACLDIR/examples/client
make lnuex ↵
```

ユーザー出口用の共有ライブラリは \$ACLDIR/examples に作成されます。

Windows プラットフォーム

必要な C ヘッダーファイルは、ディレクトリ %ADAPROGDIR%\inc および %ACLDIR%\inc に置かれます。ニュークリアスおよびユーティリティのユーザー出口にはインクルードファイル

ユーザー出口とハイパー出口

adauex.h、ADALNK ユーザー出口には lnuex.h、ハイパー出口には adahyx.h がそれぞれ必要になります。

ニュークリアスユーザー出口 1、ニュークリアスユーザー出口 2、ニュークリアスユーザー出口 4、ADACMP ユーザー出口 6、ADAULD ユーザー出口 7、ハイパー出口 1 のサンプルソースファイル、およびそれらに対応する makefile は、%ADAPROGDIR%\examples に置かれます。

これらのユーザー出口例の 1 つを構築するには、次のように入力します。

```
cd %ADAPROGDIR%\examples
nmake target
```

target には次のいずれかが入ります。

ユーザー出口／ハイパー出口	<i>target</i>
ニュークリアスユーザー出口 1	uex1
ニュークリアスユーザー出口 2	uex2
ニュークリアスユーザー出口 4	uex4
ADACMP ユーザー出口 6	uex6
ADAULD ユーザー出口 7	uex7
ハイパー出口 1	hyx1

ADALNK ユーザー出口 0 と ADALNK ユーザー出口 1 のサンプルソースファイル（ファイル名は lnuex.c）および対応する makefile は \$ACLDIR\examples に置かれます。

このユーザー出口例を構築するには、次のように入力します。

```
cd %ACLDIR%\examples
nmake lnuex ↵
```

ユーザー出口やハイパー出口の DLL は %ACLDIR%\examples に作成されます。

ニュークリアスがアクティブなときの新ハイパー出口

ニュークリアスがすでにアクティブな場合、ハイパー出口は、ユーティリティを介してアクティブにすることができます。これは、次の場合に実行できます。

- 新しいファイルが定義されたとき（ADAFDU）、
- ファイルがリストアまたはオーバーレイされたとき（ADABCK）、
- ファイルがインポートされたとき（ADAORD）または
- ADAINV を使用して既存のファイル用にハイパーディスクリプタが作成されたとき

初回の値生成コールを行う前に、ニュークリアスはハイパー出口を初期化します。

ADAFDU の場合、FDT にハイパーディスクリプタを含むファイルを定義するとき、有効な ADAHYX 環境変数をニュークリアスに転送することも可能です。

例

```
setenv ADAHYX_1 /user/adabas/hyper1.sl  
setenv FDUFDT file20.fdt  
adafdu < file20.fdu
```

ADAHYX 環境変数が指定されていない場合、またはファイルが ADABCK を使用してリストアされている場合、ニュークリアスは、ニュークリアス起動時に指定された場所でハイパー出口を検索します（上記を参照）。したがって、対応する共有ライブラリや DLL を該当するディレクトリに移動する必要があります。

10

読み取り専用デバイス上の Adabas

- Adabas ニュークリアス使用時の制限 214
- Adabas ユーティリティ使用時の制限 215

Adabas データベースは読み取り専用デバイスに保存できます。少なくとも ASSO1 が読み取り専用デバイスに保存すれば、データベース全体が読み取り専用とみなされます。

Adabas ニュークリアス使用時の制限

読み取り専用データベースの場合は、Adabas ニュークリアスの使用に関していくつかの制限があります。

ASSO1 が読み取り専用デバイスに保存されている場合、ASSO1 を書き込みモードでオープンできない場合、またはニュークリアスセッションが Adabas パラメータ `OPTIONS=READONLY` で開始されている場合、ニュークリアスは読み取り専用モードで実行されます。最初のケースでは、ニュークリアスは、`OPTIONS` パラメータの設定に関係なく、自動的に読み取り専用モードに切り換わります。2 番目のケースは、データベースが読み取り専用デバイスに保存されていないときにニュークリアスセッションを読み取り専用モードで実行するために使用できます。

書き込み可能な WORK コンテナが指定された場合、保留中の自動再スタートチェックが常に実行されます。自動再スタートが保留状態の場合、読み取り専用データベースセッションは開始できません。

ニュークリアスが読み取り専用モードで起動すると、起動パラメータとして指定されたデータベース ID は、実際のデータベース ID とは照合されません。その代わり、データベースコンテナファイルの環境変数設定がデータベースを識別します。これは、読み取り専用データベースは、有効な任意のデータベース ID で実行できることを意味しています。したがって、ニュークリアスまたはユーティリティを起動するとき、任意の番号をデータベース ID パラメータに割り当てることができます。ニュークリアスとコミュニケーションするアプリケーションは、この番号を使用してデータベースにアクセスします。

部分的に読み取り専用のデータベースおよび ASSO1 が書き込み可能なデータベースは Adabas ではサポートしません。

READONLY オプション

ニュークリアスパラメータ `OPTION=READONLY` が設定されると、ニュークリアスは読み取り専用モードで実行します。データベースは、読み取り専用または書き込み可能なデバイスに設置することができます。ASSO1 が読み取り専用デバイスにある場合、パラメータ `OPTION=READONLY` が設定されていなくても、ADANUC は自動的に読み取り専用モードに切り換わります。この場合、警告メッセージが表示されます。

読み取り専用モードでは次の制限があります。

- 更新コマンド（A1、E1、N1、N2）は無効です。
- 書き込みチェックポイントは無効です（C1 およびユーティリティ/EXU チェックポイント）。
- ロギングは無効です（ニュークリアス起動には NOPLOG が必要です）。

- ホールドロジックは無効になります。したがって、コマンド L4、L5、L6、S4 を実行することは可能ですが、ホールドキューに影響しません。つまり、これらのコマンドの動作は、それぞれ L1、L2、L3、S1 とまったく同じになります。また、コマンド BT、ET、HI、RI を実行することも可能ですが、実際には何も処理しません。
- ユーザーデータ付き ET/CL コマンドは禁止されます。

OP コマンドに指定できるすべてのファイルオープンモードが許可されるので、有効なすべてのユーザータイプも許可されます。ただし、排他制御ユーザーは OPEN および CLSE チェックポイントを書かないことに注意してください。

セッション番号は、読み取り専用モードで実行しているニュークリアスセッション実行では増えません。コマンドログはサポートされますが、コマンドログ番号は増えません。

WORK コンテナについては、2 つのオプションがあります。

1. 必須ではないが、WORK コンテナがある。
2. WORK がない。この場合、環境変数 WORK1 を「READONLY」に設定する必要があります。

Adabas ユーティリティ使用時の制限

指定したデータベース ID のニュークリアスが読み取り専用モードで実行している場合、または ASSO1 コンテナファイルが読み取り専用デバイスに設置されている場合、ユーティリティは読み取り専用データベースを処理することができます。どちらでもない場合、データベース ID は実際のデータベース ID と照合するパラメータとして与えられます（前述のニュークリアス処理と同じ）。ただし、実行しているニュークリアスとユーティリティの間のコミュニケーションを有効にするには、それらを同じデータベース ID で開始する必要があります（格納されたデータベース ID と異なっても構いません）。

ニュークリアスとユーティリティに指定したデータベース ID が異なっている場合、コミュニケーションリンクは失敗し、ニュークリアスと同じデータベース上で、ユーティリティはオフラインモードで実行することに注意してください。

書き込み可能な WORK コンテナが指定された場合、保留中の自動再スタートチェックが常に実行されます。WORK1 環境変数に READONLY を設定すると、ASSO1 コンテナは読み取り専用デバイスに設置する必要があります。または OPTIONS=READONLY でニュークリアスを実行しなければなりません。そうでないと、ユーティリティはエラーメッセージ RDONP ("Dataset WORK1, READONLY is not permitted") で終了します。

データベースを修正するためのユーティリティは、読み取り専用データベースで使うことができません。これらのユーティリティに対するオープンデータベースコールは、エラーメッセージ UNPRRD (「Readonly database, utility not permitted to run」) で終了します。このことは次のユーティリティに適用されます。

■ ADACVT

- ADADBM
- ADAFDU
- ADAREC

他のユーティリティについては、機能設定が制限されます。これらのユーティリティは警告 RONLYDB（「Readonly database, some functions disabled」）を表示します。

次のユーティリティが関連します。

- ADABCK：RESTORE および OVERLAY 機能は使用できません。
- ADAINV：SUMMARY と VERIFY 機能だけが使用できます。
- ADAMUP：SUMMARY 機能だけが使用できます。
- ADAORD：EXPORT 機能だけが使用できます。
- ADAOPR：
 - LOCK と UNLOCK 機能は使用できません。
 - FEOF=CLOG は有効ですが、CLOG 番号が増えても、次のニュークリアスセッションには引き継がれません。後続の読み取り専用ニュークリアスセッションはどれも同じ CLOG 番号で開始するということです。
- ADASCR：DISPLAY 機能だけが使用できます。

ニュークリアスが読み取り専用モードで実行するとき、または ASSO1 が読み取り専用デバイスに存在するデータベースに対してユーティリティを実行するとき、それらはパラメータとして与えられたデータベース ID とデータベースのデータベース ID とを照合しません。ただし、ニュークリアスとユーティリティ間のコミュニケーションを有効にするために、それらは同じデータベース ID で起動しなければなりません（データベースに格納されたものと異なっても構いません）。

読み取り専用 RAW セクションはサポートされません。