Net Express

通信

Micro Focus NetExpress™

通信

Micro Focus®

第 3 版
1998 年 10 月

序文

このマニュアルでは、NetExpress で使用できる通信ミドルウェアについて説明します。

対象読者

NetExpress を使用して Web アプリケーションを作成する場合、通信プログラムや構成は必要ありません。

このユーザー ガイドでは、汎用クライアント/サーバー結合モジュールを使用して、ネットワークでクライアントの COBOL プログラムをサーバーに接続する方法について説明します。

このマニュアルの使用方法

「第 1 章 はじめに」では、COBOL を使ってクライアントをサーバーに接続するさまざまな方法について簡単に説明します。

「第 2 章 クライアント/サーバー結合」では、クライアントとサーバーを結合し、クライアントの COBOL プログラムをサーバーに接続する方法について説明します。

表記規則

このユーザー ガイドでは、次の書体や規則を使用します。

- 入力するテキストは、次のように表示します。

  ```
  cat script_name | more
  ```

  斜体テキストはコマンドの一部として入力する変数を表します。

- コマンド行やコード例でオプション入力するテキストは、角かっこ ([]) で囲みます。次の式では、オプションの単語 NOT を入力しない場合、*column_name* は pattern_value に設定されます。一方、NOT を入力した場合、*column_name* は *pattern_value* 以外に設定されます。

  ```
  column_name [NOT] LIKE pattern_value
  ```

- 特定のモデルやオペレーティング システムだけに適用する項や段落については、段落のすぐ前に太字斜体の項目名が記載されています。例えば、次のようになります。

  UNIX

この段落は UNIX システムだけに適用されます。

目次

第1章 はじめに

このユーザー ガイドでは、NetExpress の通信ミドルウェアを使用して、クライアント/サーバー アプリケーションを作成し、実行する方法について説明します。

COBOL でクライアントをサーバーに接続するには、次の方法のいずれかを使用します。

- Application-to-Application Interface (AAI)

  AAI は、環境や通信プロトコルの選択肢を広く持つ、アプリケーション プログラムにさまざまな接続サービスを提供するミドルウェア製品です。

  AAI は NetExpress の一部ではなく、別途購入が可能な製品です。詳細については、Micro Focus の営業担当にお問い合わせください。

- クライアント/サーバー結合

  NetExpress では、クライアントをサーバーに接続する簡便な方法の一例として、クライアント/サーバー結合を使用します。 詳細については、*「クライアント/サーバー結合」* の章を参照してください。

  クライアント/サーバー結合と Fileshare では、通信インターフェイスとして Micro Focus 共通通信インターフェイス (CCI) を使用します。 CCI は、さまざまな通信プロトコル上で実行できます。NetExpress では、TCP/IP、NetBEUI、Novell IPX、APPC および 動的データ交換 (DDE) がサポートされています。 詳細については、NetExpress オンライン ブック *『CCI の構成』* を参照してください。

第2章 クライアント/サーバー結合

この章ではクライアント/サーバー結合の原理を説明し、ユーザーのプログラムを汎用クライアント/サーバー モジュールに結合する方法を説明します。

ダイアログ システム アプリケーションでクライアント/サーバー結合を使用する場合、オンライン マニュアル 『ダイアログ システム ユーザー ガイド』 の 「*クライアント/サーバー結合の使用*」 の章を参照してください。この章には、ダイアログ システムに固有のコード例が記載されています。

2.1 はじめに

クライアント/サーバー結合とは、ユーザーが通信プログラムに煩わされずにクライアント/サーバーアーキテクチャを実現できる、25 クライアントまでの簡便なリモート プロシージャ コール (RPC) メカニズムです。25 クライアント以上を必要とするアプリケーションへ移行したいユーザーには、クライアント/サーバー結合のAAI (Application to Application Interface) コンポーネントが用意されています。

クライアント/サーバー結合は、NetExpress の一部として提供されており、フォールト トレラントな大規模ミドルウェア ソリューションを必要としないアプリケーションで使用します。フォールト トレラント、ロールバックおよび回復オプションを必要とするアプリケーションを使用できる、より包括的なミドルウェア製品に移行する場合には、AAI クライアント コンポーネントが NetExpress に提供されています。開発プロジェクトを開始する前に、AAI インターフェイスに直接プログラミングを行う可能性もご検討ください。この API の詳細情報については、AAI サーバー コンポーネントに提供されています。AAI サーバー コンポーネントは別売です。詳細は当社営業担当にお問い合わせください。

クライアント/サーバー結合が提供する諸機能とパフォーマンスが、ご自身のビジネスに適当かどうかご確認ください。多くの場合、これこそ導入しやすくコスト効率のよいソリューションであることがおわかりになるでしょう。

このクライアント/サーバー結合が提供する mfclient と mfserver という 2 つのモジュールのおかげで、ユーザーはアプリケーションに通信プログラムを組み込む必要がありません。この 2 つのモジュールはあらゆるアプリケーションで利用できる汎用モジュールです。

この 2 つのモジュールは、構成ファイルの設定に従って通信リンクを管理し、データ転送を行います。各通信リンクの最後にはユーザー定義プログラムとやり取りもできます。mfclient モジュールはユーザーが作成したクライアント プログラム (例えば、ユーザー インターフェイスを処理するプログラム) から呼び出されますが、mfserver はユーザーが作成したサーバー プログラム (例えば、データ アクセスやアプリケーション ロジックを処理するプログラム) を呼び出します。

ユーザーが作成するクライアント プログラムとサーバー プログラムは mfclisrv.cpy というコピー ファイルを使用して、汎用のクライアント/サーバー モジュールと相互通信します。

NetExpress では、クライアント/サーバー結合の使用方法を示すアプリケーション例も使用できます。このアプリケーション例は、クライアントとサーバーの間でファイル データを転送するもので、NetExpress の ¥demo ディレクトリにある Csbind サブディレクトリにインストールされています。

2.2 クライアント/サーバー結合の内容

クライアント/サーバー結合は、サーバー側で mfserver の非専用コピーを実行することで可能になります。 mfserver は、取り決めたサーバー名を使用してすべてのクライアントと通信します。mfserver モジュールの機能は、クライアントからの要求を受けて接続を確立したり終了したりするだけなので、最初のデフォルト値のまま使用できます。

情報の流れを次に示します。(ここでは、デモ用のアプリケーション Csbind のプログラム ファイルをクライアントプログラム例およびサーバー プログラム例として使用します。)

1. ユーザーが作成したクライアント プログラム (netxcli.cbl) がアプリケーション初期化コードを実行し、ファイル名を要求するプロンプトを表示し、指定されたファイルが存在するか確認してから mfclient を呼び出します。

2. 初めて呼び出された mfclient は構成ファイル netxdem.cfg から構成情報を読み込みます。

3. mfserver が接続要求を受け取ります。

4. mfserver 各クライアントに対して、セカンダリサーバーを生成します。サーバー名は内部生成されますが、元のサーバー名に数値 ID を付加したもの (例えば、mfserver01) となります。 または、構成ファイルにサーバー名を指定しておくこともできます。

5. mfserver がセカンダリサーバー名 (mfserver01) を mfclient に送り返し、この対話は終了します。

6. mfclient はセカンダリサーバー (mfserver01) に接続し、LNK-PARAM-BLOCK (*「クライアント/サーバー結合のコピー ファイル」の項を参照*) を通して、構成ファイルから取得したパラメータを引き渡します。

7. mfserver01 はユーザーが作成したサーバー プログラム (netxserv.cbl) を呼び出し、LINKAGE SECTION を通して mfclient から受け取ったパラメータを渡します。

8. ユーザーが作成したサーバープログラム (netxserv.cbl) は初めて呼び出されたので、アプリケーション初期化コードを実行して、プログラムを終了します。制御は mfserver01 に戻ります。

9. mfserver01 は mfclient に制御を返します。

10. mfclient は接続が確立したことを確認し、ユーザーが作成したクライアント プログラム (netxcli.cbl) に制御を渡します。

11. クライアント プログラム (netxcli.cbl) はエンドユーザーに指定されたファイルをオープンし、レコードを読み込み、mfclient を呼び出して LINKAGE SECTION を通してそのレコードを渡します。

12. mfclient は渡されたデータを内部バッファを使用して mfserver01 に渡します。

13. mfserver01 がそのデータをユーザーのサーバー プログラム (netxserv.cbl) に渡すと、サーバー プログラムがファイルを作成してレコードを書き込みます。

    ユーザーのクライアント プログラム (netxcli.cbl) がファイルの終わりを検出するまで、このようなレコード単位の読み込み/書き込みループが繰り返されます。ファイルの終わりに来ると、クライアントプログラムはファイルをクローズし、プログラムの終了をサーバーに知らせます。これに対してサーバーはファイルをコピーしたディレクトリ名を返します。

14. クライアントプログラム (netxcli.cbl) はサーバー上に作成されたファイル名を表示し、LNK-CNTRL-FLAG を "client-ending" に設定して、mfclient を呼び出して終了します。

- mfclient が "client-ending" パラメータを mfserver01 に渡すと、mfserver01 はこのパラメータをユーザーが作成したサーバー プログラム (netxserv.cbl) に渡し、その結果ユーザー サーバー プログラムが終了します。

- mfclient は、セカンダリ サーバー (mfserver01) が終了したことをベース サーバー (mfserver) に知らせます。

2.3 クライアント/サーバー結合の使用方法

アプリケーションでクライアント/サーバー結合を使用するには、次の作業が必要です。

- 構成ファイル (.cfg) を作成して、mfclient と mfserver の 2 つのモジュールの動作と接続方法を設定します。構成ファイルは ASCII テキスト ファイルで、次のような重要な情報をパラメータで指定します。

    - 使用する通信プロトコル

    - mfserver が呼び出すユーザー プログラム名

- ユーザーのクライアント プログラムに mfclient モジュールを呼び出すコードを追加します。

- ユーザーのサーバー プログラムが mfserver モジュールに呼び出されてもよいようにコードを追加します。

上記の各ステップについて以下の各項で詳しく説明します。

2.4 クライアント/サーバー結合のコピー ファイル

mfclient と mfserver の 2 つのモジュールは mfclisrv.cpy というコピー ファイルに定義されたパラメータ ブロック (LNK-PARAM-BLOCK) を通して情報の受け渡しを行います。このパラメータ ブロックは、ユーザー プログラムを呼び出すときの情報の引き渡しでも使用します。mfclisrv.cpy コピー ファイルは mfclient と mfserver の 2 つのモジュールを使用するすべての COBOL プログラムでインクルードしなければなりません。

mfclisrv.cpy コピー ファイルは、NetExpress 基本ディレクトリの SOURCE ディレクトリにインストールされています。

2.5 クライアント/サーバー結合の構成ファイル

クライアント/サーバー結合では構成ファイルを使用して mfclient と mfserver の 2 つのモジュールの動作と通信リンクを制御します。

クライアント/サーバー結合の構成ファイルは拡張子が .cfg の ASCII テキスト ファイルです。構成ファイル名を特に指定しなければ、デフォルトの構成ファイル名 mfclisrv.cfg が使用されます。

アプリケーションごとに別の構成ファイルを作成することができ、また 1 つの構成ファイルに複数のエントリを入れることもできます。複数のエントリを入れる場合は、ファイル内で各アプリケーション固有のタグを指定し、その後にそのアプリケーションのパラメータ リストを続けます。クライアント プログラムでは、LNK-PARAM-BLOCK ブロックの LNK-TAGNAME フィールドを使用してこのタグ名を指定しなければなりません。この方法を使うと、デフォルトの構成ファイル mfclisrv.cfg 内に複数エントリを指定することで、コマンド ラインが長くならずに済みます。

クライアント/サーバー アプリケーションの設定や構成ファイルのエントリに誤りがあれば、プログラムはエラー内容を詳しく示すメッセージを画面に表示して終了します。

このようなエラーは画面に表示されるだけでなく mfclisrv.log ファイルにも記録されます。このファイルはカレントディレクトリに作成されるか、MFLOGDIR 環境変数で指定されたディレクトリに作成されます。このため、端末が直接つながっていないサーバーでもメッセージを記録できます。

2.5.1 構成ファイルのパラメータ

次に mfclisrv.cfg 構成ファイルのエントリ一覧をアルファベット順に示します。構成ファイルに指定されていないパラメータについてはデフォルト値が使用されます。

aaiservice=*name* PIC X(128) [default: MFCLISRV]

接続する AAI サービスの名前。AAI 管理システムの論理サービス名エントリです。

cblksize=*nnnn* PIC X(4) COMP-X [default: 0]

Dialog System の制御ブロック。Dialog System を使用する場合だけ必要です。

clierrprog=*name* PIC X(128) [default: none]

mfclientm に変わって通信エラーを処理するプログラム名。mfclient エラーを呼び出し側プログラムで処理するようにしたければ、SAME と指定します。

commsapi=*api* PIC X(4) [default: CCI]

通信 API。有効な値は、CCI または NONE。2 階層アプリケーションを作成して、通信プログラムを一切使わずに単一 PC でテストを行えるようにするには、特殊エントリの NONE を指定します。mfserver と通信要求がバイパスされ、データは mfclient とユーザー定義のサーバープログラムの間で直接送受信されます。

備考 : ユーザー定義のサーバープログラムとして既存アプリケーションを使用している場合は、このオプションを指定できません。

compress=*nnn* PIC 9(3) [default: 000]

圧縮ルーチン番号。例えば、001 ならば CBLDC001 というルーチンが使用されます。0 と指定するとデータ圧縮が行われません。

dblksize=*nnnn* PIC X(4) COMP-X [default: 0]

ユーザー定義のクライアント プログラムとサーバー プログラム間の、データの送受信で使用するユーザー データ領域のサイズ。Dialog System アプリケーションの場合は、この値が Dialog System データ ブロックのサイズになります。

警告:この値が、実際に使用しているデータ領域のサイズより小さければ、予期できないエラーが発生する可能性があります。

eblksize=*nnnn* PIC X(4) COMP-X [default: 0]

オプションの Dialog System イベント ブロックのサイズ。

machinename=*name* PIC X(34)

servername エントリで指定したサーバーが存在するマシン名。これを指定することで、システムがサーバーを捜す手間や同名のサーバーとの混乱が避けられます。

maxtrans=*nn* PIC 99 [default: 0]

転送パッケージの最大サイズをキロバイトで指定します。有効な値は、1～62 です。バッファ全体の大きさがこの値を超えると、システムはバッファを *maxtrans* サイズずつ分割して転送します。

midconfig=*filename* PIC X(128) [default: none]

クロス階層ソリューションの一環として、mfclient が mfserver に呼び出されたときに使用する構成ファイル名。このエントリを指定すると、mfserver がルータのように機能しローカルな mfclient モジュールにデータを渡します。その場合の mfclient モジュールはこの構成ファイルを使用して、別のサーバーを捜して通信します。この方法を使用すると、マシンごとにプロトコルと通信 API を変えることができます。

protocol=*protocol* PIC X(8) [default: CCITC32]

使用する CCI プロトコル。このエントリは、*commsapi* が CCI に設定されているときのみ有効です。サポートしている CCI プロトコルは、Novell IPX (CCIIX32 と指定)、NetBEUI (CCINB32 と指定)、動的データ交換 (CCIDE32 と指定)、TCP/IP (CCITC32 と指定)です。*commsapi* が CCI (デフォルト)に設定されているときに *protocol* を指定しなければ、クライアント/サーバー結合でデフォルトの TCP/IP (CCITC32) が使用されることに注意してください。

scrntype=*type* PIC X(4) [default: none]

例えば、ユーザー プログラムが GUI とキャラクタ インターフェイスの両方をサポートしている場合、このエントリを使用してクライアント プログラムがどちらのインターフェイスを使用するか指定できます。クライアント/サーバー結合モジュール自体は *scrntype* エントリを使用しません。

servername=*server* PIC X(14) [default: MFCLISRV]

通信で使用するサーバー名。

setenv=*name value* PIC X(148) [default: none]

どのサーバー プログラムの実行にも先だって環境変数を設定します。フォーマットは、

variable name PIC X(20)

variable value PIC X(128)

名前と値のフィールドは 1 つ以上の空白で区切ります。*setenv* エントリは 9 個まで指定できます。

srvanim=*x* PIC X(16) [default: N]

Y または N。このパラメータを Y にすると、サーバー上でユーザー プログラムのアニメーション表示が可能になります。すなわち、mfserver を一旦停止して、COBSW=+A として再起動しなくても動的に設定されます。

UNIX システムの場合は、x,*filename* と指定して、ユーザー プログラムのクロスセッションアニメーションを可能にできます。詳細については、*「アプリケーションのアニメート」* の項を参照してください。

srverrprog=*name* PIC X(128) [default: none]

mfserver に代わって通信エラーを処理するプログラム名。mfserver エラーのために *srvprog* エントリで指定したのと同じものにしたければ、SAME と指定します。

srvprog=*name* PIC X(128)

mfserver が呼び出すユーザー定義プログラムの名前。

srvtier=*name* PIC X(128) [default: mfserver]

サーバー階層プログラムの名前。これが指定されると、mfserver はこのプログラムから呼び出されることになります。

subserver=*base-name* PIC X(14) [default: none]

サブサーバー プロセスとの通信で使用するベース名を指定します。デフォルトのサーバー名は mfclisrv なのでそのままだとサブサーバー名は mfclisrv00001、mfclisrv00002、...となります。このパラメータを newserv と指定すると、サブサーバー名は newserv00001、newserv00002、...となります。このパラメータを使用すると、ベース サーバー名を変えずにアプリケーション固有のサブサーバー名を使用できます。

timeout=*nnnn* PIC X(4) COMP-5 [default: 120 secs)

このパラメータは、システムのデフォルトのタイムアウト値より優先されます。タイムアウト値は 1/10 秒単位で指定します。このとき既に呼び出されているものについては、呼び出された時点で有効だったタイムアウト値がそのまま使用されます。

ublksize=*nnnn* PIC X(4) COMP-X [default: 0]

オプションのユーザー データ ブロックのサイズ。

useraudit=*x* PIC X [default: N]

Y または N。このパラメータを Y に設定すると、クライアントの接続と切断の詳細がログに記録されます。その場合、*日付*、*時刻*、*接続/切断インジケータ*、*サーバー名*、*マシン名*、*プロトコル*の情報がクライアントとサーバーの両方に記録されます。記録先はシステム ログ ファイルですが、詳細については、*「監査トレールの作成」* と *「システム ログ ファイル」* の項を参照してください。

2.5.2 構成ファイルで最低限必要なエントリ

構成ファイルで最低限必要なエントリは、次の要因によって異なります。

- 使用するアプリケーション
- 実行されるサーバー数
- 使用する通信プロトコル

どんな場合にも必要なエントリを次に示します。

dblksize データ転送で使用するユーザー データ領域のサイズ。

srvprog サーバー側の処理を行うためにmfserverが呼び出すプログラム名。これは、サーバー マシンで実行されるユーザー プログラム (すなわち、データ アクセスとアプリケーション ロジックを処理するプログラム) です。

複数サーバーを実行する場合は、*servername* エントリを使用してサーバー名を指定しなければなりません。

CCI を使用しない場合は *commsapi* エントリを設定しなければなりません。また、CCI を使用しても TCP/IP 以外を使用する場合は *protocol* エントリを設定しなければなりません。

構成ファイルに設定されていないパラメータについては、それぞれ固有のデフォルト値が使用されます。

2.5.3 構成ファイルを置く場所

構成ファイルはユーザーに一番便利な場所に置くことができます。mfclient と mfserver の 2 つのプログラムは構成ファイルを次の方法で見つけ出します。

1. MFCSCFG 環境変数にファイル名が設定されているか調べます。
2. コマンドラインの中にファイル名が指定されていれば、MFCSCFG 環境変数で指定されたものより優先します。
3. 上のどちらの方法でもファイル名が指定されていなければ、デフォルト名の構成ファイル mfclisrv.cfg をカレントディレクトリから捜します。
4. mfclisrv.cfg が見つからなければ、構成ファイルの各エントリに対するデフォルト値が使用されます。

構成ファイル名をどの方法で指定する場合でも、場所と名前を合わせて 128 文字までの完全パス名で指定できます。

2.6 mfclient に対するユーザーのクライアント プログラムの接続

mfclient と mfserver のモジュールは互いに情報を受け渡すときに、mfclisrv.cpy コピー ファイル内のパラメータ ブロックを使用します。この 2 つのモジュールはユーザー プログラムに情報を渡すときもこれと同じパラメータ ブロック (LNK-PARAM-BLOCK) を使用します。

ユーザーのプログラムに次の例のようなコードを追加して、mfclient にパラメータを渡せるようにします。

```
$SET ANS85

 WORKING-STORAGE SECTION. ......

*--- mfclisrv.cpy must be included in the working storage
*--- section of the client program and in the linkage
```

```
*--- section of the server program.

 COPY "MFCLISRV.CPY".

 LINKAGE SECTION. ......

*--- Input-Rec is the area used for transfering data
*--- between the user client and the server programs.
*--- The size of this data area is defined by the user
*--- in the Client/Server Binding configuration file
*--- which is read by the mfclient module. Mfclient
*--- sets up the required memory for this area and
*--- returns a pointer to this area back to the user
*--- client program (see below).

 01 INPUT-REC            PIC X(32767)

 PROCEDURE DIVISION.

 CLIENT-CONTROL SECTION.

    PERFORM UNTIL END-CONNECTION

*--- lnk-client holds the name "mfclient".
*--- The first time through we initialize mfclient and establish
*--- contact with the server.

       CALL LNK-CLIENT USING LNK-PARAM-BLOCK

       EVALUATE TRUE

*--- Make the user data area (Input-Rec) accessible by assigning
*--- the address returned by mfclient.

         WHEN START-CONNECTION
           SET ADDRESS OF INPUT-REC TO LNK-DBLOCK-PTR

         WHEN END-CONNECTION
           EXIT PERFORM

         WHEN OTHER

*--- Perform your application client logic. For example, display
*--- and accept user interface data.

            ......
```

```
        END-EVALUATE

*--- Set a user defined flag (eg. EXIT-FLG-TRUE) to indicate
*--- that client processing has terminated. For example, you
*--- may have clicked on the EXIT push button on your application
*--- interface.

        IF EXIT-FLG-TRUE
           SET CLIENT-ENDING TO TRUE
        END-IF

    END-PERFORM.

 CLIENT-CONTROL-END.
    STOP RUN.
```

1 つのアプリケーションのクライアント数を制御したり、エラー メッセージの表示を自分のプログラムで行いたい場合は、ユーザー プログラムの最初の EVALUATE 文に次のようなコードを追加する必要があります。

```
          WHEN TOO-MANY-CLIENTS
             PERFORM OVER-CLIENT-LIMIT

          WHEN COMMS-ERROR
             PERFORM SHOW-ERROR

          ......

 OVER-CLIENT-LIMIT SECTION.
    DISPLAY SPACES AT 0101 WITH BACKGROUND-COLOR 7
       "MAXIMUM NUMBER OF CLIENTS EXCEEDED - SESSION ENDED"
       AT 1012 WITH FOREGROUND-COLOR 4
    SET EXIT-FLG-TRUE
    SET CLIENT-ENDING TO TRUE
    EXIT.



 SHOW-ERROR SECTION.
    DISPLAY LNK-ERROR-LOC AT 2201
    DISPLAY LNK-ERROR-MSG AT 2301 WITH SIZE LNK-ERROR-MSG-LEN.
    EXIT.
```

非同期要求を処理する場合は、EVALUATE 文に次のようなコードを追加しなければなりません。

備考 : AAI を使用している場合は、ASYNC-REQUEST は不要です。

```
WHEN START-CONNECTION
   PERFORM GET-USER-INPUT
   IF MAKE-ASYNC-REQUEST <* user asynchronous option
      SET ASYNC-REQUEST TO TRUE
   END-IF


WHEN ASYNC-OK
   SET TEST-ASYNC-RESULT TO TRUE
   PERFORM DELAY-LOOP


WHEN ASYNC-INCOMPLETE
   DISPLAY "REQUEST STILL BEING PROCESSED" AT 1010
   PERFORM DELAY-LOOP
   SET TEST-ASYNC-RESULT TO TRUE


WHEN RESULT-OK

   DISPLAY "REQUEST-COMPLETED " AT 1010

   PERFORM GET-USER-INPUT


WHEN ASYNC-NOT-STARTED
WHEN ASYNC-FAILED
   DISPLAY "ASYNCHRONOUS REQUEST FAILURE " AT 1010
   PERFORM SHOW-ERROR
   PERFORM GET-USER-INPUT


WHEN COMMS-ERROR
   PERFORM SHOW-ERROR
```

2.7 mfserver に対するユーザーのサーバー プログラムの接続

mfclient と mfserver のモジュールは互いに情報を受け渡すときに、mfclisrv.cpy コピー ファイル内のパラメータ ブロックを使用します。この 2 つのモジュールはユーザー プログラムに情報を渡すときもこれと同じパラメータ ブロックを使用します。

ユーザーのサーバー プログラムでは mfclisrv.cpy コピー ファイルを LINKAGE SECTION にインクルードします。また、PROCEDURE DIVISION のヘッダに LNK-PARAM-BLOCK パラメータブロックを指定して mfserver からパラメータを受け取れるようにします。

次のサンプル サーバー コードで説明します。

```
 LINKAGE SECTION.

 01 INPUT-REC              PIC X(32767).

 COPY "MFCLISRV.CPY".*--- lnk-param-block is the record definition in mfclisrv.cpy

 PROCEDURE DIVISION USING LNK-PARAM-BLOCK.

 CONTROLLING SECTION.

*----------------------------------------------------------------*

*  associate the user data area with the pointer in

*  lnk-parm-block.

*----------------------------------------------------------------*

    SET ADDRESS OF INPUT-AREA TO LNK-DBLOCK-PTR.

       EVALUATE TRUE

          WHEN START-CONNECTION
             PERFORM PROGRAM-INITIALIZE

          WHEN OTHER
             PERFORM PROGRAM-BODY

       END-EVALUATE.
       EXIT PROGRAM.

 PROGRAM-INITIALIZE SECTION.

*--- This code would include the initialization code
```

```
*--- for your server program eg. opening your application
*--- data files.

    ......

 PROGRAM-BODY SECTION.

*--- This section would include your application
*--- processing code eg. read and writing of data
*--- files.

    ......
```

エラー メッセージの表示を自分のプログラムで行う場合は、プログラムの最初の EVALUATE 文に次のようなコードを追加します。

```
         WHEN COMMS-ERROR
            PERFORM SHOW-ERROR

         ......

 SHOW-ERROR SECTION.
    DISPLAY LNK-ERROR-LOC AT 2201
    DISPLAY LNK-ERROR-MSG AT 2301 WITH SIZE LNK-ERROR-MSG-LEN.
    EXIT.
```

2.8 クライアント/サーバー結合アプリケーションの実行

モジュールは .int コード形式で提供されているので、そのまま実行するか .gnt コードを生成します。

UNIX システムで Micro Focus COBOL 3.2 以降を使用している場合は、この .int モジュールを UNIX に移し、他のプログラムとリンクして実行可能モジュールを作成できます。

mfclient と mfserver の各モジュールは、それぞれクライアントとサーバーの各コンポーネントを準備して標準の COBOL プログラムとして手動で実行します。

クライアント/サーバー結合を実行するには、

1. CCI 通信 API を使用している場合には、サーバー プログラムを起動します。AAI を使用している場合には、クライアントがサービスを要求したときに AAI がプログラムを自動的に起動するので、この操作手順は不要です。
2. クライアント プログラムを起動します。

UNIXの場合は次のようなコマンド ラインでサーバー プログラムを起動します。

```
cobrun mfserver [-b] [-p protocol] [-s server-name] [-v]
```

Windows の場合は次のようになります。

```
run mfserver [-b] [-p protocol] [-s server-name] [-v]
```

ここで、

-b	サーバーをバックグラウンド プロセスとして実行します。UNIX システムの場合は、コマンド ラインの最後に '&' を付けるのと同じです。サーバーからのエラー メッセージはログ ファイル (mfclisrv.log) に書き込まれます。
-p *protocol*	通信プロトコルを指定します。
-s *server-name*	デフォルト (mfclisrv) 以外のサーバー名を指定します。
-v	mfserver のバージョン番号を表示します。

クライアント プログラムを起動するコマンド ラインは次のようになります。

```
run program-name [config-filename]
```

ここで、

program-name	ユーザー インターフェイス プログラム名を指定します。
config-filename	使用する構成ファイル名を指定します。

2.9 アプリケーションのアニメート

NetExpress の標準アニメート機能を使用して、クライアント プログラムをアニメートできます。

サーバー プログラムをアニメーションにする場合は、構成ファイルで srvanim=y というパラメータ設定を行います (*「構成ファイルのパラメータ」* の項を参照してください)。

PC サーバーの場合、srvanim=y の設定になっていれば、クライアントがサーバーに接続すると自動的に Animator が起動します。

UNIX サーバーの場合は、mfserver を起動した端末で Animator が実行されるので、mfserver がフォアグラウンド モードで実行されていなければなりません。mfserver はバックグラウンド プロセスとして実行されるのが望ましいので、これは問題です。さらに、同時に一人のユーザーしか mfserver からアニメーションを表示できないという制約があります。この問題を解決するには、srvanim=x,*filename* と設定します。ここで、*filename* は touch コマンドで作成したファイルです。Animator の出力を表示する端末では、COBANIM_2=animator と設定してから、anim *filename* コマンドを実行します。標準入出力として使用したい端末では、構成ファイルに srvanim=x,*filename* というパラメータ設定を追加してから、通常の方法でアプリケーションを実行します。

2.10 サーバーの管理

CCI 通信 API を使用している場合は、mfcsmgr プログラムを実行して必要なパラメータと値を渡すことで、サーバーの動作を一時的に変更できます。指定できるパラメータはロケーションとアクションの 2 つのグループに分けられます。ロケーション グループのパラメータは、ターゲット サーバーを指定するためのもので、m、p、sの設定があります。アクティブ グループのパラメータはターゲット サーバーの動作に関するもので、a、c、o、r、t の設定があります。パラメータの中には o、r、t のように単独でしか指定できないものがあります。

AAI 通信 API を使用している場合は、サーバーの動作は AAI 管理システムを通じて管理されます。mfcsmgr プログラムは、AAI の管理下で実行されているアプリケーションには作用しません。

2.10.1 mfserver のシャットダウン

生成されたサーバーは、クライアントが正常終了するときにクライアント プログラムによって終了されます。CCI 通信 API を使用している場合は、最初のサーバー プログラムはクライアントの終了時に終了されないため、最初のサーバーを終了させるには手動でシャットダウンしなければなりません。AAI 通信 API を使用している場合は、最初のサーバー プログラムはクライアントの終了時に自動的に終了します。

2.10.2 許可パスワードの管理

CCI 通信 API を使用する場合は、アクティブなサーバーを誤ってシャットダウンしてしまうことがないように各サーバーにパスワードを割り当てることができます。そうすると、サーバーを終了したりサーバーのパラメータを変更するときにパスワードが必要になります。

2.10.3 最大クライアント数の設定

CCI 通信 API を使用する場合にサーバーがサポートできる最高クライアント数はデフォルトで 25 ですが、mfcsmgr を使用してより少なく設定できます。そこで指定した値はパスワード ファイルに保存され、サーバーが起動するたびに読み込まれます。

2.10.4 サーバー オーバライドの実行

サーバーがサポートしているオーバライド オプションは、*server-name*、*protocol*、*machine-name* です。サーバーは構成ファイルを使用しないので、オーバライド パラメータは mfcsmgr プログラムで指定します。このプログラムを実行すると、指定されたオーバライドを処理するためにクライアントとサーバー間で短いやり取りが起こります。

mfcsmgr のコマンドライン構文を次に示します。

Windows

```
run mfcsmgr    [-a] [-c nnnnn] [-d] [-i filename][- m machine-name]
               [-p protocol] [-s server-name] [-o m, machine-name]
```

```
                [-o p, protocol] [-o r] [-o s, server-name] [-t] [-v]
```

UNIX

```
cobrun mfcsmgr  [-a] [-c nnnnn] [-d] [-i filename] [- m machine-name]
                [-p protocol] [-s server-name] [-o m, machine-name]
                [-o p, protocol] [-o r] [-o s, server-name] [-t] [-v]
```

ここで、

-a	ターゲット サーバーの許可パスワードを変更します。
-c,*nnnnn*	サーバーがサポート可能なクライアント数を設定します。
-d	クライアントの終了時に、ローカルなオーバライド ファイル (mfcsovrd.cfg) を削除します。
-i *filename*	クライアントの終了時に、指定のファイルをクライアントのローカル ディレクトリにインストールします。
-m *machine-name*	ターゲット サーバーを実行しているマシン名を指定します。この指定は、複数のプラットフォームに同名のサーバーが存在する場合に必要になります。
-p *protocol*	使用する通信プロトコルを指定します。TCP/IP (CCITC32)、NetBEUI (CCINB32)、Novell IPX (CCIIX32)、動的データ交換 (CCIDE32) のどれかです。
-s *server-name*	デフォルト (mfclisrv) 以外のサーバー名を指定します。
-o m,*machine-name*	オーバライドを実行するマシン名を指定します。
-o p,*protocol*	オーバライド サーバーへのアクセスにはこのプロトコルを使用します。
-o r	現在アクティブなオーバライドをリセットし、元の設定に戻します。
-o s,*server-name*	ターゲット サーバーへの接続をこのサーバーへの接続にオーバライドします。
-t	サーバーを終了します。
-v	mfclient のバージョン番号を表示します。

上記のフラグは - または / を付けて指定します。大文字小文字は区別しません。

2.11 高度なトピック

2.11.1 システム ログ ファイル

mfclient と mfserver はエラーとメッセージのログを維持管理します。ログ エントリはすべて日付と時刻を付けて mfclisrv.log ファイルに保存されます。このログ ファイルはプログラムを起動したディレクトリ、または MFLOGDIR 環境変数で指定したディレクトリにあります。ログ ファイルをときどきチェックして、システムが正常に機能していることを確認してください。

2.11.2 監査トレールの作成

クライアントがサーバーに接続したときの日付と時刻を記録した監査トレールを作成できます。それには、構成ファイルに useraudit=y という設定を入れます。

監査トレールの情報は前項で説明したシステム ログ ファイルに記録されます。

2.11.3 構成ファイルの各エントリのオーバライド

クライアント/サーバー結合では、各クライアントの起動時 (ただし構成ファイルを読み込んだ後) に mfcsovrd.cfg というオーバライド ファイルを捜します。このファイルが見つかると、その内容を調べ、構成ファイルに従って設定されているパラメータをオーバライドします。オーバライド ファイルの形式は通常の構成ファイルとほぼ同じですが、override-cntrl というエントリが 1 つだけ追加されています。このエントリでは、オーバライドの対象を、サーバー名とタグ名のどちらで指定するのかを指示します。ここにサーバー名を指定すると、そのサーバーを使用するクライアントすべてのパラメータがオーバライド ファイルの内容に従って変更されます。タグ名を指定すると、そのタグ名を使用するクライアントのパラメータだけがオーバライドされます。

このようなクライアント オーバライド機能を使用するのは、例えばサーバーが使用不可能になったのでアプリケーションを別のマシン上で実行しなければならない場合です。もちろん個々の構成ファイルを変更する方法もありますが、オーバライド ファイルを使用すれば、1 つのファイルだけですべてのアプリケーションの接続先を変更できます。

このオーバライド機能はサーバーでも使用できますが、その場合はサーバー マシンが立ち上がって稼動していなければなりません。この場合もサーバー名またはタグ名でオーバライドできます。

オーバライド ファイルが見つかると、そのことがログ ファイルに記録されます。同時にオーバライドの対象となるパラメータもすべて記録されます。

オーバライド機能を使用して、クライアントの接続先サーバーを変更する例を次に示します。

```
[override-cntrl]
override=servername
[oldserver]
```

```
servername=newserver
```

上の例では、[override-cntrl] セクションでオーバライドの対象とするサーバー名を指定 (override=servername) し、そのサーバー名 ([oldserver]) の下に新たな接続先サーバー名を指定 (servername=newserver) しています。この場合、ログ ファイルには次のように記録されます。

```
20/04/1998 11:01:02 Using Local File: mfcsovrd.cfg
Overriding Entries for Servername:OLDSERVER
servername=newserver
20/04/1998 11:01:02 Override Completed:
```

特定のタグを使用する、すべてのクライアントのサーバーをオーバライドする例を次に示します。

```
[override-cntrl]
override=tagname
[mf-clisrv]
servername=newserv
```

上の例では、[override-cntrl] セクションでオーバライドの対象とするタグ名を指定 (override=tagname) し、そのタグ名 ([mf-clisrv]) の下にオーバライドするパラメータ (この場合はサーバー名) を指定 (servername=newserv) しています。この場合、ログ ファイルには次のように記録されます。

```
20/04/1998 11:04:02 Using Local File: mfcsovrd.cfg
Overriding Entries for Tagname:MF-CLISRV
servername=newserv
20/04/1998 11:04:02 Override Completed:
```

2.11.4 Dialog System 「コールアウト」の使用

クライアント/サーバー結合で Dialog System のコールアウト機能を使用したい場合には、ユーザー クライアント プログラムを変更して、接続のどちらか一方でコールアウト要求をサービスするプログラムを作成する必要があります。次のように、ユーザ クライアント プログラムに 2 行および指令を追加してください。

プログラムの最初に、次のように追加してください。

```
$set ans85 linkcount(128)
```

WORKING-STORAGE に、次のように追加してください。

```
 01  lnk-param-block-addr        POINTER EXTERNAL.
```

PROCEDURE DIVISION に、次のように追加してください。

```
SET lnk-param-block-addr TO ADDRESS OF lnk-param-block
```

このように追加することで、コールアウト要求を扱うダイアログに呼び出されるプログラムが、ユーザー クライアント プログラムと同じパラメータ エントリにアクセスします。これは、外部データ ポインタを使用することで実現されます。コールアウト プログラムには、次の例のようなコードが必要です。

```
$set ans85
 WORKING-STORAGE SECTION.

 01  lnk-param-block-addr        POINTER EXTERNAL.

 LINKAGE SECTION.

 COPY "mfclisrv.cpy".

 01  user-data-block             PIC X(30).

 PROCEDURE DIVISION.
 Controlling SECTION.
    SET ADDRESS OF lnk-param-block TO lnk-param-block-addr
    SET ADDRESS OF user-data-block TO lnk-ublock-ptr
    SET ds-callout TO TRUE
    MOVE "callosrv" TO user-data-block
    CALL lnk-client USING lnk-param-block
    EXIT PROGRAM.
```

サーバーでコールアウト要求を受け取るプログラムの名前は、user-data-block に設定されます。このプログラムの名前が、Dialog System に使用されているコントロール ブロックのいずれかのデータからだけで決定される場合には、次のプログラムに類似のコーディングを使用してこれらのブロックをアクセス可能にしてください。

```
$set ans85 linkcount(128)
 WORKING-STORAGE SECTION.

 01  lnk-param-block-addr        POINTER EXTERNAL.

 LINKAGE SECTION.

 COPY "mfclisrv.cpy".

 01  user-data-block            PIC X(30).

    COPY "DS-CNTRL.MF".
    COPY "CUSTOMER.CPB".



 PROCEDURE DIVISION USING ds-control-block customer-data-block.
```

```
Controlling SECTION.
    SET ADDRESS OF lnk-param-block TO lnk-param-block-addr
    SET ADDRESS OF user-data-block TO lnk-ublock-ptr
    SET ds-callout TO TRUE
    EVALUATE cust-callout-flg

      WHEN 1
        MOVE "callprog1" TO user-data-block
      WHEN 2
        MOVE "callprog2" TO user-data-block
    END-EVALUATE
    CALL lnk-client USING lnk-param-block
    EXIT PROGRAM.
```

サーバー側のコールアウト プログラムは、次のようになります。

```
$set ans85

LINKAGE SECTION.

    COPY "DS-CNTRL.MF".
    COPY "CUSTOMER.CPB".

PROCEDURE DIVISION USING ds-control-block customer-data-block.
Controlling SECTION.
    ............ Code to
       ........... process callout
          ............ request....
    EXIT PROGRAM.
```

2.11.5 既存の非 Dialog System プログラムをサーバー モジュールとして使用する

クライアント/サーバー結合では、既存の Dialog System プログラムをユーザー サーバー モジュールとして使用することが可能です。これは、`srvtier=<your program>` の時に行われます (詳細は、*「構成ファイルのパラメータ」* を参照してください）。

このインスタンスでは、Dialog System API に準拠するパラメータのセットで mfserver が呼び出されます。このように使用されると、ユーザー プログラムは mfserver を呼び出してクライアントにデータを転送し、サーバーの制御要素になります。Dialog System で使用されている API に準拠していれば、その他のプログラムでも同じ機能を行うことができます。

クライアント/サーバー結合では、通常、API の最初のパラメータ内で特定の項目を検査し、見つけた値に基づいて特定の機能を行います。この機能を行わないようにするには、これらの項目を HIGH-VALUES に設定します。次の 2

つのパラメータで mfserver を呼び出してください。

- 最初のパラメータは Dialog 制御ブロックを表し、長さは 6～399 バイトです。6 番目のバイトが検査されるので、Dialog System への呼び出しが行われないように、HIGH-VALUES に設定してください。このパラメータのその他の部分は必要に応じて使用、または無視してください。
- 2 番目のパラメータはユーザー データ ブロックです。このパラメータのサイズの制限はありません。

これらの項目のサイズは両方とも、アプリケーションの実行に使用する構成ファイルで定義してください。デモ プログラムと必要な構成ファイルの例を次に示します。

o nodscli.cbl - The client module

```
$set ans85
 WORKING-STORAGE SECTION.
 01  w-prog-id              pic x(30) value
     '"@(#) nodscli.cbl 1.1.1"'.

 copy "mfclisrv.cpy".
 01  ws-scrn-pos.
     03  ws-line            pic 99 value 07.
     03  ws-col             pic 99 value 1.

 LINKAGE SECTION.
 01  ds-control-block        pic x(6).

 01  NODS-DATA-BLOCK.
     03  NODS-PARAM          PIC 99.
     03  NODS-DATA           PIC X(120).

 procedure division.
 000-control section.
* Make initial contact with the server
    call lnk-client using lnk-param-block
    if start-connection
       set address of ds-control-block to lnk-cblock-ptr
       set address of nods-data-block to lnk-dblock-ptr
    end-if

    display spaces at 0101
         "Starting Test Session" at 0510
    perform 010-run-test until nods-param = 10

* Send disconnect request
```

```
      set client-ending to true
      call lnk-client using lnk-param-block
      display "Test Session Completed" at 2010
      stop run.

   010-run-test section.

  * Get data from server
      call lnk-client using lnk-param-block
      add 1 to ws-line
      display nods-data(1:80) at ws-scrn-pos
      exit.
```

o nodssrv.cbl - The server module

```
  $set ans85
   working-storage section.
   01  w-prog-id               pic x(30) value
       '"@(#) nodssrv.cbl 1.1.1"'.

   01  dummy-cntrl-block       pic x(6) value high-values.

   01  NODS-DATA-BLOCK.
       03  NODS-PARAM          PIC 99.
       03  NODS-DATA           PIC X(120).

       copy "mfclisrv.cpy".

   01  ws-sub                  pic 99.
   01  ws-table.
      03  ws-letts             pic x occurs 10.

   procedure division.
   controlling section.

      move "ABCDEFGHIJ" to ws-table
      move 0 to nods-param

  *
  * Establish contact with the client.
  * This needs to be done as early as possible to stop the
  * Client timing out and 'polling' the network looking for
  * the server.
  *
```

```
           call "mfserver" using dummy-cntrl-block nods-data-block
      *
      * You must ensure that you don't call 'mfserver' after the
      * client has issued a shutdown request. In this system
      * nods-param = 10 causes to client to issue the shutdown
      * request and ends the driving loop in this program.
      *
           perform varying nods-param from 1 by 1
              until nods-param > 10
                 perform varying ws-sub from 1 by 1
                   until ws-sub > 80
                      move ws-letts(nods-param)
                        to nods-data(ws-sub:1)

                 end-perform
                 call "mfserver" using dummy-cntrl-block
                                       nods-data-block

           end-perform
           stop run.
```

o nods.cfg - The configuration file

```
*****************************************************************************
* Micro Focus - クライアント サーバー モジュール構成ファイル                  *
*****************************************************************************

[mf-clisrv]
srvtier=nodssrv
cblksize=6
dblksize=122
servername=nods
```

もし、このサーバープログラムが呼び出しプログラムであり、データをクライアントに渡すたびに「EXIT PROGRAM」を使用しなければならない場合には位置を失いますが、この場合は、このサーバー プログラムをコンパイルして実行すると、位置を失わずにクライアントにデータを渡します。既存のプログラムが、必要な Dialog System API に準拠できない場合でも、次に示すブリッジ プログラムを導入することで、上記の方法を使用することができます。

```
       WORKING-STORAGE SECTION.
```

```
01  dummy-control-block        PIC X(6) VALUE HIGH-VALUES.
01  grouped-data-block.
   03  ws-param-1             PIC X(50).
   03  ws-param-2             PIC X(120).
   03  ws-param-3             PIC X(18).
   03  ws-param-4             PIC X(48).

LINKAGE SECTION.

01  lnk-param-1               PIC X(50).
01  lnk-param-2               PIC X(120).
01  lnk-param-3               PIC X(18).
01  lnk-param-4               PIC X(48).

PROCEDURE DIVISION USING

       lnk-param-1 lnk-param-2
       lnk-param-3 lnk-param-4.

   MOVE lnk-param-1 TO ws-param-1
   MOVE lnk-param-2 TO ws-param-2
   MOVE lnk-param-3 TO ws-param-3
   MOVE lnk-param-4 TO ws-param-4

   CALL "mfserver" USING dummy-control-block
                         grouped-data-block



   MOVE ws-param-1 TO lnk-param-1
   MOVE ws-param-2 TO lnk-param-2
   MOVE ws-param-3 TO lnk-param-3
   MOVE ws-param-4 TO lnk-param-4

   EXIT PROGRAM.
```

2.11.6 インライン構成

クライアント/サーバー結合の強力な機能の1つとして、構成ファイルで通信条件を制御できる点があります。ただし、エンドユーザーが構成ファイルのエントリを変更するだけでアプリケーションの動作が変わってしまうのは望ましくない場合があります。

そこで、構成パラメータをすべてクライアント プログラム内で指定する方法があります。そうすると構成ファイルが一切使用されないため、エンドユーザーはアプリケーションの動作を変更できません。その場合もユーザーが複雑な通信コードを書く必要はなく、指定したいパラメータをクライアント プログラム内で簡単に指定できます。パラメータの指定は、load-inline-cfg と end-inline-cfg の 2 つの設定の間で行います。パラメータ エントリは 1 つずつ lnk-error-msg にロードし、mfclient を呼び出して処理します。パラメータの設定は実際の処理ループに先だって行わなければなりません。インライン設定の例を次に示します。

```
WORKING-STORAGE SECTION.

COPY "mfclisrv.cpy".

LINKAGE SECTION.

01 INPUT-REC              PIC X(32767)

PROCEDURE DIVISION.

Client-Control SECTION.

   SET load-inline-cfg TO TRUE
   MOVE "clierrprog=same" TO lnk-error-msg
   CALL lnk-client USING lnk-param-block

   MOVE "srverrprog=same" TO lnk-error-msg
   CALL lnk-client USING lnk-param-block

   MOVE "servername=mainserv" TO lnk-error-msg
   CALL lnk-client USING lnk-param-block

   SET end-inline-cfg TO TRUE

*  The main loop is repeated until the connection with
*  the server ends

   PERFORM UNTIL End-Connection

*  'lnk-client' holds the name 'mfclient'
*  The first time through we initialize the system and establish
*  contact with the server.

      CALL lnk-client USING lnk-param-block


      EVALUATE TRUE
```

```
           WHEN start-connection

       ........ The rest of the program is standard from this
       ........... point onwards  ............
```

外部の構成ファイルとインライン設定の両方を使用することもできます。そうすると、エンドユーザーも必要に応じてシステムをある程度制御でき、またクライアント プログラムも最終的な制御権を握っているという両方の利点が得られます。構成ファイルが先に処理され、次にインライン設定が処理されます。したがって、構成ファイルで不適切なパラメータ設定が行われてもインライン設定で上書きできます。この方法を使用するときは、インライン設定を use-combined-cfg と end-inline-cfg の 2 つの設定の間で行います。

```
      WORKING-STORAGE SECTION.

      COPY "mfclisrv.cpy".

      LINKAGE SECTION.

      01 INPUT-REC              PIC X(32767)

      PROCEDURE DIVISION.

      Client-Control SECTION.


         SET use-combined-cfg TO TRUE
         CALL lnk-client USING lnk-param-block

         SET load-inline-cfg TO TRUE
         MOVE "servername=mainserv" TO lnk-error-msg
         CALL lnk-client USING lnk-param-block

         SET end-inline-cfg TO TRUE

      *  The main loop is repeated until the connection with
      *  the server ends

         PERFORM UNTIL End-Connection

      *  'lnk-client' holds the name 'mfclient'
      *  The first time through we initialize the system and establish
      *  contact with the server.


            CALL lnk-client USING lnk-param-block
```

```
        EVALUATE TRUE
           WHEN start-connection

      ........ The rest of the program is standard from this

      ........... point onwards  ............
```

2.11.7 縮小データ転送 (RDT) 機能

クライアント/サーバー結合では、構成ファイルで指定された大きさのデータ ブロックを格納できるバッファが割り当てられます。クライアント プログラムとサーバー プログラムの間で制御が移動するたびに、このサイズのバッファが送受信されます。そのため、ネットワークに不必要に大きな負荷がかかる場合があります。例えば、22K のレコード領域を持つ 24K のバッファが割り当てられた場合を考えてみましょう。ファイルが 10 バイトのレコードキーを持つ場合、クライアントとサーバーの間でキーをやり取りするには 10 バイトのみ必要ですが、24K のバッファが使用されます。実際にはデータ サイズより 1〜2 バイト余分に必要ですが、それでもバッファ全体は大きすぎます。そこで、ネットワーク上を転送されるデータ サイズを必要な大きさに限定するための縮小データ転送 (RDT) 機能が提供されています。この機能は、CCI 通信 API を使用している場合だけに利用可能です。AAI 通信 API を使用している場合には、AAI API が自動的にデータを圧縮する機能を持っているため、この機能は不要です。AAI 通信 API とともに RDT 機能を使用すると、予期しない結果が生じる場合があります。

RDT 機能を使用するには、制御フラグ (use-rdt) と次の 3 つのパラメータが必要です。

lnk-usr-fcode ユーザー機能インジケータ。受信側にこのデータの処理方法を示します。

lnk-usr-retcode Theバッファ先頭ポイント。4 つのデータ領域のどれが転送開始ポイントか指定します。有効な値は、1=cblock、2=dblock、3=eblk、4=ublk です。11〜14 も同じアドレス領域を示しますが、この場合はデータが圧縮されていることを示します。データ圧縮を使用するには構成ファイルで指定しておく必要があり、そうでないとデフォルトオプション (圧縮なし) になります。0 と指定するとデータ転送が行われません。0 で NULL 操作を表せるので、IF 文を多用してさまざまな判定をする必要がなくコードがシンプルになります。NULL 操作は、ローカルに完結する処理のときにネットワークトラフィックをゼロにできるので、ネットワークの負荷軽減に大変有効です。

lnk-data-length 転送するデータサイズ。

インデックス ファイルに対してレコード (顧客詳細情報) の追加、削除、読み込みを行うアプリケーションを考えてみましょう。このインデックス ファイルのレコード キーは顧客コードです。このアプリケーションには、顧客レコードの操作だけでなく顧客情報をインターフェイス画面から消してしまうオプションもあります。このサンプル アプリケーションのコードの一部を次に示します。*user-data-block* 領域には、6 バイトのレコード キーが入ります。クライアントとサーバーの間で顧客詳細情報レコードをやり取りするために、データ ブロック領域 (*dblksize*) を使用しますが、これが *customer-data-block* です。*customer-data-block* 内の 6 バイトのデータ項目 customer-c-code にレコード キーが入ります。

クライアント側のコードは次のようになります。

```
           EVALUATE TRUE

             WHEN customer-load-flg-true

*---       User has entered customer code and selected the "LOAD"
*---       option on the interface to read and display the customer
*---       details relating to that code.

                MOVE customer-c-code TO user-data-block
                SET use-rdt TO TRUE
                MOVE 1 TO lnk-usr-fcode
                MOVE 4 TO lnk-usr-retcode
                MOVE 6 TO lnk-data-length

             WHEN customer-del-flg-true

*---       User has entered a customer code selected the DELETE
*---       option to delete the customer record from the file.

                MOVE customer-c-code TO user-data-block
                SET use-rdt TO TRUE
                MOVE 2 TO lnk-usr-fcode
                MOVE 4 TO lnk-usr-retcode
                MOVE 6 TO lnk-data-length
                initialize customer-data-block

             WHEN customer-clr-flg-true

*---       User has selected the CLEAR option to clear the current
*---       customer details from the screen.

                SET use-rdt TO TRUE
                MOVE 0 TO lnk-usr-retcode
                initialize customer-data-block
                PERFORM Set-Up-For-Refresh-Screen
           END-EVALUATE
```

CLEAR オプションが選択された場合、画面をクリアするのはローカルに完結する操作なので、サーバーに接続する必要がなく NULL 操作を使用しています。RDT 機能に対応するサーバー側のサンプルプログラムの一部を次に示します。クライアント/サーバー結合では *send-via-rdt* フラグがセットされるので、*lnk-usr-fcode* をチェックしなけれ

ばならないかわかります。サーバー側のコードは次のようになります。

```
           WHEN send-via-rdt
              EVALUATE lnk-usr-fcode
                 WHEN 1


*--- For the LOAD function the server program reads the customer
*--- details from the data file and sends the data back to the
*--- client using the data area customer-data-block rather using
*--- the RDT faclity.  Unless the RDT flag is set the client/server
*--- bindings will always pass the complete data area (defined by
*--- dblksize in the configuration file) between the client and the
*--- server.

                    MOVE user-data-block TO customer-c-code
                    SET customer-load-flg-true TO TRUE
                    PERFORM ....... rest of program.......
                 WHEN 2
                    MOVE user-data-block TO customer-c-code
                    SET customer-del-flg-true TO TRUE
                    PERFORM ....... rest of program.......
              END-EVALUATE
```

Csbind デモ用アプリケーションは RDT 機能を使用しています。

2.11.8 サーバーからのファイル管理機能

クライアント/サーバー ソリューションにつきまとう問題の 1 つとして、クライアント数の増加に伴うクライアント プログラムなどの、更新の煩雑さがあります。

オーバライド機能 (*「構成ファイルの各エントリのオーバライド」* の項を参照) を使用すると、各クライアントの構成ファイルを個別に書き換えなくても、メンテナンス中のサーバーから別のサーバーに変更できます。オーバライドはローカルでもリモートでも実行できますが、各クライアントのローカルなオーバライド ファイルをインストールしたり削除したりする手間がかかります。

mfcsmgr プログラムを使用すると、-i または -d オプションを使用して、クライアントにオーバライド ファイルをインストールまたは削除できます (*「サーバーの管理」* の項を参照)。インストール/削除はクライアント プログラムの終了時に行われます。

そこで、mfcsmgr プログラムで -i オプションを使用すればオーバライド ファイルだけでなく任意のファイルをクライアント システム (クライアントのローカル ディレクトリ) にインストールできます。すなわち、この方法で新

たなスクリーン セットや最新プログラム ファイルをインストールすれば、管理センターから各クライアントに更新データを配布できます。セキュリティ上の理由から削除オプションはこれほど強化されておらず、削除できるのはオーバライド ファイル (mfcsovrd.cfg) に限られます。

-i オプションを使用してインストールするファイルはサーバー上になければなりません。目的のファイルが mfserver を起動したディレクトリ内にある場合は、ファイル名だけを指定します。別のディレクトリにある場合は、完全パス名で指定する必要があります。例えば、/u/live/update/newprog.int、d:¥testprog.int、$LIVE/newtest.int はどれも有効ですが、$newfile という指定は無効です。

この機能を使用するのにプログラムの変更は必要ありません。クライアントにはファイルが転送されたことを示すメッセージが表示され、システム ログ ファイルに詳しい情報が書き込まれます。

2.12 クライアント/サーバー結合における制限事項

クライアント/サーバー結合には制限事項はほとんどありませんが、次の点に注意してください。

- Windows 95、Windows 98、Windows NT、UNIX プラットフォームでは、.int または .gnt 形式のクライアント/サーバー結合モジュールを実行できます。また UNIX の場合は、クライアント/サーバー結合モジュールをユーザーが書いたアプリケーション プログラムとリンクして、実行可能オブジェクトを作成することもできます。

- CCI 通信 API では、サポートされるクライアント数は 25 までです。AAI 通信 API でサポートされるクライアント数は、サーバーの能力、ネットワークプロトコル、またはエンドユーザーの性能要件によって制限される場合があります。たとえば UNIX では、mfserver が生成するサブプロセスの数によって制限が決まります。ユーザーが UNIX マシンにログオンすると、サポートできるサブプロセスの数に制限のある一意のプロセス ID がユーザーに割り当てられます。クライアントがクライアント/サーバー結合経由で接続すると、それらはすべてベース サーバーのサブプロセスとなります。もちろん、許可されるサブプロセス数を変更するか、または複数のベース サーバーを動作させることで、この制限は解決できます。UNIX マシンが数 100 人の直接ログイン ユーザーをサポートするように構成されている場合には、サブプロセス制限の問題が解決されると、このマシンで実行されているクライアント/サーバー結合は同数のクライアントをサポートすることができます。

- CCI 通信 API で実行している場合、クライアント/サーバー結合には回復機能がありません。ネットワークがダウンするとデータが失われます。クライアント/サーバーのどちら側でも障害発生を検知してログに記録しますが、データは回復できません。接続のどちらかのユーザー プログラムを終了させるような RTS エラーの場合も同様です。この点において、クライアント/サーバー結合は標準 RTS 以上のものは提供しません。AAI 通信 API は回復機能を必要とするアプリケーションのために提供されています。これらの機能の詳細については、AAI 製品に付属のマニュアルを参照してください。

索引

例