CICS® Transaction Gateway

C++ プログラミング

バージョン 4.0

SC88-8996-00
(英文原典：SC34-5945-00)

CICS® Transaction Gateway

C++ プログラミング

バージョン 4.0

SC88-8996-00
(英文原典：SC34-5945-00)

ご注意

本書の情報およびそれによってサポートされる製品を使用する前に、117ページの『特記事項』に記載する一般情報をお読みください。

本書は、CICS Transaction Gateway (プログラム番号 5724-A75) に適用されます。また、新版で特に明示されない限り、これ以降のすべてのバージョン、リリース、および修正レベルにも適用されます。

本書は、SC88-7779 の改訂版です。ページ左側の縦線は、この版で新しくなった部分を示しています。

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

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

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

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

(URL は、変更になる場合があります)

原　典： SC34-5945-00
CICS® Transaction Gateway
C++ Programming
Version 4.0

発　行： 日本アイ・ビー・エム株式会社

担　当： ナショナル・ランゲージ・サポート

第1刷 2001.6

目次

本書について

本書は CICS 外部呼び出しインターフェース (ECI) および CICS 外部表示インターフェース (EPI) のためのオブジェクト指向プログラミングについて説明します。 IBM CICS ユニバーサル・クライアント バージョン 4.0 で提供されているクラスとメソッドを使用した例を使って、プログラム作成の指針を提供します。

本書の対象読者

本書はオブジェクト指向 CICS クライアント・プログラムを開発するために、IBM CICS ユニバーサル・クライアント バージョン 4.0 で提供されたオブジェクト指向 (OO) クラスの使用方法を知りたい CICS アプリケーション・プログラマーを対象としています。

オブジェクト指向プログラムは、高水準で書かれているため理解しやすく、かつ保守が容易で、再使用に向いています。クラス・ライブラリーに組み込まれたコードによって、エラーを起こしやすいあるいは繰り返しやすいエレメントをプログラミングする作業からプログラマーが解放されるだけでなく、クラスそのものが CICS 機能を完全かつ適切に活用できるように書かれています。

クライアントは、外部呼び出しインターフェース (ECI) および 外部表示インターフェース (EPI) によって CICS サービスを使用することができます。CICS クライアント・クラスにより、C++ プログラマーはオブジェクト指向方式で ECI および EPI インターフェースにアクセスすることができます。

C++ は、このバージョンでサポートされるプログラミング言語です。

本書の直前の旧版、*「CICS ユニバーサル・クライアント C++ プログラミングの手引き」(SC88-7779-00)* は、CICS クライアントの旧バージョンにご使用ください。

本書で使用される規則と用語

本書では、*CICS* ユニバーサル・クライアントという用語は CICS Transaction Gateway のクライアント・コンポーネントを示します。

わかりやすくするために、「C」はプログラム言語 C と C++ の両方を示すために使用されます。 『CICS on System/390®』 は、以下の CICS サーバー製品を示すために使用されます。

- CICS® for MVS/ESA™
- CICS® Transaction Server for OS/390®
- CICS® Transaction Server for VSE/ESA™
- CICS/VSE®

CICS ユニバーサル・クライアント は、 Windows NT® および Windows 2000 で稼働します。本書で Windows® と示す場合には、Windows の特定のバージョンを指定していない限り、 Windows NT と Windows 2000 の両方を意味します。

OS/390 と示す場合には、OS/390 と z/OS オペレーティング・システムの両方を示します。

前提条件と関連情報

本書は、読者がオブジェクト指向の概念と C++ 言語に精通し、かつ CICS が提供する既存のサービスを適正に理解していることを前提としています。

CICS サービスを使用する際の詳しい情報は、「*CICS® ファミリー: クライアント / サーバー・プログラミング*」を参照してください。

変更の要約

CICS ユニバーサル・クライアント バージョン 4.0 で変更された機能は、以下のとおりです。

- TCP62 がサポートされています。
- 以下は、サポートされなくなりました。
  - COBOL
  - PL1
  - REXX
  - CICSTELD
  - ECI_VERSION_0
  - OS/2®、Windows 95、Windows 98
  - Windows オペレーティング・システム上の以下の機能
    - 16 ビット・アプリケーション
    - NetBIOS および DCE プロトコル
    - IBM VisualAge® C++ for Windows®
  - Solaris オペレーティング・システム上の以下の機能
    - SUNLINK

本書は、SC88-7779 の改訂版です。ページの左端に縦線のある個所が、本書で変更された部分です。

第1部 クライアント・クラス — 指針

第1章 オブジェクト指向プログラミングの紹介

CICS® ファミリーは、IBM® が提供する主なハードウェア・プラットフォーム、また UNIX® などの主要な非 IBM プラットフォーム全般にわたる、強力なトランザクション処理機能を備えています。また、クライアント / サーバーをサポートする幅広い機能を提供して、エンド・ユーザーに情報を提供する最新のグラフィカル・インターフェースの使用を可能にします。CICS ファミリーは、オブジェクト指向プログラミング用に出現するテクノロジーをサポートするようになっており、CICS ユーザーには、CICS 技術、データ、ならびにアプリケーションへの投資を活用しながら、オブジェクト・テクノロジーの利点の多くを利用する方途が提供されています。

オブジェクト指向プログラミングでは、現実性の高いモデルを、より柔軟性に富んだプログラミング言語で作成することができます。オブジェクトの新しい型もしくはクラスを定義できると同時に、これらのオブジェクトをさまざまな構造体を取り入れて表すことができます。

オブジェクト指向プログラミングを用いると、特定の型のオブジェクトに関連する動作を定義するメソッド (メンバー関数) を作成し、当該データに関連するセマンティックを多く捕らえることで、データにより多くの意味を関連づけることもできます。

ソフトウェアの複雑な部分を、より単純な外部シェルに包みこむか、カプセル化することで、コード再使用の手掛かりが得られます。このようにして定義されたオブジェクトは、広範囲のさまざまなアプリケーションが使用することができます。オブジェクトを適切に定義し個別に用意することが、その後のアプリケーションのより迅速かつ経済的な作成を可能にする、再使用可能なパーツのライブラリーの基礎となります。既存のパーツは、すでにテスト済みで、他のアプリケーションで使用されているため、その再使用によってより高いレベルのソフトウェア品質が得られることになります。

CICS クライアントでのオブジェクト指向サポート

CICS クライアントで提供される基本的な通信メカニズムは、外部呼び出しインターフェース (ECI) です。ECI の完全機能をオブジェクト指向の方法でモデル化する、ECI クラス・ライブラリーを用意することで、拡張サポート構築の基礎が与えられます。

クラスが提供されることで、サーバーへのリンク、サーバー上の CICS プログラムの呼び出し、状況の検出、そして作業単位 (UOW) の活用を行う機能が得られます。

CICS クライアントのアプリケーション・プログラマーが使用できるもう一つのインターフェースは、外部表示インターフェース (EPI) です。CICS アプリケーションにもとづく 3270 端末との通信は、端末、画面、フィールドおよび BMS マップをカプセル化するクラスによって提供されます。EPI クラスを用いると、3270 データ・ストリームを直接処理する必要がなくなり、出力画面の内容の検査、更新がさらに容易になります。

プログラミング言語サポート

オブジェクト指向ライブラリーは、C++ および BASIC プログラマー用の CICS ユニバーサル・クライアントと共に提供されます。本書では、C++ プログラミングのトピックを記載します。COM ライブラリーを使用したい場合は、「CICS Transaction Gateway COM オートメーション・プログラミング」を参照してください。

第2章 作業環境の確立

AIX®、HP-UX、Linux、Solaris、および Windows® オペレーティング・システムで使用する CICS クライアントには、 C++ (オブジェクト指向 (OO)) サポートが提供されています。このサポートには、クラス・ライブラリー、 C++ ヘッダー・ファイル、BMS マップ・ユーティリティー、およびサンプル・コードが含まれています。

CICS クライアントがサポートする CICS サーバー・プラットフォームの詳細については、各「*CICS Transaction Gateway: クライアント管理*」資料の『サポートされるソフトウェア』を参照してください。

第3章 CICS クライアント用 C++ クラスの使用

コンパイルとリンク

C++ プログラム・ソースには、CICSECI.HPP (ECI クラス用) か CICSEPI.HPP (EPI クラス用) を組み込むための、#include 文が必要です。これらのファイルは include サブディレクトリーにあります。

以下にリストされているディレクトリーには、ECI および EPI を呼び出すサンプル・プログラムが含まれています。また、これらのディレクトリーには、サンプル・プログラムのコンパイルおよびリンクに使用できるコマンド・ファイルまたは make ファイルも含まれています。これらのファイル内のコマンドは、アプリケーションのコンパイル方法およびリンク方法を示す例としても使用できます。

- IBM CICS ユニバーサル・クライアント (Windows 版)®:
  - ...¥IBM CICS Transaction Gateway¥SAMPLES¥C
  - ...¥IBM CICS Transaction Gateway¥SAMPLES¥CPP
- CICS ユニバーサル・クライアント (AIX 版)®:
  - /usr/lpp/ctg/samples/c
  - /usr/lpp/ctg/samples/cpp
- CICS ユニバーサル・クライアント (HP-UX 版)、CICS ユニバーサル・クライアント (Linux 版)、CICS ユニバーサル・クライアント (Solaris 版):
  - /opt/ctg/samples/c
  - /opt/ctg/samples/cpp

マルチスレッド化

CICS クライアント C++ ライブラリーは全体としては「スレッド・セーフ」ではありません。 2 つのスレッドによる 1 つのオブジェクトの同一インスタンスへの同時更新を妨ぐための、クリティカル・セクションまたはセマフォーを持っていません。しかし、データを共有しないクラスは、「作法に従った」 マルチスレッド・プログラムで使用することができます。通常の技法は、それぞれのスレッドが CclConn、CclFlow、CclBuf (軽いオブジェクト) などの独自のインスタンスを持っています。

例外処理

ほとんどのクラス・メソッドは例外を生成することがあります。デフォルトの例外ハンドラーが **CclECI** クラスと **CclEPI** クラスの handleException メソッドにあります。これは、**CclException** オブジェクトの C++ スローを行う単純なルーチンです。オブジェクトのデストラクター内で例外が発生した場合には、何も行いません。デストラクターの中でスローを実行しないでください。予期しない結果を招く原因となります。

遅延同期および同期の同期モードを使用する際のほとんどの場合に、このルーチンが適しています。この例を次に示します。

```
#include <iostream.h>
#include <cicseci.hpp>

void main(void) {
  CclECI *eci;
  eci = CclECI::instance();
  CclFlow flow(Ccl::sync);
  CclBuf buf;
  CclConn conn("CICSOS2","SYSAD","SYSAD");
  buf.setDataLength(80);
  try {
    conn.link(flow,"EC01",&buf);
    cout << (char *)buf.dataArea() << endl;
  }
  catch(CclException &exc) {
    cout << "link failed" << endl;
    cout << "diagnose:" << exc.diagnose() << endl;
    cout << "abend code:" << exc.abendCode() << endl;
  }
};
```

明示的にオブジェクトのデストラクター内で例外を処理したい場合は、 CclECI クラスまたは CclEPI クラスをサブクラス化して、独自の例外ハンドラーを実装する必要があります。

```
void CclECI::handleException(CclException except) {
  if (*(except.methodName()) != '~') {
    throw( except );
  }   else {

// Handle a destructor exception, but ensure that this
// routine just returns

  }
}
```

非同期例外処理

非同期の同期モードを使用する場合は、 CclECI をサブクラス化して ECI handleException ルーチンをオーバーライドする必要があります。非同期モードを指定すると、クラス・ライブラリー DLL に制御される別のスレッドが作成され、そのスレッド内で例外が発生することになります。そのスレッドで例外が生じると、デフォルトの例外ハンドラーがその例外をスローしますが、そのスローをトラップするコードはクラス・ライブラリーにはありません。処理できない例外の場合、ほとんどのコンパイラー実行時のデフォルト・アクションとしてアプリケーションを終了させます。

新規に例外ハンドラーを作成するには、次のようにします。

```
class MyCclECI : public CclECI {
public:
   void handleException( CclException ex) {
         // Place whatever code you want here, for example set a
         // semaphore, or generate a Window Message
         }
   };
```

ECI クラスをサブクラス化しても、このクラスのオブジェクトを 1 つだけユーザーのアプリケーション用に作成することができますが、インスタンス・メソッドは使用せず、次のようにしてオブジェクトを明示的に作成するか、

```
MyCclECI myeci;
```

次のように新規演算子を使用しなければなりません。

```
MyCclECI *pmyeci;
pmyeci = new MyCclECI;
```

外部呼び出しインターフェース

ECI は、2 つのインターフェースのうちの 1 つで、非 CICS クライアントはこれを用いて CICS サーバーと対話することができます。ECI オブジェクト・モデルは、ECI の機能にアクセスする際の一連のクラスから構成され、ECI による CICS クライアント・プログラミングへのオブジェクト指向的アプローチをサポートします。次のクラスが含まれます。

表 1. C++ ECI クラス

オブジェクト	クラス名	説明
グローバル	Ccl	グローバル列挙を含む
バッファー	CclBuf	サーバーとのデータの交換に使用
Connection	CclConn	接続をサーバーにモデル化

表 1. C++ ECI クラス (続き)

オブジェクト	クラス名	説明
ECI	CclECI	CICS サーバーへのアクセスを制御し、リストする
例外	CclException	例外情報をカプセル化する
フロー	CclFlow	単一クライアント / サーバーの対話を処理する
SecAttr	CclSecAttr	セキュリティー属性 (パスワード) について情報を提供する
SecTime	CclSecTime	日付と時間情報を提供する
UOW	CclUOW	サーバーの作業単位に一致する － リカバリー可能リソースへの更新管理に使用される

Commarea の使用

CommArea はプログラムによって割り振られるストレージのブロックです。クライアント・プログラムは commarea を使用してサーバーにデータを送信し、サーバーは同じストレージを使用してクライアントにデータを戻します。そのため、 CommArea の大きさはサーバーに送信される予定のすべての情報を含む十分な大きさであり、かつサーバーから戻される可能性のあるすべての情報を含む十分な大きさで作成しなければなりません。

たとえば、12 バイトのシリアル番号をサーバーに送信する必要がある場合でも、サーバーからは 20 KB 戻される可能性もあります。この場合は、CommArea を 20 KB のサイズで作成しなければなりません。コードは次のようになります。

```
 // serialNo is a Null terminated string
CclBuf Commarea;  // create extensible buffer object
Commarea.assign(strlen(serialNo),serialNo);  // Won't include the Null
Commarea.setDataLength(20480);  // stores Nulls in the unused area
```

この例では、シリアル番号が新規の commarea に保管され、その後サイズが 20480 に増やされます。余分なバイトはヌルで埋められます。サーバーに伝送される情報を最少に抑えていることが重要です。クライアント・ソフトウェアは余分なヌルをストリップし、サーバーに 12 バイトを伝送します。

潜在サーバーの検出

クライアント・プログラムが使用できる CICS サーバーに関する情報は、CICS Transaction Gateway 構成ファイル、ctg.ini で定義されます。詳細につい

ては、「*CICS Transaction Gateway: Gateway 管理*」の『構成』を参照してください。この種の定義が存在するからといって、サーバーが使用できるとは限りません。

ECI オブジェクト **CclECI** では、その **serverCount**、 **serverDesc**、および **serverName** メソッドを用いたこのサーバー情報にアクセスします。

ECI クラスがサブクラス化されていない限り、その固有のインスタンスは、次の例のようなクラス・メソッド **instance** を用いて検出されます。

```
CclECI* pECI = CclECI::instance();
printf( "Server Count = %d¥n", pECI-> serverCount() );
printf( "Server1 Name = %s¥n", pECI-> serverName( 1 ) );
  ...
```

以下は、代表的な作成出力です。

```
Server Count = 2
Server1 Name = DEVTSERV
```

サーバー接続

クライアント・プログラムには、対話の対象となる CICS サーバーごとに、1 つの接続オブジェクト **CclConn** が必要です。接続オブジェクトの作成時に、以下のようなオプションのデータを指定することができます。

- 接続するサーバーの名前。これは、CICS クライアント初期設定ファイルで定義されたサーバー名の 1 つでなければなりません。これを省略すると、ECI によって定義されたデフォルトのサーバーが使用されます。
- ユーザー ID。サーバーによっては、特定の対話を許可するにはクライアントがユーザー ID (およびパスワード) を示すことが必要な場合があります。
- パスワード。

この例で、接続オブジェクトは、サーバー名、ユーザー ID およびパスワードによって作成されます。

```
CclConn serv2( "Server2","sysad","sysad" );
```

接続オブジェクトを作成しても、それ自体ではサーバーとの対話は起こりません。接続オブジェクトの情報は、次のサーバー要求呼び出しのいずれかが出されたときに使用されます。

link − サーバー・プログラムの実行を要求する。

status − サーバーの状況 (可用性) を要求する。

changed − この状況での変更の通知を要求する。

cancel — **changed** 要求の取り消しを要求する。

これらは、接続クラスのメソッドです。サーバー要求呼び出しには、他に 2 つがあります。すなわち、作業単位クラスの **backout** メソッドと **commit** メソッドです。これらのすべてのメソッドの使用方法の詳細については、次の節に記載しました。

サーバー・プログラムへのデータの受け渡し

バッファー・オブジェクト **CclBuf** は、サーバー・プログラムとの間のデータの受け渡しに使用する通信域をカプセル化するときに、クライアント・プログラムで使用します。バッファー・オブジェクトの使用は、通信域に限定されません。これらは、汎用のデータ整理にはかなり柔軟性があります。

次のコードは、バッファー・オブジェクトを構成し、そのデータ域にテキスト・ストリングを割り当て、挿入、かつ付加しながら、それを動的に拡張します。

```
CclBuf comma1;
comma1 = "Some text";
comma1.insert( 9,"inserted ",5 ) += " at the end";
cout << (char*)comma1.dataArea() << endl;
  ...
```

以下は作成出力です。

```
Some inserted text at the end
```

次の例では、既存のメモリー構造が使用されます。これは、たとえば、サーバー・プログラムで使用されるレコードと一致しています。この場合、バッファー・オブジェクトは、レコードが固定長、外部定義であることを認識し、かつ以降の処理で拡張できないことを確認します。リンク呼び出しは、serv2 接続オブジェクトが定義した CICS サーバー上でのプログラム QVALUE の実行を要求し、バッファー・オブジェクト comma2 がオーバーレイされる構造体を介してデータを渡します。

```
struct rec{
         short key;
         char name[8];
         char retval[70];
         };
rec record1 = { 1234,"Hilary" };
CclBuf comma2( sizeof(rec),&record1 );
serv2.link( sflow,"QVALUE",&comma2 );
  ...
```

サーバーから戻される通信域もバッファー・オブジェクトに入っています。

サーバーの対話の制御

フロー・オブジェクト **CclFlow** は、クライアント・プログラムとサーバー間の各対話を制御し、応答処理の同期状態 (同期、遅延同期もしくは非同期) を判別します。この例では、同期フロー・オブジェクトを作成します。

```
CclFlow sflow( Ccl::sync );
```

フロー・オブジェクトは、サーバー要求呼び出しが最初に出されたときに参照され、そのとき以降、サーバー からの対応する応答のすべてのクライアント処理が完了するまでアクティブ状態を保ちます。フロー・オブジェクトは、その時点で非アクティブに設定され、再使用可能もしくは削除可能になります。フロー・オブジェクトは、そのアクティブ状態の間中、制御対象であるクライアント / サーバーの対話の状態を保ちます。

応答の受信の際に呼び出される応答ハンドラーのインプリメンテーションを行うには、フロー・クラスのサブクラス化が必要です。このことは、同期タイプに関係なく該当します。応答ハンドラーには、サーバーが戻す通信域が入ったバッファー・オブジェクトが渡されます。デフォルトの応答ハンドラーが指定されます。これは、なんの処理も行わずに、単に呼び出し側に戻るだけです。

互いに異なるクライアント / サーバーの通信域プロトコルの必要に応じるには、個別のフロー・サブクラスが必要になります。同時に多数のフローがアクティブ状態になる場合があります。同じクライアントによって、同時に多数のサーバーが使用される場合があります。

同期応答処理

同期モデルでは、クライアントは、サーバーから最終的に応答が受け取られるまで、サーバー要求呼び出しでブロックされます。

次のコードは、コマンド行で指定されたパラメーターを用いてサーバー・プログラムを呼び出します。これは、例外の処理またはサーバーからの応答の処理のために、サブクラス化を行いません。

```
  ...
CclECI* pECI = CclECI::instance();
CclConn server1( argv[1],argv[2],argv[3] );
CclBuf  comma1( argv[4] );
CclFlow sflow( Ccl::sync );
server1.link( sflow,"ECIWTO",&comma1 );
```

プログラムは、ECI オブジェクトに対するアクセス権を獲得し、指定されたサーバー名、パスワードおよびユーザー ID を用いて接続オブジェクトを構成し

ます。次に、コマンド行からのテキストを用いてバッファー・オブジェクトが構成され、同期フロー・オブジェクトが作成されます。

リンク呼び出しは、サーバー上の ECIWTO サンプル・プログラムの実行を要求し、バッファー内のそのサンプル・プログラムにテキストを渡します。この場合、サーバーからの応答が受信されるまで処理はブロックされます。
ECIWTO は、通信域をサーバーのオペレーター・コンソールに単に書き込むだけで、それを未変更のままクライアントに戻します。

応答が受信された後、プログラムは最新の例外コードを報告し、戻された通信域を印刷します。

```
cout << "Link returned with ¥""
          << pECI-> exCodeText() << "¥"" << endl;
cout << "Reply from CICS server: "
          << (char*)comma1.dataArea() << endl;
```

以下のように、プログラム ECICPO1 が呼び出された場合、

```
ECICPO1 DEVTSERV sysad sysad "Hello World"
```

正常終了時に予期される出力は、次のようになります。

```
Link returned with "no error"
Reply from CICS server: Hello World
```

対話を制御するフロー・オブジェクトが、応答ハンドラーをインプリメントしたサブクラスのインスタンスである場合は、処理が元のサーバー要求呼び出しの後の文で継続する前に、これが呼び出され、実行されます。たとえば、次の非同期の例で定義されているフロー・サブクラスが、使用されている場合があります。

非同期応答処理

非同期モデルでは、クライアント・プログラムはサーバー要求呼び出しを出してから、応答を待たずに次の文でただちに継続します。サーバーから応答を受け取ると、その応答はただちに (クライアントがほかになんらかの処理を行っていても、それと並行して) その対話を制御するフロー・オブジェクトの応答ハンドラーに渡されます。

次のコードは、コマンド行で指定されたパラメーターを用いてサーバー・プログラムを呼び出します。このコードは、ECI クラスをサブクラス化して例外を処理し、フロー・クラスをサブクラス化してサーバーからの応答を処理します。

以下は、応答ハンドラーのインプリメンテーションが受信した応答を印刷するだけの、フロー・クラスの単純なサブクラスです。

```
class MyCclFlow : public CclFlow {
public:
       MyCclFlow( Ccl::Sync sync ) : CclFlow( sync ) {}
  void handleReply( CclBuf* pcomm ){
    cout << "Reply from CICS server: "
                << (char*)pcomm-> dataArea() << endl;
    }
  };
```

プログラムは、サブクラス化した ECI オブジェクトを構成し、次に、指定されたサーバー名、パスワードおよびユーザー ID を用いて接続オブジェクトを構成します。プログラムは、コマンド行からのテキストを用いてバッファー・オブジェクトを構成し、非同期のサブクラス化フロー・オブジェクトを構成します。

リンク呼び出しは、サーバー上の ECIWTO サンプル・プログラムの実行を要求し、バッファー・オブジェクト内のそのサンプル・プログラムにテキストを渡します。次に、リンク呼び出しの後の文の処理を継続します。

```
  ...
MyCclECI myeci;
CclConn server1( argv[1],argv[2],argv[3] );
CclBuf  comma1( argv[4] );
MyCclFlow asflow( Ccl::async );
server1.link( asflow,"ECIWTO",&comma1 );
  ...
```

サンプルでは、主プログラムに対してほかに行うことがないため、主プログラムにユーザー入力を待たせて、終了が早すぎないようにします。

```
cout << "Server call in progress. Enter q to quit..." << endl;
char input;
cin >> input;
```

その間に、サーバーから応答が戻ったときに、応答ハンドラーが呼び出され、例外がないという前提で、戻された通信域を印刷します。非同期モデルでは、戻された通信域を保持するバッファー・オブジェクトは、内部的にフロー・オブジェクト内に割り振られ、応答ハンドラーの実行後に削除されますので、注意してください。元のリンク呼び出しで指定されたバッファー・オブジェクトは、応答には使用されないため、リンク呼び出しが戻ると同時に削除することができます。

以下のように、プログラム ECICPO2 が呼び出された場合、

```
ecicpo2 DEVTSERV sysad sysad "Hello World"
```

正常終了時に予期される出力は、次のようになります。

```
Server call in progress. Enter q to quit...
Reply from CICS server: Hello World
q
```

クライアント・プログラムは、サーバーから応答が受信されるまでは実際になにも行えないと決めた時点で、適切なフロー・オブジェクトに **wait** メソッドを使用することができます。これで対話は実質的に同期化され、クライアントをブロックします。

```
asflow.wait();
```

遅延同期応答処理

遅延同期モデルでは、クライアント・プログラムはサーバー要求呼び出しを出してから、応答を待たずに次の文でただちに継続します。非同期の場合と異なり、サーバー応答がその到着と同時に処理される場合、クライアントは応答の **poll** を待機する時点を決定します。

ポーリングが出されると、フロー・オブジェクトは、元のサーバー要求から実際に応答があったかどうかを検査します。あった場合、フロー・オブジェクトの応答ハンドラーが同期的に呼び出され、バッファー・オブジェクトの通信域に渡されます。ポーリングは、その呼び出し側に応答の受信の有無を示す値を戻します。応答がない場合は、後で再度応答することができます。

上記で説明したフロー・クラスの場合と同じ単純なサブクラスが使用されます。遅延同期応答処理であることを示す、若干の変更が主プログラムに加えられています。

```
  ...
MyCclECI myeci;
CclConn server1( argv[1],argv[2],argv[3] );
CclBuf  comma1( argv[4] );
MyCclFlow dsflow( Ccl::dsync );
server1.link( dsflow,"ECIWTO",&comma1 );
  ...
```

デモンストレーションの目的で、応答がサーバーから受信されたことをポーリングが示すまで、遅延によってプログラムがループするようになっています。遅延同期モデルの場合は、戻される通信域を保持するバッファー・オブジェクトを、パラメーターとして **poll** メソッドに指定することができます。以下の例で、**poll** メソッドにバッファー・オブジェクトが指定されなかった場合、バ

ッファー・オブジェクトは内部的にフロー・オブジェクト内に割り振られ、応答ハンドラーの実行後削除されます。

```
  ...
Ccl::Bool reply = Ccl::no;
while ( reply == Ccl::no ) {
  cout << "DSync polling..." << endl;
  reply = dsflow.poll();
  if ( reply == Ccl::no ) DosSleep( msecs );
  }
  ...
```

正常終了の場合の代表的な出力は、以下のようになります。

```
DSync polling...
DSync polling...
DSync polling...
Reply from CICS server: Hello World
```

非同期モデルの場合のように、遅延同期フローの同期化に **wait** メソッドが使用されて、クライアントをブロックします。

サーバーの可用性のモニター

接続オブジェクト **CclConn** には 3 つのメソッドがあり、それが表すサーバー接続の可用性を判別する際に使用することができます。

status
: サーバーの状況 (すなわち、可用性) を要求する。

changed
: この状況での変更の通知を要求する。

cancel
: **changed** 要求の取り消しを要求する。

下記の例で、他の作業に使用中のクライアント・プログラムでサーバーの可用性をモニターする方法を示します。

サーバー状況呼び出しに使用するように設計された、フロー・クラスのサブクラスがあります。応答ハンドラーのインプリメンテーションは、サーバー名とその新しく変更された状況を印刷します。戻された通信域は無視します。次に、次のサーバー状況の変更が受信されるように、変更されたサーバー要求を出します。応答ハンドラーは、サーバーの可用性が変更になるたびに呼び出されます。

```
class ChgFlow : public CclFlow {
public:
       ChgFlow( Ccl::Sync stype ) : CclFlow( stype ) {}
  void handleReply( CclBuf* ) {
         CclConn* ccon = connection();
         cout << ccon-> serverName() << " is "
              << ccon-> serverStatusText() << endl;
         ChgFlow* sflow = new ChgFlow( Ccl::async );
         ccon-> changed( *sflow );
         }
  };
```

主プログラムは、CICS クライアント初期設定ファイルにリストされたサーバー全体で繰り返されます。各主プログラムごとに、非同期状況要求呼び出しが出されます。プログラムは、他に行う作業があると継続します。

```
int numservs = myeci.serverCount();
CclConn* pcon;
ChgFlow* pflo;
for ( int i = 1; i <= numservs ; i++ ) {
  pcon = new CclConn( myeci.serverName( i ) );
  pflo = new ChgFlow( Ccl::async );
  pcon-> status( *pflo );
  }
  ...
```

作成される出力は、以下のようになります。

```
PROD1    is unavailable
DEVTSERV is unavailable
PROD1    is available
```

ECI クライアント・プログラムが実行されていないため、最初は両方のサーバーが使用できません。プログラムが開始し、しばらくして、いずれかのサーバーと接触します。

作業論理単位の管理

作業論理単位とは、ファイルまたはキューなどのリカバリー可能リソースに対して一連の更新を行うためにサーバー上で確立しなければならないすべての処理のことです。作業単位が正常に終了すると、変更はすべてコミットか、バックアウトすることができます。異常終了した場合は (たとえば、プログラムの異常終了のため)、変更はすべてバックアウトされます。

クライアントが管理する作業論理単位には多数のサーバー・リンク要求を組み込むことができ、また多数のアクティブ状態の作業単位を同時にクライアントが管理することができますが、ECI から若干の制約が課せられます。所定の作業論理単位がリンクを組み込めるサーバーは、1 つに限られます。作業論理単位で一度にアクティブ状態になれるリンクは 1 つのみであるため、非同期要求では注意が必要です。

クライアント・プログラムは、管理する必要がある作業論理単位ごとに作業単位オブジェクト **CclUOW** を使用します。このコードは、作業単位オブジェクトを作成します。

```
CclUOW uow;
```

作業単位に加わるサーバー・リンク要求は、対応する作業単位オブジェクトを参照します。作業単位に加入するリンクがすべて正常に完了すると、作業単位は、作業単位オブジェクトの **commit** メソッドによってコミットするか、**backout** によってバックアウトすることができます。

```
serv1.link( sflow, "ECITSQ", &( comma1="1st link in UOW" ), &uow );
serv1.link( sflow, "ECITSQ", &( comma1="2nd link in UOW" ), &uow );
  ...
uow.backout( sflow );
```

UOW オブジェクトが使用されない場合、各リンク呼び出しは、完全な作業単位 (CICS サーバーの LINK SYNCONRETURN と同じ) になります。

作業論理単位を使用するときは、アクティブな作業単位のバックアウトまたはコミットを必ず行うようにしてください。特にプログラムの終了時には必ず行ってください。作業論理単位がまだアクティブかどうかは、uowId メソッドでゼロ以外の値を検査すると知ることができます。

ECI のためのセキュリティー管理

パスワード満了管理をサポートするサーバーでセキュリティー管理を行えるようになりました。サポートしているサーバーとプロトコルの詳細については、「*CICS® ファミリー: クライアント / サーバー・プログラミング*」を参照してください。

これらのフィーチャーを使用するためには、最初に接続オブジェクトを構成しなければなりません。選択可能な 2 つのメソッドは、サーバー・セキュリティー・システムを持つ接続オブジェクト内でユーザー ID とパスワードを検査する **verifyPassword** と、サーバーでパスワードを変更することができる **changePassword** です。成功すると、その結果、接続オブジェクトのパスワードが更新されます。

いずれかの呼び出しが成功すると、セキュリティーについての情報を提供する内部オブジェクト CclSecAttr オブジェクトへのポインターが戻されます。このオブジェクトは最新の検査済み日付と時間、満了日およびおよび最新のアクセスされた日付と時間などの情報を提供します。たとえば、最新の検査済み日付

を照会すると、さまざまな形式で情報を入手できるようにするオブジェクトへのポインターが戻されます。次にこれらのさまざまなオブジェクトの使用法を示すサンプル・コードを示します。

```
// Connection object already created called conn
CclSecAttr *pAttrblock;             // pointer to security attributes
CclSecTime *pDTinfo;                // pointer to Date/Time information
try {
     pAttrblock = conn->verifyPassword();
     pDTinfo = pAttrblock->lastVerifiedTime();
     cout << "last verified year  :" <<pDTinfo->year() << endl;
     cout << "last verified month :" <<pDTinfo->month() << endl;
     cout << "last verified day   :" <<pDTinfo->day() << endl;
     cout << "last verified hours :" <<pDTinfo->hours() << endl;
     cout << "last verified mins  :" <<pDTinfo->minutes() << endl;
     cout << "last verified secs  :" <<pDTinfo->seconds() << endl;
     cout << "last verified 100ths:" <<pDTinfo->hundredths() << endl;
// Use a tm structure to produce a single line text of information
     tm mytime;
     mytime = pDTinfo->get_tm();
     cout << "full info:" << asctime(&mytime) << endl;
}
catch (CclException &ex)
{

// Could check for expired password error and handle if required
  cout << "Exception occurred: " <<ex.diagnose()<< endl;
}
```

セキュリティー属性と日付 / 時間メモリーはすべて接続オブジェクトで処理されることに注意してください。接続オブジェクトを破棄すると、そのオブジェクトによって保持されているセキュリティー情報も破棄されます。

C++ 外部表示インターフェース

既存の CICS サーバー・アプリケーションの多くは、3270 端末インターフェース用に作成されており、CICS には、基本マッピング・サポート (BMS) など、これらのデータ・ストリームを処理するいくつかの強力な機能があります。これは、クライアントがこれらのサーバー・プログラムとインターフェースをとる際に役立ちます。

手続き型プログラミングの場合は、外部表示インターフェース (EPI) に、クライアントがサーバー上のトランザクションと通信し、3270 データ・ストリームを処理するためのメカニズムがあります。

EPI のサポート用に用意されたクラスによって、オブジェクト指向技法を用いるプログラマーは EPI に備わった以下の機能に簡単にアクセスすることができます。

- CICS サーバーへの 3270 セッションの接続
- CICS トランザクションの開始
- 3270 データ・ストリームの送受信

クラスには、3270 データ・ストリーム処理用の高水準の構成が備わっており、以下のような手続き型の CICS EPI サポートも拡張されます。

1. フィールドと属性のような 3270 データ・ストリーム、およびトランザクション ID などの CICS トランザクション・ルーティング・データを処理する、汎用 C++ クラス。
2. BMS マップ・ソース・ファイルからの、特定 CICS アプリケーション用 C++ クラスの生成。これらのクラスによって、クライアント・アプリケーションは、CICS サーバーの BMS アプリケーションで使用される場合と同じフィールド名を用いて、3270 パネルのデータにアクセスすることができます。

注: これらのクラスには、 DBCS フィールドを含む 3270 データ・ストリームに対する特定のサポートは含まれていません。ただし、これらのクラスは、 SBCS および DBCS 混合フィールドを含む 3270 データ・ストリームもサポートしていません。

BMS ユーティリティーは、CICS BMS マップ・セットから、 C++ クラス・ソース・コード定義とインプリメンテーションを静的に作成するためのツールです。

以下は、提供されるクラスの概要です。

表 2. C++ EPI クラス

オブジェクト	クラス名	説明
グローバル	Ccl	グローバル列挙を含む。
EPI	CclEPI	EPI を初期化する。このクラスにも、クライアントがアクセスできる CICS サーバーに関する情報を得るメソッドがある。
例外	CclException	エラー情報をカプセル化する。
フィールド	CclField	仮想画面上の単一のフィールドをサポートし、フィールド・テキストと属性にアクセスする。

表 2. C++ EPI クラス (続き)

オブジェクト	クラス名	説明
マップ	CclMap	このクラスは、BMS マップ情報を用いて、**CclField** オブジェクトにアクセスする。CICSBMSC ユーティリティーは、**CclMap** から派生するクラスを生成する。
Screen	CclScreen	端末 (**CclTerminal** オブジェクト) ごとに、それに関連づけられた仮想画面をもつ。**CclScreen** クラスには、一連の **CclField** オブジェクトとそれらのオブジェクトにアクセスするメソッドが含まれている。また、汎用画面処理用のメソッドも含まれている。
SecAttr	CclSecAttr	セキュリティー属性 (パスワード) について情報を提供する。
SecTime	CclSecTime	日付と時間情報を提供する。
セッション	CclSession	同期、非同期および遅延同期モードでのサーバーとの通信を制御する。 アプリケーションは、**CclSession** を用いてその独自のクラスを派生させ、特定の CICS トランザクションをカプセル化することができる。
端末	CclTerminal	3270 端末の CICS への接続を制御する。 **CclTerminal** クラスは、CICS 会話型トランザクション、疑似会話型トランザクション、および ATI トランザクションを処理する。1 つのアプリケーションで、多数の **CclTerminal** オブジェクトを作成することができる。

自動トランザクション始動 (ATI) のサポート

CICS サーバーの API 呼び出し EXEC CICS START を使用すると、サーバー・プログラムは特定の端末でトランザクションを開始することができます。自動トランザクション開始 (ATI) と呼ばれるこの機構では、これらのトランザクションと通常のクライアント始動トランザクションとの間の対話を処理するために、クライアント・サイドで追加のプログラミングが必要です。

まず、クライアント・アプリケーションは **CclTerminal** クラスの setATI() および queryATI() メソッドを使用して、 ATI トランザクションが許可されるか

どうかを制御することができます。 ATI のデフォルト設定は使用不可です。次に示すコードの一部は、特定の端末で ATI を使用可能にする方法です。

```
// Create terminal connection to CICS server
CclTerminal terminal( "myserver" );
// Enable ATIs
terminal.setATI(CclTerminal::enabled);
```

CICS クライアントは、トランザクションの処理中は端末で ATI をキューします。 CclTerminal クラスは以下の 1 つ以上の機能を行うことができます。

- トランザクション終了後すぐに未解決の ATI を実行する
- ATI 応答を処理するために必要な追加のプログラミングを呼び出す
- クライアント始動トランザクションの前または間で ATI を実行する

どれを行うかは呼び出しの同期タイプが同期、非同期、または遅延同期によって異なります。

同期

CclTerminal send() メソッドを呼び出す場合は、未解決の ATI はクライアント始動トランザクションが完了した後で実行されます。 CclTerminal クラスは ATI 応答を待ち、それから CclScreen の内容を synchronous send() 呼び出しの一部として更新します。クライアント始動トランザクションの前または間で ATI を生じさせたい場合は、 CclTerminal receiveATI() メソッドを呼び出して ATI について同期がとられるのを待ちます。

非同期

クライアント・アプリケーションが非同期セッションのために **CclTerminal** send() メソッドを呼び出すと、 **CclTerminal** クラスは応答を処理するために別のスレッドを開始します。ATI が使用不可の場合は、このスレッドは CICS トランザクションが完了したら終了します。ATI が使用可能の場合は、応答スレッドはトランザクション間で実行し続けます。CclTerminal 状態がアクティブ停止になると、すべての未解決の ATI が実行し、続いて受信された ATI が即時に実行します。応答スレッドは最初の CclTerminal::send() 呼び出しまで開始しないので、任意のクライアント始動トランザクションの前に ATI を生じさせたい場合は、 receiveATI() メソッドを呼び出して応答スレッドを開始することができます。

遅延同期

遅延同期セッションのために **CclTerminal** send() メソッドを呼び出した後で、 poll() メソッドを使用して応答を受信します。未解決の ATI は最後の応答が受信されると (すなわち最後の poll() 呼び出しで) 開始されます。 poll() メソッドを呼び

出して、クライアント始動トランザクション間で、ATI の開始および応答の受信をすることもできます。 poll() メソッドはクライアント始動トランザクションの前または間で呼び出すことができるので、 receiveATI() メソッドは遅延同期セッションに不要 (また無効) です。どのタイプの同期についても、CclSession クラスをサブクラス化することによって handleReply() メソッドを提供できます。クライアント始動トランザクションについては、 ATI 3270 データが受信され、CclScreen オブジェクトが更新されると、このメソッドが呼び出されます。 CclTerminal または CclSession で transID() メソッドを呼び出して ATI を識別することができます。

3270 端末の CICS への接続の開始

CICS に対する端末接続を行うには、CclEPI オブジェクトを作成して、CICS クライアント EPI を初期化しておかなければなりません。 CclECI オブジェクト同様、CclEPI オブジェクトも、CICS Transaction Gateway 構成ファイル、ctg.ini で構成された CICS サーバーに関する情報にアクセスします。次の C++ サンプルで、CclEPI オブジェクトの使い方を示します。

```
#include <cicsepi.hpp> // CICS client EPI headers
  ...
CclEPI epi; // Initialize CICS client EPI
// List all CICS servers in クライアント初期設定ファイル
for ( int i=1; i<= EPI.serverCount(); i++ )
    cout << EPI.serverName(i) << "   "
         << EPI.serverDesc(i) << endl;
```

CICS への 3270 端末の接続を確立するときは、CclTerminal オブジェクトが作成されます。使用される CICS サーバー名は、クライアントの クライアント初期設定ファイル に構成されなければなりません。 CICS サーバーでトランザクションを開始するときは、セッションを制御するための CclSession オブジェクトが必要です。これで、CclTerminal オブジェクトの **send** メソッドを用いて、必要なトランザクション (この例では、CICS 提供のサインオン・トランザクション CESN) を開始することができます。

```
      try {
         // Connect to CICS server
         CclTerminal terminal( "CICS1234" );
    // Start CESN transaction on CICS server
         CclSession session( Ccl::sync );
    terminal.send( &session, "CESN" );
    ...
      } catch (CclException &exception) {
    cout << "CclClass exception: " << exception.diagnose() << endl;
}
```

CICS クラスから渡される例外を処理するときの、試行および補足ブロックの使用に注意してください。

CICS 3270 画面のフィールドのアクセス

CICS への端末の接続が確立されると、CclTerminal、CclSession、CclScreen および CclField オブジェクトを使用して、CICS サーバー・アプリケーションが表示する画面を探して、必要に応じて画面データの読み取りや更新を行います。

CclScreen オブジェクトは、CclTerminal オブジェクトによって作成され、CclTerminal オブジェクトの **screen** メソッドによって得られます。これには、3270 画面に関する一般情報 (たとえば、カーソル位置) を得るためのメソッドと、個々のフィールドにアクセスする (行 / 列の画面位置または索引による) ためのメソッドが備わっています。次の例では、フィールドの内容を印刷出力してから、**PF3** に戻って、先ほど開始した CESN トランザクションを終了します。

```
// Get access to the CclScreen object
CclScreen* screen = terminal.screen();
for ( int i=1; i ≤ screen->fieldCount(); i++ ) {
    CclField* field = screen->field(i); // get field by index
    if ( field->textLength > 0 )
        cout << "Field " << i << ": " << field->text();
}
// Return PF3 to CICS
        screen->setAID( CclScreen::PF3 );
terminal.send( &session );
// Disconnect the terminal from CICS
terminal.disconnect();
```

CclField クラスでは、個々の 3270 フィールドのテキストと属性にアクセスします。これらをいろいろな方法で使用して、3270 画面の情報を見つけ、操作することができます。

```
for ( int i=1; i ≤ screen->fieldCount(); i++ ) {
    CclField* field = screen->field(i); // get field by index
    // Find unprotected (i.e. input) fields
    if ( field->inputProt() == CclField::unprotect )
        ...
    // Find fields containing a specific text string
    if ( strstr(field->text(), "CICS Sign-on") )
        ...
    // Find red fields
    if ( field->foregroundColor() == CclField::red )
        ...
}
```

上記サンプルのストリング「Sign-on」は、それぞれの規則に合わせて変更しなければならない場合があります。たとえば、AIX サーバーではストリング「SIGNON」を使用します。

EPI 呼び出し同期タイプ

CICS クライアント EPI C++ クラスは、同期 (「ブロッキング」)、および遅延同期 (「ポーリング」) と非同期 (「コールバック」) プロトコルをサポートします。

上記の例では、**Ccl::sync** の同期タイプで、CclSession オブジェクトが作成されます。この CclSession オブジェクトが、CclTerminal **send** メソッドの最初のパラメーターとして渡されると、CICS に対する同期呼び出しが行われます。これで、C++ プログラムは、CICS から応答が受信されるまでブロックされます。応答が受信されると、受信した 3270 データ・ストリームに従って CclScreen オブジェクトに更新が加えられてから、C++ プログラムに制御が戻ります。

非同期呼び出しを行うときは、CclTerminal **send** メソッドで使用する CclSession オブジェクトが **Ccl::async** の同期タイプで作成されます。CICS への呼び出しは CclTerminal **send** メソッドを用いて行われますが、制御は、CICS からの応答を待たずに即時にクライアント・アプリケーションに戻ります。CclTerminal オブジェクトは、CICS からの応答を待つ個別のスレッドを開始します。応答が受信されると、CclSession オブジェクトの **handleReply** メソッドが呼び出されます。応答を処理するには、**handleReply** メソッドを **CclSession** サブクラスでオーバーライドする必要があります。

```
class MySession : public CclSession {
public:
    MySession(Ccl::Sync protocol) : CclSession( protocol ) {}
    // Override reply handler method
    void handleReply( State state, CclScreen* screen );
};
```

handleReply メソッドのインプリメンテーションは、CclScreen オブジェクトで使用可能な画面データ (CICS から送信された 3270 データ・ストリームによりインラインに更新されている) を処理することができます。

```
void MySession::handleReply( State state, CclScreen* screen ) {
    // Check the state of the session
    switch( state ) {
    case CclSession::client:
    case CclSession::idle:
        // Output data from the screen
        for ( int i=1; i < screen->fieldCount(); i++ ) {
            cout << "Field " << i << ": " << screen->field->text();
        screen->setAID( CclScreen::PF3 );
        ...
    } // end switch
}
```

CICS から受信される伝送ごとに、**handleReply** メソッドが呼び出されます。CICS サーバー・プログラムの設計によっては、**send** 呼び出しの結果 1 つまたは複数の応答を生じる場合があります。**handleReply** メソッドの *state* パラメーターは、サーバーが応答の送信を終了したかどうかを示します。

CclSession::server
: CICS サーバー・プログラムが依然実行中で、まだ送信データがあることを示します。クライアント・アプリケーションは、現行画面の内容をただちに処理することも、単に応答を待つこともできます。

CclSession::client
: CICS サーバー・プログラムが現在応答待ちであることを示します。クライアント・アプリケーションは、画面の内容を処理し、応答を送信する必要があります。

CclSession::idle
: CICS サーバー・プログラムが完了したことを示します。クライアント・プログラムは、画面の内容を処理し、端末を切断するか、以降のトランザクションを開始する必要があります。

クライアント・アプリケーションのほとんどは、CICS サーバー・プログラムがデータの送信を終了するまで待ってから (すなわち、CclSession/CclTerminal 状態は client または idle)、画面を処理します。しかし、処理の長いサーバー・プログラムによっては、状態がまだ server の間に有効にアクセスできる中間結果や進ちょく情報を送信する場合もあります。

handleReply メソッドのインプリメンテーションは、CclScreen オブジェクトからのデータを読み取り、かつ処理し、必要に応じてフィールドを更新し、そして、戻り伝送に備えたカーソル位置と AID キーを CICS に設定することができます。アプリケーションの主プログラムは、サーバー・アプリケーションを動かすには、さらに CclTerminal オブジェクトのメソッド (**send** または **disconnect**) を呼び出す必要があります。

```
try {
    // Connect to CICS server
    CclTerminal terminal( "CICS1234" );
    // Create asynchronous session
    MySession session(Ccl::async);
    // Start CESN transaction on CICS server
    terminal.send( &session, "CESN" );
    // Replies processed asynchronously in overridden
    // handleReply method
    ...
} catch ( CclException &exception ) {
    cout << "CclClass exception: " << exception.diagnose() << endl;
}
```

handleReply メソッドは、個別のスレッドで実行することに注意してください。主プログラムが応答の受信時点を知る必要がある場合は、メッセージもしくはセマフォーを用いると、**handleReply** メソッドと主プログラム間を連絡することができます。

遅延同期呼び出しを行うときは、CclTerminal **send** メソッドで使用する CclSession オブジェクトが **Ccl::dsync** の同期タイプで作成されます。非同期の場合と同様、CICS への呼び出しは CclTerminal **send** メソッドを用いて行われ、制御は、CICS からの応答を待たずに即時にクライアント・アプリケーションに戻ります。3270 画面の CICS からの更新は、端末オブジェクトのポーリング・メソッドを用いて、後で検索しなければなりません。

```
try {
    // Connect to CICS server
    CclTerminal terminal( "CICS1234" );
    // Create deferred synchronous session
    MySession session(Ccl::dsync);
    // Start CESN transaction on CICS server
    terminal.send( &session, "CESN" );
    ...
    if ( terminal.poll())
        // reply processed in handleReply method
    else
        // no reply received yet
} catch ( CclException &exception ) {
    cout << "CclClass exception: " << exception.diagnose() << endl;
}
```

CICS サーバー・トランザクションは、CclTerminal **send** 呼び出しに応答して、複数の応答を送信する場合があります。したがって、すべての応答を収集するには、複数の CclTerminal **poll** 呼び出しが必要になる場合があります。まだ応答があるかどうかを調べるには、CclTerminal **state** メソッドを使用します。その場合に戻される値は、server です。

同期および非同期の場合は、1 つまたは複数の伝送で CICS から戻された 3270 データを処理するコードをカプセル化するときは、**handleReply** メソッドを使用すると便利です。

EPI BMS 変換ユーティリティー

既存の CICS アプリケーションは、大きな部分で 3270 画面出力用に BMS マップを使用します。すなわち、サーバー・アプリケーションは、3270 データ・ストリームを直接処理するのではなく、BMS マップの名前つきフィールドに対応するデータ構造を使用することができることを意味します。EPI BMS 変換ユーティリティーは、BMS マップ・ソースの情報を用いて、個々のマップに固有のクラスを生成することができます。この結果、フィールドをその名前によってアクセスしたり、フィールドの長さや属性をコンパイル時に知ることができます。

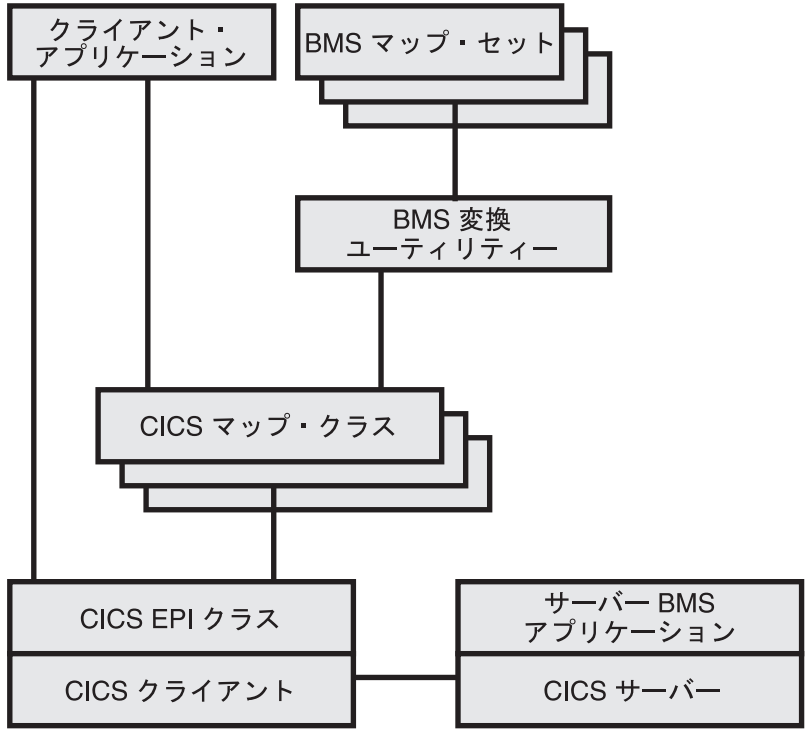

図 1. BMS マップ・クラスの使用

ユーティリティーは、アプリケーションがマップ・オブジェクト内の名前つきフィールドとしてのマップ・データをアクセスするときに使用できる、 C++ クラス定義とインプリメンテーションを生成します。クラスは、マップごとに

定義するため、フィールドの名前と長さをコンパイル時に識別することができます。C++ クラスは、CICS EPI 基底クラスを用いてインバウンドおよびアウトバウンド 3270 データ・ストリームを処理します。生成されたクラスは、すべてのマップ・クラスが必要とする汎用関数を備えた基底クラス **CclMap** を継承します。

以下のように、BMS ソースの CICSBMSC ユーティリティーを実行します。

```
CICSBMSC <filename>.BMS
```

ユーティリティーは、マップ・クラスの定義とインプリメンテーションが入った .HPP および .CPP ファイルを生成します。

注: このユーティリティーは、Linux では提供されていません。

EPI BMS ユーティリティーを用いてマップ・クラスを生成しているため、EPI 基底クラスを使用して、通常の方法で必要な 3270 画面を得ます。次にマップ・クラスを用いて、フィールドに BMS マップのその名前でアクセスします。マップ・クラスは、現行 **CclScreen** オブジェクトのデータに照らして妥当性検査されます。

単一マップを含むマップ・セット

図2 にリストされたマップ・セットには、単純なマップ、MAPINQ1 が入っています。

```
*****************************************************************
*  cicssda  MAPINQ1  -- Wed  2 Aug 14:14:02 1995
*****************************************************************
MAPINQ1 DFHMSD TYPE=&SYSPARM,MODE=INOUT,LANG=C,STORAGE=AUTO,TIOAPFX=YES
MAPINQ1 DFHMDI SIZE=(24,80),MAPATTS=(COLOR,HILIGHT,VALIDN),LINE=1,     X
               COLUMN=1,COLOR=NEUTRAL,HILIGHT=OFF
DTITLE  DFHMDF POS=(2,2),LENGTH=5,ATTRB=(PROT,NORM),COLOR=TURQUOISE,   X
               CASE=MIXED,INITIAL='Date:'
DATE    DFHMDF POS=(2,9),LENGTH=8,ATTRB=(PROT,BRT),CASE=MIXED
...
PRODNAM DFHMDF POS=(5,24),LENGTH=40,ATTRB=(PROT,BRT),CASE=MIXED
...
APPLID  DFHMDF POS=(15,15),LENGTH=8,ATTRB=(PROT,BRT),CASE=MIXED
...
MAPINQ1 DFHMSD TYPE=FINAL
```

図 2. サンプル・マップ・クラス — BMS ソース

BMS 変換ユーティリティーは、このマップ・セットから C++ クラス定義 (31ページの図3 に表示) を生成します。クラス名「MAPINQ1Map」は、BMS ソースのマップ名から派生します。クラスは、**CclMap** クラスを継承します。

クラスは、以下の主要な操作を備えています。

1. コンストラクター **MAPINQ1Map** は、親コンストラクターを呼び出します。親コンストラクターは現行画面と照らし合わせてオブジェクト・マップを妥当性検査します。
2. メソッドの **field** は、BMS ソース・フィールド名 (クラス内の列挙として指定された) を用いてマップのフィールドにアクセスします。

```
//************* CICS Client Classes ************************************
//
// FILE NAME:   epiinq.hpp
//
// DESCRIPTION: C++ header for epiinq.bms
//              Generated by CICS BMS Conversion Utility - Version 1.0
//
//*********************************************************************
#include <cicsepi.hpp>                // CICS Client EPI classes
//---------------------------------------------------------------------
// MAPINQ1Map class declaration
//---------------------------------------------------------------------
class MAPINQ1Map : public CclMap {
public:
  enum           FieldName {
                   DTITLE,
                   DATE,
                   ...
                   PRODNAM,
                   ...
                   APPLID,
                   ...
                 };
//-------------- Constructors/Destructors -------------------------------
                 MAPINQ1Map( CclScreen* screen );
                 ~MAPINQ1Map();
//-------------- Actions ------------------------------------------------
  CclField*      field( FieldName name );         // access field by name
...
}; // end class
```

図 3. サンプル・マップ・クラス — 生成された C++ ヘッダー

EPI BMS マップ・クラスの使用

CICSBMSC を用いて生成されるマップ・クラスは、コンパイルして、クライアント・アプリケーション内に構築することができます。プリコンパイルのヘッダーを用いて Windows NT アプリケーションを作成するときは、 CICSBMSC によって生成された .cpp ファイルに #include stdafx.h を加えることを忘れないでください。

CICS トランザクションを開始するときは、 CclEPI、CclTerminal および CclSession オブジェクトが正規の方法で使用されます。

```
try {
   // Initialize CICS client EPI
   CclEPI epi;
   // Connect to CICS server
   CclTerminal terminal( "CICS1234" );
   // Start transaction on CICS server
   CclSession session( Ccl::sync );
   terminal.send( &session, "EPIC" );
```

この例で、サーバー・プログラムは、その最初のパネルに、マップ・クラス「MAPINQ1Map」が生成されている BMS マップを使用します。マップ・オブジェクトが作成されると、コンストラクターは、マップに定義されたフィールドで画面の内容を妥当性検査します。妥当性検査が正常に行われると、フィールドへのアクセスを、CclScreen オブジェクトからの索引または位置ではなく、その BMS フィールド名を用いて行うことができます。

```
    MAPINQ1Map map( terminal.screen() );
    CclField* field;
    // Output text from "PRODNAM" field
    field = map.field(MAPINQ1Map::PRODNAM);
    cout << "Product Name: " << field->text() << endl;
    // Output text from "APPLID" field
    field = map.field(MAPINQ1Map::APPLID);
    cout << "Product Name: " << field->text() << endl;
} catch (CclException &exception) {
    cout << exception.diagnose()<<endl;
}
```

BMS マップ・オブジェクトは、非同期および遅延同期呼び出しの **handleReply** メソッド内で使用することもできます。

妥当性検査を正常に行うには、現行画面で BMS マップ全体が使用可能でなければなりません。したがって、マップ・クラスは、BMS マップの一部または全部が他のマップか個々の 3270 フィールドによってオーバーレイされている場合は使用することができません。

EPI のためのセキュリティー管理

パスワード満了管理をサポートするサーバーでセキュリティー管理を行えるようになりました。 サポートしているサーバーとプロトコルについては、CICS クライアントの汎用プログラミング・ガイドを参照してください。

これらの機能を使用するには、サインオンできない端末オブジェクトを最初に構築する必要があります。すなわち、ユーザー ID とパスワード (ヌルの場合もある) が必要です。選択可能な 2 つのメソッドは、セキュリティー・サーバ

ー・システムを持つ端末オブジェクト内でユーザー ID とパスワードを検査する **verifyPassword** と、そのサーバーでパスワードを変更するために使用できる **changePassword** です。成功すると、続いて接続オブジェクトのパスワードが更新されます。

いずれかの呼び出しが成功すると、セキュリティーについての情報を提供する内部オブジェクト CclSecAttr オブジェクトへのポインターが戻されます。このオブジェクトは最新の検査済み日付と時間、満了日およびおよび最新のアクセスされた日付と時間などの情報を提供します。たとえば、最新の検査済み日付を照会すると、さまざまな形式で情報を入手できるようにするオブジェクトへのポインターが戻されます。次にこれらのさまざまなオブジェクトの使用法を示すサンプル・コードを示します。

```
// Terminal object already created called term
CclSecAttr *pAttrblock;             // pointer to security attributes
CclSecTime *pDTinfo;                // pointer to Date/Time information
try {
     pAttrblock = term->verifyPassword();
     pDTinfo = pAttrblock->lastVerifiedTime();
     cout << "last verified year  :" <<pDTinfo->year() << endl;
     cout << "last verified month :" <<pDTinfo->month() << endl;
     cout << "last verified day   :" <<pDTinfo->day() << endl;
     cout << "last verified hours :" <<pDTinfo->hours() << endl;
     cout << "last verified mins  :" <<pDTinfo->minutes() << endl;
     cout << "last verified secs  :" <<pDTinfo->seconds() << endl;
     cout << "last verified 100ths:" <<pDTinfo->hundredths() << endl;

// Use a tm structure to produce a single line text of information

     tm mytime;
     mytime = pDTinfo->get_tm();
     cout << "full info:" << asctime(&mytime) << endl;
}
catch (CclException &ex)
{

// Could check for expired password error and handle if required
  cout << "Exception occurred: " <<ex.diagnose()<< endl;
}
```

セキュリティー属性と日付 / 時刻メモリーはすべて端末オブジェクトで処理されることに注意してください。端末オブジェクトを破棄すると、そのオブジェクトによって保持されているセキュリティー情報も破棄されます。

第2部 CICS クライアント用 C++ クラスの説明

以降の章では、すべてのクライアント・クラスに関する説明をアルファベット順に記載しています。インターフェース内で、メソッドのアルファベット順リストが示されています。これらのインターフェースの使用方法に関する詳細については、第 1 部を参照してください。

第4章 CclClass クラス

このクラスは、他のクラス (ECI と EPI の両方) が使用する一部の列挙を定義します。

列挙

Bool

以下の 2 つの値の等しいペアがあります。

no と yes
off と on

Sync

以下の値を使用することができます。

async	非同期
dsync	遅延同期
sync	同期

ExCode

使用可能な値については、105ページの表3 を参照してください。

第5章 CclBuf クラス

CclBuf オブジェクトには、情報の保持に使用できるメモリー内のデータ域が入っています。CclBuf オブジェクト特有の使用法は、CICS サーバーとの間でデータを受け渡しするときに使用する COMMAREA の保持です。

CclBuf オブジェクトの主な用途は、バイト (2 進) データでの使用です。一般に、COMMAREA には、CICS サーバー PL/1 または C プログラムから生じることが多い、アプリケーション固有のデータ構造が入っています。したがって、**assign()** および **insert()** などのメソッドには、アプリケーションのデータ入力用の **void*** パラメーター型が備わっています。SBCS ヌル終了ストリングには限定されたサポートがありますが (一部のコードには、これを利用するものもある)、**CclBuf** クラスには、コード・ページ変換も DBCS サポートもありません。

バッファーの最大データ長は、32 ビット・プラットフォームの無符号長 (2^{32}) の最大値です。CICS では、COMMAREA の 32500 バイトの制限が課せられます。これは、CICS クライアント初期設定ファイルに *MaxBufferSize* パラメーターを設定して、減らすことができます。詳細については、「*CICS クライアント管理*」を参照してください。COMMAREA として使用するバッファー・オブジェクトが長すぎる場合は、データ長例外になります。

CclBuf オブジェクトは、作成時に、そのバッファーとして渡されてくるメモリーの領域を使用するか、その独自のバッファーを割り振ります。このバッファーのデータの長さは、CclBuf オブジェクトの作成後減らすことができます。このバッファーのデータの長さを最初の長さを超えて増やすことができるのは、CclBuf オブジェクトが extensible の **DataAreaType** で作成されている場合に限られます。 extensible の代替は、fixed です。

バッファー・オブジェクトの **DataAreaType** が fixed で、かつメソッドが呼び出された結果、そのデータ域の長さが超過になるような場合は、バッファー・オーバーフロー例外が生じます。例外が処理されないと、バッファーには呼び出しの結果が入り、データ域の長さに切り捨てられます。

メソッドが呼び出された結果、バッファー・オブジェクトのデータ長がそのデータ域の長さより小さい場合、データはヌルで埋め込まれます。

メソッドの多くは、オブジェクト参照を戻します。この結果、ユーザーは呼び出しをメンバー関数に連鎖することができます。たとえば、次のようなコードでは、

```
CclBuf comma1;
comma1="Some text";
comma1.insert( 9,"inserted ",5) += " at the end";
```

次のストリングを作成します。

```
Some inserted text at the end
```

CclBuf コンストラクター

CclBuf (1)

CclBuf(unsigned long *length* **= 0, DataAreaType** *type* **= extensible)**

length
データ域の最初の長さ (バイト)。デフォルトの長さは、0。

type
データ域が拡張可能かどうかを示す列挙。使用可能な値は、extensible か fixed です。デフォルトは、extensible です。

CclBuf オブジェクトを作成し、所定の長さでその独自のデータ域を割り振ります。その中のバイトは、すべてヌルに設定されます。データ長はゼロに設定され、データがバッファーに入れられるまでゼロのままとなります。

CclBuf (2)

CclBuf(unsigned long *length***, void*** *dataArea***)**

length
提供されたデータ域の長さ (バイト)。

dataArea
提供されたデータ域の最初のバイトのアドレス。

拡張不能な CclBuf オブジェクトを作成し、与えられたデータ域をその独自のデータ域として受け入れます。DataAreaOwner は外部に設定されます。

CclBuf (3)

CclBuf(const char* *text***, DataAreaType** *type* **= extensible)**

text
新しい CclBuf オブジェクトにコピーされるストリング。

type
データ域が拡張可能かどうかを示す列挙。使用可能な値は、extensible か fixed です。デフォルトは、extensible です。

CclBuf オブジェクトを作成し、*text* ストリングと同じ長さでその独自のデータ域を割り振り、ストリングをそのデータ域にコピーします。

CclBuf (4)

CclBuf(const CclBuf& *buffer*)

buffer

コピーされる CclBuf オブジェクトの参照。

このコピー・コンストラクターは、所定のオブジェクトのコピーである新しい CclBuf オブジェクトを作成します。新しいバッファーのデータ長、データ域の長さ、およびデータ域の型は、旧バッファーと同じです。新しいバッファーのデータ域所有者は、internal です。

パブリック・メソッド

assign

CclBuf& assign(unsigned long *length*, const void* *dataArea*)

length

ソース・データ域の長さ (バイト)。

dataArea

ソース・データ域のアドレス。

データ域の現在の内容をソース・データで上書きし、データ長をリセットします。

cut

CclBuf& cut(unsigned long *length*, unsigned long *offset* = 0)

length

データ域から切り取るバイト数。

offset

データ域へのオフセット。デフォルトはゼロです。

指定データをデータ域から切り取ります。データ域のデータはヌルで埋め込まれます。

dataArea

const void* dataArea(unsigned long *offset* = 0) const

offset

データ域へのオフセット。デフォルトはゼロです。

所定のオフセットのアドレスをデータ域に戻します。

dataAreaLength

unsigned long dataAreaLength() const
データ域の長さ (バイト) を戻します。

dataAreaOwner

DataAreaOwner dataAreaOwner() const
データ域が **CclBuf** コンストラクターによって割り振られたか、ほかから提供されたかを示す、列挙値を戻します。使用可能な値は、internal と external です。

dataAreaType

DataAreaType dataAreaType() const
データ域が拡張可能かどうかを示す、列挙値を戻します。使用可能な値は、extensible と fixed です。

dataLength

unsigned long dataLength() const
データ域のデータの長さを戻します。この長さは、**dataAreaLength** によって戻される値を超えてはなりません。

insert

CclBuf& insert(unsigned long *length*,
const void* *dataArea*,
unsigned long *offset* **= 0)**

length
CclBuf オブジェクトに挿入される、データの長さ (バイト)。

dataArea
CclBuf オブジェクトに挿入される、ソース・データの開始。

offset
データが挿入されるデータ域へのオフセット。デフォルトはゼロです。

ソース・データをデータ域に所定のオフセットで挿入します。

listState

const char* listState() const
オブジェクトの現在の状態が入った定様式ストリングを戻します。例を以下に示します。

```
Buffer state..&CclBuf=000489B4  &CclBufI=00203A00
dataLength=8  &dataArea=002039C0
dataAreaLength=8  dataAreaOwner=0  dataAreaType=1
```

operator= (1)

CclBuf& operator=(const CclBuf& *buffer***)**

buffer

CclBuf オブジェクトの参照。

データを別のバッファー・オブジェクトから割り当てます。

operator= (2)

CclBuf& operator=(const char* *text***)**

text

CclBuf オブジェクトに割り当てられるストリング。

データをストリングから割り当てます。

operator+= (1)

CclBuf& operator+=(const CclBuf& *buffer***)**

buffer

CclBuf オブジェクトの参照。

データ域のデータに、別のバッファー・オブジェクトからのデータを付加します。

operator+= (2)

CclBuf& operator+=(const char* *text***)**

text

CclBuf オブジェクトに付加されるストリング。

データ域のデータに、ストリングを付加します。

operator==

Ccl::Bool operator==(const CclBuf& *buffer***) const**

buffer

CclBuf オブジェクトの参照。

2 つの CclBuf オブジェクトのバッファーに含まれるデータが同じかどうかを示す、列挙を戻します。使用可能な値は、yes か no です。yes は、データ長が同じで、内容が同じであること意味します。

operator!=

Ccl::Bool operator!=(const CclBuf& *buffer***) const**

buffer

CclBuf オブジェクトの参照。

2 つの CclBuf オブジェクトのバッファーに含まれるデータが異なるかどうかを示す、列挙を戻します。使用可能な値は、yes か no です。no は、データ長が同じで、内容が同じであること意味します。

replace

CclBuf& replace(unsigned long *length*,
const void* *dataArea*,
unsigned long *offset* **= 0)**

length
ソース・データ域の長さ (バイト)。

dataArea
ソース・データ域の開始のアドレス。

offset
CclBuf データ域の開始に相対的な、新規データが書き込まれる位置 デフォルトはゼロです。

所定のオフセットのデータ域の現在の内容を、ソース・データで上書きします。データ長は変りません。

setDataLength

unsigned long setDataLength(unsigned long *length***)**

length
データ域の新しい長さ (バイト)。

データ域の現在の長さを変更して、新しい長さを戻します。 CclBuf オブジェクトが extensible でない場合、データ域の長さは、データ域の元の長さか *length* のどちらか小さい方に設定されます。

length がデータ域の長さより大きい場合、データはヌルで埋め込まれます。

列挙

DataAreaOwner

CclBuf オブジェクトのデータ域が、オブジェクト外に割り振られているかどうかを示します。以下の値を使用することができます。

internal
データ域は、**CclBuf** コンストラクターによって割り振られました

external
データ域は、外部的に割り振られました。

DataAreaType

CclBuf オブジェクトのデータ域をその元の長さより長くできるかどうかを示します。以下の値を使用することができます。

extensible
CclBuf オブジェクトのデータ域をその元の長さより長くできます。

fixed CclBuf オブジェクトのデータ域をその元の長さより長くできません。

第6章 CclConn クラス

クライアントと名前つきサーバー間の ECI 接続を表すときは、クラス **CclConn** のオブジェクトが使用されます。11ページの『サーバー接続』を参照してください。サーバーへのアクセスは、オプションでユーザー ID とパスワードによって制御されます。このオブジェクトは、サーバー内のプログラムを呼び出すことも、接続の状態に関する情報を得ることもできます。詳細については、12ページの『サーバー・プログラムへのデータの受け渡し』および17ページの『サーバーの可用性のモニター』を参照してください。

CclConn オブジェクトを作成しても、CICS サーバーとの対話が生じるわけでも、サーバーが要求の処理に必ず使用可能というわけでもありません。

クライアントとサーバー間の対話には、CclFlow オブジェクトの使用が必要です。詳細については、13ページの『サーバーの対話の制御』を参照してください。

CclConn オブジェクトは、コピーすることも割り当てることもできません。CclFlow もしくは CclUOW オブジェクトがアクティブ中の CclConn オブジェクトを削除しようとすると、activeFlow もしくは activeUOW 例外が生じます。

CclConn コンストラクター

```
CclConn(const char* serverName = 0,
        const char* userId = 0,
        const char* password = 0,
        const char* runTran = 0,
        const char* attachTran = 0)
```

serverName

サーバーの名前。名前が付けられていない場合は、デフォルトのサーバーが使用されます。この名前は、**serverName** メソッドを用いてサーバーへの最初の呼び出しを行った後に見つけることができます。必要なら、ブランクの埋め込みか切り捨てによって、長さを 8 文字に調整することができます。

userId
ユーザー ID (必要な場合)。必要なら、ブランクの埋め込みか切り捨てによって、長さを 16 文字に調整することができます。

password
userID のユーザー ID に対応するパスワード (必要な場合)。必要なら、ブランクの埋め込みか切り捨てによって、長さを 16 文字に調整することができます。

runTran
呼び出されたプログラムが実行する CICS トランザクション。デフォルトでは、デフォルトのサーバー・トランザクションを使用します。必要なら、ブランクの埋め込みか切り捨てによって、長さを 4 文字に調整することができます。

attachTran
呼び出されたプログラムが接続される CICS トランザクション。デフォルトは、デフォルト CPMI の使用です。必要なら、ブランクの埋め込みか切り捨てによって、長さを 4 文字に調整することができます。

このコンストラクターは CclConn オブジェクトを作成します。作成しても、CICS サーバーとの対話が生じるわけでも、サーバーが要求の処理に必ず使用可能というわけでもありません。接続が状況呼び出しにのみ使用される場合、あるいはサーバーにセキュリティーがない場合は、ユーザー ID とパスワードは不要です。

パブリック・メソッド

alterSecurity

void alterSecurity(const char* *newUserid*, **const char*** *newPassword*)

newUserid
新規のユーザー ID。

newPassword
新規のユーザー ID に対応する新規パスワード。

次のリンク呼び出しで使用するユーザー ID とパスワードを更新します。

cancel

void cancel(CclFlow& *flow*)

flow
サーバーの要求呼び出しの制御に使用する CclFlow オブジェクトへの参照。

この接続に関連するサーバーにそれまでに出された **changed** 呼び出しを、すべて取り消します。

changed

void changed(CclFlow& *flow***)**

flow

サーバーの要求呼び出しの制御に使用する CclFlow オブジェクトへの参照。

現行接続状況が変更になったときに、クライアントに通知するようサーバーに要求します。この接続に未決の **changed** 呼び出しがまだある場合、この呼び出しは無視されます。サーバーの可用性を得るには、**serverStatus** または **serverStatusText** を使用します。

changePassword

CclSecAttr* changePassword(const char* *newPassword***)**

newPassword

指定する新規パスワード。

端末オブジェクトに保持されているパスワードと、端末オブジェクトに保持されているユーザー ID 用に外部セキュリティー・マネージャーが記録したパスワードを、クライアント・アプリケーションが変更できるようにします。外部セキュリティー・マネージャーは端末オブジェクトによって定義されたサーバーに置かれているとみなされます。

link

void link(CclFlow& *flow***,**
const char* *programName***,**
CclBuf* *commarea* **= 0,**
CclUOW* *unit* **= 0)**

flow

サーバーの要求呼び出しの制御に使用する CclFlow オブジェクトへの参照。

programName

呼び出されるサーバー・プログラムの名前。必要なら、ブランクの埋め込みか切り捨てによって、長さを 8 文字に調整することができます。

commarea

COMMAREA の呼び出されたプログラムに渡されるデータを保持する、CclBuf オブジェクトを指すポインター。デフォルトでは、COMMAREA を渡しません。

unit
この呼び出しが加わる作業単位 (UOW) を識別する CclUOW オブジェクトを指すポインター。デフォルトは、なしです。18ページの『作業論理単位の管理』を参照してください。

サーバー上の指定プログラムの実行を要求します。サーバー・プログラムは、着信呼び出しを EXEC CICS LINK 呼び出しとみなします。

commarea バッファー・オブジェクトが長すぎると、dataLength 例外が起き、要求は拒否されます。CICS では、32500 バイトの制限を設けています。これは、CICS クライアント初期設定ファイルの MaxBufferSize パラメーターを用いて小さくすることができます。

listState

const char* listState() const

オブジェクトの現在の状態が入った定様式ストリングを戻します。例を以下に示します。

```
Connection state..&CclConn=000489AC  &CclConnI=00203A50
flowCount=0 &CclFlow(changed)=00000000  token(changed)=0
serverName="server  "  userId="userId          "  password="password        "
&CclUOWI=00000000 runTran="run "  attachTran="att "
```

makeSecurityDefault

void makeSecurityDefault()

接続オブジェクトの構造体で指定したように、サーバーに渡される ECI および EPI 要求のデフォルトに、このオブジェクトの現行ユーザー ID とパスワードがなることを、クライアントに通知します。

password (1)

const char* password() const

CclConn オブジェクトが保持するパスワードを 10 文字までスペースで埋めて戻すか、またはパスワードがなければブランクを戻します。

password (2)

void password(Ccl::Bool *unpadded***)**

unpadded

Ccl::Yes
ストリングにスペースを埋め込まずに、保管されたパスワードのヌル終了ストリングを戻します。

Ccl::No
スペースで埋められたストリングを戻します。パラメーターなしでパスワード・メソッドを呼び出すのと同様です。

serverName (1)

const char* serverName() const

CclConn オブジェクトが保持するサーバー・システムの名前をスペースで埋めて戻すか、もしくはデフォルトのサーバーが使用される予定で、まだ呼び出しが行われていない場合は、ブランクが戻されます。

serverName (2)

void serverName(Ccl::Bool *unpadded***)**

unpadded

Ccl::Yes
ストリングにスペースを埋め込まずに、保管されたサーバー名のヌル終了ストリングを戻します。

Ccl::No
スペースで埋められたストリングを戻します。パラメーターなしで serverName メソッドを呼び出すのと同様です。

status

void status(CclFlow& *flow***)**

flow
サーバーの要求呼び出しの制御に使用する CclFlow オブジェクトへの参照。

サーバー接続の状況を要求します。応答が受信されたら、**serverStatus** または **serverStatusText** を用いてサーバーの可用性を得ます。

serverStatus

ServerStatus serverStatus() const

前の **status** または **changed** 要求によって設定された、サーバーの可用性を示す列挙値を戻します。使用可能な値は、列挙の下にリストされています。

serverStatusText

const char* serverStatusText() const

前の **status** または **changed** 要求によって設定された、サーバーの可用性を示すストリングを戻します。

userId (1)

const char* userId() const

CclConn オブジェクトが保持するユーザー ID をスペースで埋めて戻すか、またはそれがなければブランクを戻します。

userId (2)

void userId(Ccl::Bool *unpadded***)**

unpadded

Ccl::Yes
ストリングにスペースを埋め込まずに、保管されたユーザー ID のヌル終了ストリングを戻します。

Ccl::No
スペースで埋められたストリングを戻します。パラメーターなしでユーザー ID メソッドを呼び出すのと同様です。

verifyPassword

CclSecAttr* verifyPassword()

CclConn オブジェクトに保持されているパスワードと CclConn オブジェクトに保持されているユーザー ID によって保持される外部セキュリティー・マネージャーが記録したパスワードが一致しているかどうかを、クライアント・アプリケーションが検査できるようにします。外部セキュリティー・マネージャーは CclConn オブジェクトによって定義されたサーバーに置かれているとみなされます。

列挙

ServerStatus

サーバーの可用性を示します。以下の値を使用することができます。

unknown
サーバーの状況は不明です。

available
サーバーは使用可能です。

unavailable
サーバーは使用不能です。

第7章 CclECI クラス

CclECI クラスのインスタンスは 1 つのみ存在できます。これは、**instance** クラス・メソッドによって作成されます。これは、使用可能サーバーへのクライアントのインターフェースを制御します。

独自の handleException メソッドをインプリメントするには、**CclECI** をサブクラス化する必要があります。

存在できるのは、**CclECI** サブクラスの 1 つのインスタンスのみです。複数のインスタンスを作成しようとすると、multipleInstance 例外が生じます。

CclECI オブジェクトは、コピーすることも割り当てることもできません。

CclECI コンストラクター (protected)

CclECI()

このコンストラクターは保護されており、アクセスできるのはサブクラスのみです。

パブリック・メソッド

exCode

使用すべきでないメソッド
新規アプリケーションではこのメソッドを使用しないでください。このメソッドは使用すべきでなく、旧バージョンとの互換性の目的でのみ提供されています。

Ccl::ExCode exCode() const
最新の例外コードを示す列挙を戻します。使用可能な値は、 105ページの表3にリストされています。

exCodeText

> **使用すべきでないメソッド**
> 新規アプリケーションではこのメソッドを使用しないでください。このメソッドは使用すべきでなく、旧バージョンとの互換性の目的でのみ提供されています。

const char* exCodeText() const
最新の例外コードを記述するテキスト・ストリングを戻します。

handleException

virtual void handleException(CclException &*except*)

except
生じたばかりの例外に関する情報が入った CclException オブジェクト。

このメソッドは、例外が生じる度に呼び出されます。例外を処理するには、常に **CclECI** をサブクラス化し、**handleException** の独自のインプリメンテーションを行う必要があります。8ページの『例外処理』を参照してください。デフォルトのインプリメンテーションは、例外オブジェクトを渡すだけです。

instance

static CclECI* instance()
クライアント上に存在する単一の CclECI オブジェクトへのポインターを戻すクラス・メソッド。以下の例で、その使い方を示します。

```
CclECI* pmgr = CclECI::instance();
```

listState

const char* listState() const
オブジェクトの現在の状態が入った定様式ストリングを戻します。例を以下に示します。

```
ECI state..&CclECI=00203AE0  &CclECII=00203B20
retCode=0  exCode=0
serverCount=0  &serverBuffer=00000000
```

serverCount

unsigned short serverCount() const
クライアント初期設定ファイルの構成に従ってクライアントが接続できる、使用可能サーバーの数を戻します。実際上は、これらのサーバーのうち一部または全部が使用不能な場合があります。10ページの『潜在サーバーの検出』を参照してください。

serverDesc

const char* serverDesc(unsigned short *index* **= 1) const**

index

リスト内の接続サーバーの索引。デフォルトの索引は 1 です。

*index*th サーバーの記述を戻します。10ページの『潜在サーバーの検出』を参照してください。

serverName

const char* serverName(unsigned short *index* **= 1) const**

index

リスト内の接続サーバーの索引。デフォルトの索引は 1 です。

*index*th サーバーの名前を戻します。10ページの『潜在サーバーの検出』を参照してください。

第8章 CclEPI クラス

CclEPI クラスは、CICS クライアント EPI 関数を初期化し、終了します。また、クライアント初期設定ファイルで構成された CICS サーバーに関する情報の入手を可能にするメソッドを備えています。CclTerminal オブジェクトを作成して CICS サーバーに接続するには、アプリケーション処理ごとに、このクラスのオブジェクトを 1 つ作成しておかなければなりません。

CclEPI コンストラクター

CclEPI()

このメソッドは、クライアント上の CICS EPI インターフェースを初期化します。初期化が失敗すると、initEPI 例外が生じます。CICS クライアント EPI の初期化は同期を取ります。つまり、初期化は、**CclEPI** コンストラクターの呼び出しが戻ったときに完了します。

パブリック・メソッド

diagnose

const char* diagnose() const
最新のサーバー呼び出しが戻す状態の記述を保持する、文字ストリングを戻します。

exCode

使用すべきでないメソッド
新規アプリケーションではこのメソッドを使用しないでください。このメソッドは使用すべきでなく、旧バージョンとの互換性の目的でのみ提供されています。

Ccl::ExCode exCode() const
最新の例外コードを示す列挙を戻します。使用可能な値は、105ページの表3 にリストされています。

exCodeText

> **使用すべきでないメソッド**
> 新規アプリケーションではこのメソッドを使用しないでください。このメソッドは使用すべきでなく、旧バージョンとの互換性の目的でのみ提供されています。

const char* exCodeText() const
最新の例外コードを記述するテキスト・ストリングを戻します。

handleException

virtual void handleException(CclException &*except*)

except
生じたばかりの例外に関する情報が入った CclException オブジェクト。

このメソッドは、例外が生じる度に呼び出されます。例外を処理するには、try ... catch を使用することも、 **CclEPI** をサブクラス化して、独自の **handleException** のインプリメンテーションを行うこともできます。デフォルトのインプリメンテーションは、例外オブジェクトを渡すだけです。

serverCount

unsigned short serverCount()
クライアント初期設定ファイル の構成に従ってクライアントが接続できる、使用可能サーバーの数を戻します。

serverDesc

const char* serverDesc(unsigned short *index* **= 1)**

index
構成されたサーバーの索引

選択した CICS サーバーの記述、もしくは、指定サーバーの クライアント初期設定ファイル に情報がない場合は、ヌルを戻します。索引が、構成したサーバーの数を超えると、maxServers 例外が生じます。

serverName

const char* serverName(unsigned short *index* **= 1)**

index
構成されたサーバーの索引

要求した CICS サーバーの名前、もしくは、指定サーバーの クライアント初期設定ファイル に情報がない場合は、ヌルを戻します。索引が、構成したサーバーの数を超えると、maxServers 例外が生じます。

state

State state() const
EPI の状態を示す列挙を戻します。以下の値を使用することができます。
active
EPI は、正常に初期化されました
discon
EPI は終了しました
error EPI の初期化は失敗しました

terminate

使用すべきでないメソッド
新規アプリケーションではこのメソッドを使用しないでください。このメソッドは使用すべきでなく、旧バージョンとの互換性の目的でのみ提供されています。

void terminate()
CICS クライアント EPI を、制御された方法で終了します。CclEPI オブジェクトは、終了の際に発生したものがアプリケーションによってモニターできる状態のままに置かれます。

terminate メソッドは CclEPI オブジェクトの消滅時に呼び出されるので、ユーザーがこのメソッドを独自に呼び出す必要はなくなります。

列挙

State

EPI の状態を示す列挙。以下の値を使用することができます。
active
EPI は、正常に初期化されました
discon
EPI は終了しました
error EPI の初期化は失敗しました

第9章 CclException クラス

CICS クライアント・オブジェクトは、問題を検出すると、**CclException** クラスのオブジェクトを構成します。

このような問題を処理するには、**CclECI** または **CclEPI** クラスをサブクラス化して、独自の **handleException** メソッドのインプリメンテーションを行う必要があります。8ページの『例外処理』を参照してください。このメソッドは、CclException オブジェクトのメソッドにアクセスでき、必要なアクションはすべてとるようにコーディングすることができます。たとえば、プログラムを停止したり、ダイアログ・ボックスをポップアップすることがきます。

あるいは、C++ `try ... catch` ブロックを用いて例外を処理することができます。

CclException オブジェクトは割り当てることができません。そのコンストラクターは、CICS クライアント・クラスのインプリメンテーションのみが使用するように考えられています。

パブリック・メソッド

abendCode

const char* abendCode()

ECI 異常終了コードを含むヌル終了ストリングを戻します。(選択可能な異常終了コードがない場合はブランクを戻します。)

className

const char* className() const

例外が生じたクラスの名前を戻します。

diagnose

const char* diagnose() const

診断出力で使用する例外を説明したテキストを戻します。以下にその例を示します。

```
unknown server, classname=CclFlowI, methodName=afterReply, originCode=13
"link", flowId=2, retCode=-22, abendCode="    "
```

exCode

Ccl::ExCode exCode() const
例外コードを戻します。105ページの表3 を参照してください。

exCodeText

const char* exCodeText() const
例外コードを記述したテキスト・ストリングを戻します。

exObject

void* exObject() const
このメソッドは、ECI と EPI の両方に関係するものです。

exObject は、例外のときにサーバーの対話を制御するオブジェクトを指すポインターを戻します。このようなオブジェクトがない場合は、ヌル・ポインターが戻されます。

- ECI の場合、ポインターは **CclFlow*** にキャストされる必要があります。例を以下に示します。

```
CclFlow* pflo = (CclFlow*) ex.exObject();
```

- EPI の場合、exObject は、例外ブロック内の関係のある CclTerminal オブジェクト・ポインターを戻します。これは、**CclTerminal*** にキャストされる必要があります。例を以下に示します。

```
CclTerminal* pTerm = (CclFlow*)ex.exObject();
```

methodName

const char* methodName() const
例外が生じたメソッドの名前を戻します。

第10章 CclField クラス

CclField クラスのオブジェクトは、責任をもって 3270 画面の 1 つのフィールドを監視しなければなりません。**CclField** オブジェクトは、作成され、CICS サーバーからの 3270 データが CclScreen オブジェクトによって処理されると削除されます。

このクラスのメソッドによって、フィールドのテキストと属性を読み取り、かつ更新することができます。修正済みフィールドは、次の **send** で CICS サーバーに送信されます。

パブリック・メソッド

appendText (1)

void appendText(const char* *text*, **unsigned short** *length*)

text

フィールドに付加されるテキスト

length

フィールドに付加される文字数

text からの *length* 文字を、すでにフィールド内にあるテキストの終わりに付加します。

appendText (2)

void appendText(const char* *text*)

text

フィールドに付加されるヌル終了ストリング

text ストリング内の文字を、すでにフィールド内にあるテキストの終わりに付加します。

backgroundColor

Color backgroundColor() const

フィールドのバックグラウンド・カラーを示す列挙を戻します。使用可能な値を、このクラスの説明の終わりの **Color** の下に示します。

baseAttribute

char baseAttribute() const

フィールドの 3270 基本属性を戻します。

column

unsigned short column() const

画面のフィールドの開始位置の列番号 (左端列を 1 とした) を戻します。

dataTag

BaseMDT dataTag() const

フィールドのデータが修正されているかどうかを示す列挙を戻します。以下の値を使用することができます。

modified
unmodified

foregroundColor

Color foregroundColor() const

フィールドのフォアグラウンド・カラーを示す列挙を戻します。使用可能な値を、このクラスの説明の終わりの **Color** の下に示します。

highlight

Highlight highlight() const

使用される強調表示の型を示す列挙を戻します。使用可能な値は、このクラスの説明の終わりの **Highlight** の下に示します。

inputProt

BaseProt inputProt() const

フィールドが保護されるかどうかを示す列挙を戻します。以下の値を使用することができます。

protect
unprotect

inputType

BaseType inputType() const

このフィールドの入力データ型を示す列挙を戻します。以下の値を使用することができます。

alphanumeric
numeric

intensity

BaseInts intensity() const

フィールドの輝度を示す列挙を戻します。以下の値を使用することができます。

dark
normal
intense

length

unsigned short length() const

フィールドの全長を戻します。ここには、3270 属性バイト情報を保管するために使用される 1 バイトが含まれています。そのため、実際のデータ用のスペースはこのメソッドで戻される値より 1 小さくなります。 textLength メソッドも参照してください。

position

unsigned short position() const

画面のフィールドの開始位置を戻します。位置の決め方は、列番号 + (*n* x 行数) です。この場合の *n* は、行の中の列の数 (通常は 80) です。

resetDataTag

void resetDataTag()

修正データ・タグ (MDT) を *unmodified* にリセットします。

row

unsigned short row() const

画面のフィールドの開始位置の行番号を戻します。最上行は 1 です。

setBaseAttribute

void setBaseAttribute(char *attribute***)**

attribute

フィールドに入力される基本 3270 属性型の値。

3270 基本属性を設定します。

setExtAttribute

void setExtAttribute(char *attribute***, char** *value***)**

attribute

設定される拡張属性の型

value
拡張属性の値
拡張 3270 属性を設定します。無効な 3270 属性の型または値が指定されると、parameter 例外が生じます。

setText (1)

これらのメソッドは、所定のテキストでフィールドを更新します。

void setText(const char* *text*, **unsigned short** *length*)
text
フィールドに入力されるテキスト
length
フィールドに入力される文字数
text からの *length* 文字をフィールドにコピーします。

setText (2)

void setText(const char* *text*)
text
フィールドに入力されるヌル終了ストリング
text を、終了ヌルなしでフィールドにコピーします。

text

const char* text() const
現在フィールドに保持されているテキストを戻します。

textLength

unsigned short textLength() const
現在フィールドに保持されている文字数を戻します。

transparency

Transparency transparency() const
フィールドのバックグラウンドの透過性を示す列挙を戻します。使用可能な値を、このクラスの説明の終わりの **Transparency** の下に示します。

列挙

BaseInts

フィールドの輝度を示します。以下の値を使用することができます。

normal
intense
dark

BaseMDT

フィールドのデータが修正されているかどうかを示します。以下の値を使用することができます。

unmodified
modified

BaseProt

フィールドが保護されるかどうかを示します。以下の値を使用することができます。

protect
unprotect

BaseType

フィールド入力のデータ型を示します。以下の値を使用することができます。

alphanumeric
numeric

Color

以下の値を使用することができます。

defaultColor	yellow	paleGreen
blue	neutral	paleCyan
red	black	gray
pink	darkBlue	white
green	orange	
cyan	purple	

Highlight

使用される強調表示の型を示します。以下の値を使用することができます。

defaultHlt	blinkHlt	underscoreHlt
normalHlt	reverseHlt	intenseHlt

Transparency

フィールドのバックグラウンドの透過性を示します。以下の値を使用することができます。

defaultTran
: デフォルトの透過性

orTran
: 基礎カラーつき OR

xorTran
: 基礎カラーつき XOR

opaqueTran
: 不透明

第11章 CclFlow クラス

CclFlow オブジェクトは、クライアント / サーバーの対の ECI 通信を制御し、応答処理の同期を判別するときに使用されます。同期の説明については、13ページの『サーバーの対話の制御』を参照してください。**CclFlow** は、応答が使用可能なときに自動的にその **handleReply** メソッドを呼び出します。これで、インターリーブ応答の制御は単純になります。独自の **handleReply** メソッドをインプリメントするには、**CclFlow** をサブクラス化する必要があります。

CclFlow オブジェクトは、クライアント・サーバーの対話 (クライアントからの要求とサーバーからの応答) ごとに作成されます。CclFlow オブジェクトは非アクティブになる、つまり応答処理が完了すると、再使用できるようになります。アクティブ状態にある CclFlow オブジェクトを削除もしくは再使用しようとすると、activeFlow 例外を起こします。

CclFlow コンストラクター

CclFlow (1)

CclFlow(Ccl::Sync *syncType*, **unsigned long** *stackPages* **= 3)**

syncType
同期の型

stackPages
非同期ならば、4kb のスタック・ページ数。デフォルトは 3 です。非同期でなければ、このパラメーターは無視されます。

CclFlow (2)

CclFlow(Ccl::Sync *syncType*,
unsigned long *stackPages*,
const unsigned short &*timeout***)**

syncType
同期の型

stackPages
非同期ならば、4kb のスタック・ページ数。非同期でなければ、このパラメーターは無視されます。

timeout
ECI プログラムが応答するのを待つ秒単位の定義済みタイムアウト。タイムアウトが生じると、 HandleException メソッドが timeout CclException オブジェクトで呼び出されます。有効な値は 0 ～ 32767 です。

パブリック・メソッド

abendCode

const char* abendCode() const
最後に実行された CICS トランザクションからの異常終了コード、もしくは、ない場合はブランクを戻します。

callType

CallType callType() const
サーバー要求の最新の型を示す列挙値を戻します。

callTypeText

const char* callTypeText() const
最新のサーバー要求の名前を戻します。

connection

CclConn* connection() const
使用されているサーバーを示す CclConn オブジェクトへのポインター (ある場合)、もしくはゼロを戻します。

diagnose

const char* diagnose() const
診断出力で使用する例外を説明するテキストを戻します。以下にその例を示します。

```
"link", flowId=2, retCode=-22, abendCode="    "
```

flowId

unsigned short flowId() const
この CclFlow オブジェクトの固有の ID を戻します。

forceReset

void forceReset()
フローを非アクティブにし、リセットします。代表的には、フローを破棄した後に、たとえば、例外ハンドラーで C++ スローが使用されるときに、CclFlow

オブジェクトの再使用または削除に備えるのに使用します。これは dsync および async フローにのみ適用されます。これを他のスレッドから sync 呼び出しで発行することはできません。

handleReply

virtual void handleReply(CclBuf* *commarea***)**

commarea

戻された COMMAREA が含まれている CclBuf オブジェクトへのポインター、あるいはそれがない場合はゼロ。

このメソッドは、サーバーから応答が受信されるつど、同期の型または呼び出しの型に関係なく、呼び出されます。13ページの『サーバーの対話の制御』を参照してください。応答を処理するには、**CclFlow** をサブクラス化し、**handleReply** の独自のインプリメンテーションを行う必要があります。デフォルトのインプリメンテーションは、単に呼び出し側に戻るだけです。

listState

const char* listState() const

オブジェクトの現在の状態が入った定様式ストリングを戻します。例を以下に示します。

```
Flow state..&CclFlow=000489A4  &CclFlowI=00203B70
syncType=2  threadId=0  stackPages=9  callType=0  flowId=0  commLength=0
retCode=0  systemRC=0  abendCode="    "  &CclConnI=00000000  &CclUOWI=00000000
```

poll

Ccl::Bool poll(CclBuf* *commarea* **= 0)**

commarea

戻された COMMAREA を含めるときに使用する CclBuf オブジェクトを指す、オプションのポインター。

応答が、遅延同期の **Backout**、**Cancel**、**Changed**、**Commit**、**Link**、もしくは **Status** の呼び出し要求から受信されたかを示す、**Ccl** クラス内で定義された、列挙を戻します。**poll** が遅延同期ではないフロー・オブジェクトで使用されると、syncType 例外を生じます。以下の値を使用することができます。

yes 応答が受信されました。**handleReply** は同期的に呼び出されました。

no 応答は受信されませんでした。クライアント処理はブロックされません。

setTimeout

void setTimeout(const unsigned short &*timeout***)**

timeout

ECI プログラムが応答するのを待つ秒単位の定義済みタイムアウト。タイ

ムアウトが生じると、 HandleException メソッドが timeout CclException オブジェクトで呼び出されます。有効な値は 0 ～ 32767 です。
フローの次のアクティブ化のためにフロー・オブジェクトのタイムアウト値をセットします。フローがアクティブな間にこの値をセットすることができますが、現行のアクティブなフローには影響を与えません。

syncType

Ccl::Sync syncType() const
Ccl クラス内で定義された、使用される同期のタイプを示す列挙を戻します。使用可能な値を、39ページの『Sync』に示します。

timeout

short timeout()
フロー・オブジェクトにセットされた現行のタイムアウト値を検索します。

uow

CclUOW* uow() const
この対話に関連する作業単位 (UOW) に関する情報が入った、CclUOW オブジェクトがある場合は、それを指すポインターを戻します。

wait

void wait()
サーバーからの応答を待ち、その間クライアント処理をブロックします。同期フロー・オブジェクトに **wait** が使用されると、syncType 例外が生じます。

列挙

CallType

以下は、CclFlow オブジェクトの制御下で進行中のサーバー要求に対して使用可能な値です。

inactive
現在進行中のサーバー呼び出しはありません。

link サーバー・プログラムへの **CclConn::link** 呼び出し

backout
サーバー上のリカバリー可能リソースに対して行われた変更をバックアウトする **CclUOW::backout** 呼び出し

commit
サーバー上のリカバリー可能リソースに対して行われた変更をコミットする **CclUOW::commit** 呼び出し

status
サーバー接続の状況を判別する **CclConn::status** 呼び出し

changed
サーバーへの接続の状況が変更になったときの通知を要求する **CclConn::changed** 呼び出し

cancel
それまでの **CclConn::changed** 要求を取り消す **CclConn::cancel** 呼び出し

第12章 CclMap クラス

CclMap クラスは、CICS BMS マップ変換ユーティリティーによって作成されたマップ・クラスの基底クラスです。**CclMap** クラスによって指定されたメソッドは、BMS マップから生成されたクラスによって継承されます。

CclMap コンストラクター

CclMap(CclScreen* *screen***)**

screen

一致 CclScreen オブジェクトを指すポインター。

CclMap オブジェクトを作成し、マップが、CclScreen オブジェクトによって定義された画面の内容に一致しているか検査 (妥当性検査) します。妥当性検査で正しくなかった場合は、invalidMap 例外が生じます。指定された CclScreen オブジェクトが無効な場合は、parameter 例外が生じます。

パブリック・メソッド

exCode

使用すべきでないメソッド

新規アプリケーションではこのメソッドを使用しないでください。このメソッドは使用すべきでなく、旧バージョンとの互換性の目的でのみ提供されています。

Ccl::ExCode exCode() const

最新の例外コードを示す列挙を戻します。使用可能な値は、 105ページの表3にリストされています。

exCodeText

> **使用すべきでないメソッド**
> 新規アプリケーションではこのメソッドを使用しないでください。このメソッドは使用すべきでなく、旧バージョンとの互換性の目的でのみ提供されています。

const char* exCodeText() const
最新の例外コードを記述するテキスト・ストリングを戻します。

field (1)

CclField* field(unsigned short *index***)**

index
必要な CclField オブジェクトの索引番号。

BMS マップの *index* で識別された CclField オブジェクトを指すポインターを戻します。

field (2)

CclField* field(unsigned short *row***, unsigned short** *column***)**

row
マップ内の必要 CclField オブジェクト の行番号。1 は、最上位行を意味します。

column
マップ内の必要 CclField オブジェクト の列番号。1 は、左の列を意味します。

BMS マップの位置で識別された CclField オブジェクトを指すポインターを戻します。

保護メソッド

namedField

CclField* namedField(unsigned long *index***)**

index
必要な CclField オブジェクトの索引番号。

*index*th オブジェクトのアドレスを戻します。

validate

void validate(const MapData* *map*,
const FieldIndex* *index*,
const FieldData* *fields*)

map

マップに関する情報が入った構造体。構造体は、このクラスで定義され、すべて無符号短整数の以下のメンバーが入っています。

row 画面上のマップ行位置

col 画面上のマップ列位置

width

列のマップ幅

depth

行のマップ奥行き

fields

フィールド数

labels

ラベルつきフィールドの数

index

必要な CclField オブジェクトの索引番号。**FieldIndex** は、このクラスの typedef で、無符号短整数と同じです。

fields

特定のフィールドに関する情報が入った構造体。構造体は、このクラスで定義され、すべて無符号短整数の以下のメンバーが入っています。

row フィールド行 (マップ内の)

col フィールド列 (マップ内の)

len フィールド長

現行画面に照らしてマップを妥当性検査します。

第13章 CclScreen クラス

CclScreen EPI クラスは、すべてのデータを 3270 仮想画面上に保ち、このデータにアクセスします。これには、現行の 3270 画面上のフィールドを表す一連の CclField オブジェクトが入っています。

単一の CclScreen オブジェクトは、CclTerminal オブジェクトによって作成され、CclTerminal オブジェクトの **screen** メソッドを用いて取得する必要があります。CclScreen オブジェクトは、3270 データが CICS から受信されるときに、CclTerminal オブジェクトによって更新されます。サポートされていないデータ・ストリームが受信されると、dataStream 例外が生じます。

パブリック・メソッド

cursorCol

unsigned short cursorCol() const
カーソルの現在位置の列番号を戻します。左端列 = 1。

cursorRow

unsigned short cursorRow() const
カーソルの現在位置の行番号を戻します。最上位行 = 1。

depth

unsigned short depth() const
画面の行数を戻します。

field (1)

これらのメソッドを使用して、ポインターを当該 CclField オブジェクトに向けると、現行画面のフィールドにアクセスすることができます。

CclField* field(unsigned short *index***)**
index
当該フィールドの索引番号

field (2)

CclField* field(unsigned short *row*, **unsigned short** *column*)

row

フィールドの行番号

column

フィールドの列番号

fieldCount

unsigned short fieldCount() const

画面のフィールド数を戻します。

mapName

const char* mapName()

端末リソース用に処理される SEND MAP コマンドの MAP オプションで参照される最新のマップ名を指定しているストリングで、終わりをヌルで埋めたストリングを戻します。端末リソースが BMS でサポートされない場合、または送信されるいずれのマップにもレコードがない場合は、戻される値はスペースです。

mapSetName

const char* mapSetName()

端末リソース用に処理される SEND MAP コマンドの MAPSET オプションで参照される最新のマップ・セット名を指定しているストリングで、終わりをヌルで埋めたストリングを戻します。最新の要求で MAPSET オプションが指定されていない場合は、 BMS マップ・セット名としてマップ名を使用しました。両方の事例で、使用されるマップ・セット名は端末接尾部によって接尾部が付けられます。端末リソースが BMS でサポートされない場合、または送信されるいずれのマップ・セットにもレコードがない場合は、戻される値はスペースです。

setAID

void setAID(const AID *key*)

key

AID キー。この章の終わりの AID 列挙を参照してください。

次の伝送のサーバーに渡される AID キー値を設定します。

setCursor

void setCursor(unsigned short *row*, **unsigned short** *col*)

row

カーソルの必要行番号。1 は、最上位行を意味します。

col

カーソルの必要列番号。1 は、左の列を意味します。

カーソル位置の設定を要求します。指定された行または列の値が、画面の境界外にあると、parameter 例外を生じます。

width

unsigned short width() const

画面上の列数を戻します。

列挙

AID

AID キーを示します。以下の値を使用することができます。

enter
clear
PA1 ～ PA3
PF1 ～ PF24

第14章 CclSecAttr

CclSecAttr クラスは、CclConn または CclTerminal オブジェクトで、verifyPassword または changePassword メソッドを発行したときに、外部セキュリティー・マネージャーによって報告されるパスワードについての情報を提供します。

このオブジェクトは CclConn または CclTerminal オブジェクトによって作成および所有され、またこのオブジェクトへのアクセスは verifyPassword または changePassword メソッドを呼び出すと提供されます。

パブリック・メソッド

expiryTime

CclSecTime* expiryTime() const

パスワードの有効期限が切れる日付と時間を含む CclSecTime オブジェクトを戻します。

invalidCount

unsigned short invalidCount() const

そのユーザー ID に無効なパスワードを入力した回数を戻します。

lastAccessTime

CclSecTime* lastAccessTime() const

そのユーザー ID が最後にアクセスされた日付と時間を含む CclSecTime オブジェクトを戻します。

lastVerifiedTime

CclSecTime* lastVerifiedTime() const

最新の検査の日付と時間を含む CclSecTime オブジェクトを戻します。

第15章 CclSecTime

CclSecTime クラスは、CclConn または CclTerminal オブジェクトで、verifyPassword または changePassword メソッドを発行したときに、外部セキュリティー・マネージャーによって報告されるさまざまなエントリー用の CclSecAttr オブジェクトに日付と時間の情報を提供します。

これらのオブジェクトは CclSecAttr オブジェクトによって作成および所有され、オブジェクトへのアクセスはこのオブジェクトで選択可能なさまざまなメソッドを通じて獲得できます。選択可能なコンストラクターとデストラクターはありません。

パブリック・メソッド

day

unsigned short day() const

1 から 31 の範囲の日付を戻します。1 は月の最初の日を表します。

get_time_t

time_t get_time_t() const

time_t 形式で日付と時間を戻します。

get_tm

tm get_tm() const

tm 構造で日付と時間を戻します。

hours

unsigned short hours() const

0 から 23 の範囲で時間を戻します。

hundredths

unsigned short hundredths() const

0 から 99 の範囲で百分の一秒を戻します。

minutes

unsigned short minutes() const

0 から 59 の範囲で分を戻します。

month

unsigned short month() const

1 から 12 の範囲で月を戻します。1 は 1 月を表します。

seconds

unsigned short seconds() const

0 から 59 の範囲で秒を戻します。

year

unsigned short year() const

4 桁の年を戻します。

第16章 CclSession クラス

プログラマーは、**CclSession** クラスを使用すると、3270 変換のセグメント (1 つまたは複数の伝送) を処理する再使用可能コードをインプリメントすることができます。マルチスレッド環境の場合、このクラスは CICS からの応答を非同期処理します。

CclSession クラスは、単一 3270 セッション内の CICS との間のデータのフローを制御します。**CclSession** から独自のクラスを派生させる必要があります。

CclSession コンストラクター

CclSession(Ccl::Sync *syncType***)**

syncType

CICS サーバーへの伝送で使用するプロトコル。以下の値を使用することができます。

async	非同期
dsync	遅延同期
sync	同期

パブリック・メソッド

diagnose

const char* diagnose() const

最後のエラーのテキスト記述を戻します。

handleReply

virtual void handleReply(State *state***, CclScreen*** *screen***)**

state

データ・フローの状態を示す列挙。値の有効範囲は、このクラス内であり、このクラスの説明の終わりの **State** の下にあります。

screen

CclScreen オブジェクトを指すポインター

これは、**CclSession** から派生した独自のクラスを開発するときにオーバーライドできる仮想メソッドです。これは、データが CICS から受信されたときに呼び出されます。

state

State state() const
セッションの現在の状態を示す列挙を戻します。使用可能な値を、このクラスの説明の終わりの **State** の下に示します。

terminal

CclTerminal* terminal() const
このセッションの CclTerminal オブジェクトを指すポインターを戻します。このメソッドは、CclSession オブジェクトが CclTerminal オブジェクトに関連付けられるまで (すなわち、CclSession オブジェクトが CclTerminal **send** メソッドのパラメーターとして使用されるまで)、ヌル・ポインターを戻すので、注意してください。

transID

const char* transID() const
現行トランザクションの 4 文字の名前を戻します。

列挙

State

セッションの状態を示します。以下の値を使用することができます。

idle 端末は接続され、かつ進行中の CICS トランザクションはありません。

server
サーバーに進行中の CICS トランザクションがあります。

client
進行中の CICS トランザクションがあり、サーバーは、クライアントからの応答待ちです。

discon
端末は切断されています。

error 端末にエラーがあります。

第17章 CclTerminal クラス

クラス **CclTerminal** のオブジェクトは、CICS サーバーへの 3270 端末の接続を表します。CICS 接続は、オブジェクトの作成時に確立されます。これで、CICS サーバーの 3270 端末アプリケーション (BMS アプリケーションの場合が多い) との会話に、メソッドを使用することができます。

CclTerminal オブジェクトを作成するには、EPI を初期化しておく必要があります。 (すなわち、CclEPI オブジェクトの作成)。

CclTerminal コンストラクター

CclTerminal (1)

CclTerminal(const char* *server* **= NULL,**
const char* *devtype* **= NULL,**
const char* *netname* **= NULL)**

server
通信する必要があるサーバーの名前。名前が指定されない場合は、デフォルトのサーバー・システムとみなされます。長さは、ブランクの埋め込みによって 8 文字に調整されます。

devtype
サーバーが端末リソース定義の生成に使用するモデル端末定義の名前。ストリングが指定されない場合は、デフォルトのモデルが使用されます。長さは、ブランクの埋め込みによって 16 文字に調整されます。

netname
インストールもしくは予約する端末リソースの名前。デフォルトでは、*devtype* の内容を使用します。長さは、ブランクの埋め込みによって 8 文字に調整されます。

クライアントとサーバー間の EPI 通信に使用される CclTerminal オブジェクトを作成します。

このコンストラクターは、暗黙的なインストール端末の役割を果たします。この方法で端末オブジェクトを構成する場合は、インストール・メソッドを呼び出す必要はありません。

指定されたサーバーが クライアント初期設定ファイル で構成されていない場合は、unknownServer 例外が発生します。

server、*devtype*、または *netname* に無効な値が指定された場合は、parameter 例外が発生します。

CclEPI オブジェクトが作成されていない場合は、initEPI 例外が発生します。

サポートされている端末接続の数が最大数を超えている場合は、 maxRequests 例外が発生します。

CclTerminal (2)

CclTerminal(const char* *server*,
const char* *devtype*,
const char* *netname*,
signonType *signonCapability*
const char* *userid*
const char* *password*
const unsigned short *&readTimeOut*,
const unsigned short *CCSid*)

server
通信する必要があるサーバーの名前。名前が指定されない場合は、デフォルトのサーバー・システムとみなされます。長さは、ブランクの埋め込みによって 8 文字に調整されます。

devtype
サーバーが端末リソース定義の生成に使用するモデル端末定義の名前。ストリングが指定されない場合は、デフォルトのモデルが使用されます。長さは、ブランクの埋め込みによって 16 文字に調整されます。

netname
インストールもしくは予約する端末リソースの名前。デフォルトでは、*devtype* の内容を使用します。長さは、ブランクの埋め込みによって 8 文字に調整されます。

signonCapability

端末のサインオン機能のタイプをセットします。

以下の値を使用することができます。

```
CclTerminal::SignonCapable
CclTerminal::SignonIncapable
```

userid
このリソースに関連させるユーザー ID の名前。

password
このユーザー ID と関連するパスワード。

readTimeOut
0 から 3600 の範囲の値で、そのクラスが clientrepl 状態に進みアプリケーション・プログラムが応答メソッドを呼び出す間の最大時間を秒数で指定します。
CCSid
終端リソースと CICS トランザクション間で渡されるデータについて、クライアント・アプリケーションが使用するコード化図形文字セットを識別するための、コード化文字セット ID (CCSID) を指定する符号なし単形式。ゼロのストリングはデフォルトの使用を暗黙指定します。
暗黙のインストール端末を行わない Terminal オブジェクトを作成します。その端末をインストールするためには install メソッドを実行しなければなりません。

パブリック・メソッド

alterSecurity

void alterSecurity(const char* *userid*,**const char*** *password*)

userid
新規のユーザー ID。

password
userid の新規パスワード
端末リソースにユーザー ID とパスワードを再定義できるようにします。端末をインストールする前にメソッドを呼び出すことができます。端末定義のみを変更するので、新規のユーザー ID とパスワードは install が呼び出されるとその端末で使用されます。

changePassword

CclSecAttr* changePassword(const char* newPassword)

newPassword
新規パスワード。
端末オブジェクトに保持されているパスワードと、端末オブジェクトに保持されているユーザー ID 用に外部セキュリティー・マネージャーが記録したパスワードを、クライアント・アプリケーションが変更できるようにします。外部セキュリティー・マネージャーは端末オブジェクトによって定義されたサーバーに置かれているとみなされます。

CCSid

unsigned short CCSid()
符号なし単形式として選択済みコード・ページを戻します。

diagnose

const char* diagnose()
最新のサーバー呼び出しが戻すエラーの記述を保持する、文字ストリングを戻します。

disconnect(1)

void disconnect()
端末を CICS から切断します。未解決の実行中のトランザクションをパージする作業は行われません。

disconnect(2)

void disconnect(Ccl::Bool *withPurge***)**

withPurge

Ccl::Yes
端末を CICS から切断し、未解決の実行中のトランザクションのパージを試行します。

Ccl::No
端末を CICS から切断します。未解決の実行中のトランザクションをパージする作業は行われません。

discReason

void discReason(void)
切断の理由を戻します。100ページの『EndTerminalReason』を参照してください。

exCode

使用すべきでないメソッド
新規アプリケーションではこのメソッドを使用しないでください。このメソッドは使用すべきでなく、旧バージョンとの互換性の目的でのみ提供されています。

Ccl::ExCode exCode() const
最新の例外コードを示す列挙を戻します。使用可能な値は、105ページの表3 にリストされています。

exCodeText

> **使用すべきでないメソッド**
> 新規アプリケーションではこのメソッドを使用しないでください。このメソッドは使用すべきでなく、旧バージョンとの互換性の目的でのみ提供されています。

const char* exCodeText() const
最新の例外コードを記述するテキスト・ストリングを戻します。

install

void install(CclSession *_session_,
const unsigned short &_installTimeOut_)

session
CICS サーバー対話で使用される CclSession オブジェクトへのポインター。

installTimeOut
0 から 3600 の範囲の値で、端末リソースのインストールで取ることのできる時間の最大を秒数で指定します。 0 の値は限界をセットしないこと意味します。

非接続端末リソースに接続します。すでに接続されている場合は invalidState エラーを出すか、またはタイムアウトになるとタイムアウト・エラーを出します。

makeSecurityDefault

void makeSecurityDefault()
端末オブジェクトの構造体で指定したように、サーバーに渡される ECI および EPI 要求のデフォルトにこのオブジェクトの現行ユーザー ID とパスワードがなることを、クライアントに通知します。

netName

const char* netName() const
端末のネットワーク名をヌル終了ストリングとして戻します。

password

const char* password()
その端末の現行パスワード設定値を含むヌル終了ストリングを戻します。ない場合はヌルを戻します。

poll

Ccl::Bool poll()
CICS サーバーからのデータをポーリングします。

遅延同期伝送 (つまり、前の送信呼び出しで遅延同期 CclSession オブジェクトが使用された場合) では、アプリケーションが CICS サーバーからデータを受信したいときに、**poll** メソッドを呼び出します。CICS からの応答が受信されると、 CclTerminal オブジェクトは、CclScreen オブジェクトを CICS から受信した 3270 データ・ストリームの内容で更新し、 CclScreen オブジェクトの **handleReply** 仮想関数が呼び出されて、**poll** メソッドは `Ccl::yes` を戻します。CICS からの応答がまだ受信されていない場合、**poll** メソッドは `Ccl::no` を戻します。

poll メソッドを使用するのは、遅延同期伝送の場合に限られます。poll メソッドが、同期セッションまたは非同期セッションが使用中のときに呼び出されると、syncType 例外を生じます。それまでに **send** 呼び出しがなかったときに **poll** メソッドが呼び出されると、invalidState 例外を生じます。ポーリングを呼び出すには、CclTerminal オブジェクトが server 状態になければなりません。

CICS サーバー・トランザクションは、CclTerminal **send** 呼び出しに応答して、複数の応答を送信する場合があります。したがって、すべての応答を収集するには、複数の CclTerminal **poll** 呼び出しが必要になる場合があります。まだ応答があるかどうかを調べるには、CclTerminal **state** メソッドを使用します。その場合に戻される値は、server です。 26ページの『EPI 呼び出し同期タイプ』を参照してください。

queryATI

ATIState queryATI()
「自動トランザクション始動」(ATI) が使用可能か使用不能かを示す列挙を戻します。以下の値を使用することができます。

disabled
enabled

readTimeout

const char* readTimeout()

端末の読み取りタイムアウト値をヌル終了ストリングとして戻します。

receiveATI

void receiveATI(CclSession* *session***)**

session

CICS サーバー対話で使用される CclSession オブジェクトへのポインター。

CICS ATI トランザクションで 3270 データ・ストリームを待ち、受信します。パラメーターとして提供された CclSession オブジェクトは、呼び出しが同期か非同期かを判別し、応答ハンドラーを提供するためにサブクラス化することができます。

screen

CclScreen* screen() const

この端末セッションに関連する 3270 画面を処理する、CclScreen オブジェクトを指すポインターを戻します。

send (1)

void send(CclSession* *session***,**
const char* *transid***,**
const char* *startdata* **= NULL)**

session

使用されるセッションを制御する、CclSession オブジェクトを指すポインター。指定された CclSession オブジェクトが無効な場合は、parameter 例外を生じます。

transid

開始されるトランザクションの名前

startdata

トランザクション・データを開始します。デフォルト設定では、開始されるトランザクションのデータはありません。

3270 データ・ストリームをフォーマットして、送信し、指名したトランザクションを開始します。CclTerminal オブジェクトは、idle 状態になければなりません。 (すなわち、CICS サーバーに接続されていても、進行中のトランザクションがない)。オブジェクトが idle 状態にない場合は、invalidState 例外を生じます。

send (2)

void send(CclSession* *session*)

session パラメーターについては前述しました。

3270 データ・ストリームをフォーマットして、送信します。CclTerminal オブジェクトは、idle 状態 (上記を参照) か client 状態 (すなわち、トランザクションが進行中で、CICS サーバーが応答待ち) になければなりません。オブジェクトが idle か client 状態にない場合は、invalidState 例外を生じます。

setATI

void setATI(ATIState *newstate*)

newstate

ATI が使用可能になるか、使用不能になるかを示す列挙。値の有効範囲は、このクラス内であり、使用できる値は disabled と enabled です。

signonCapability

signonType signonCapability()

インストール時に端末に適用されたサインオン機能のタイプを戻します。

以下の値を使用することができます。

```
CclTerminal::signonCapable
CclTerminal::signonIncapable
CclTerminal::signonUnknown
```

state

State state() const

セッションの現在の状態を示す列挙を戻します。使用可能な値を、このクラスの説明の終わりに示します。

serverName

const char* serverName() const

この端末セッションが接続される CICS サーバーの名前を戻します。

termID

const char* termID() const

4 文字の端末 ID を戻します。

transID

const char* transID() const

現行の CICS トランザクションの 4 文字の名前を戻します。現行のトランザクションから RETURN IMMEDIATE を実行すると、 TransId は新規トランザクションの名前を提供せず、最初のトランザクションの名前が含まれていることに注意してください。

userId

const char* userId()

その端末の現行ユーザー ID 設定値を含むヌル終了ストリングを戻します。ない場合はヌルを戻します。

verifyPassword

CclSecAttr* verifyPassword()

端末オブジェクトに保持されているパスワードと、検査オブジェクトに保持されているユーザー ID によって保持される外部セキュリティー・マネージャーが記録したパスワードが一致しているかどうかを、クライアント・アプリケーションが検査できるようにします。外部セキュリティー・マネージャーは端末オブジェクトによって定義されたサーバーに置かれているとみなされます。

列挙

ATIState

「自動トランザクション始動」(ATI) が使用可能か使用不能かを示します。以下の値を使用することができます。

enabled
disabled

signonType

端末のサインオン機能を示します。以下の値を使用することができます。

signonCapable
サインオン可能

signonIncapable
サインオン不可能

signonUnknown
サインオン不明

State

CclTerminal オブジェクトの状態を示します。以下の値を使用することができます。

client
進行中の CICS トランザクションがあり、サーバーは、クライアントからの応答待ちです。

discon
端末は切断されています。

error 端末にエラーがあります。

idle 端末は接続され、かつ進行中の CICS トランザクションはありません。

server
サーバーに進行中の CICS トランザクションがあります。

termDefined
定義済みで未インストールの端末。

EndTerminalReason

CclTerminal オブジェクトの EndTerminalReason を示します。以下の値を使用することができます。

signoff
切断が要求されたかまたはユーザーが端末をサインオフしました。

shutdown
CICS サーバーがシャットダウンされました。

outofService
端末はサービス適用外に切り替えられました。

unknown
不明状態が発生しました。

failed
端末の切断が失敗しました。

notDiscon
端末は切断されません。

第18章 CclUOW クラス

「作業単位」(UOW) 内のサーバーのリカバリー可能リソースを更新するときは、この ECI クラスを使用します。UOW の各更新は、その **CclUOW** を参照してクライアントで識別されます。**CclConn** の **link** (51ページの『link』) を参照してください。

CclUOW オブジェクトは、コピーすることも割り当てることもできません。CclFlow オブジェクトがアクティブ中の CclUOW オブジェクトを削除しようとすると、activeFlow 例外を生じます。アクティブ状態の CclUOW オブジェクト (つまり、コミットもバックアウトもされていない) を削除しようとすると、activeUOW 例外を生じます。

CclUOW コンストラクター

CclUOW()

CclUOW オブジェクトを作成します。

パブリック・メソッド

backout

void backout(CclFlow& *flow***)**

flow

クライアント - サーバー呼び出しの制御に使用する CclFlow オブジェクトへの参照。

この UOW を終了し、サーバーのリカバリー可能リソースに加えられたすべての変更をバックアウトします。

commit

void commit(CclFlow& *flow***)**

flow

クライアント - サーバー呼び出しの制御に使用する CclFlow オブジェクトへの参照。

この UOW を終了し、サーバーのリカバリー可能リソースに加えられたすべての変更をコミットします。

forceReset

void forceReset()

この UOW を非アクティブにし、リセットします。

listState

const char* listState() const

オブジェクトの現在の状態が入った、ゼロで終了する定様式ストリングを戻します。例を以下に示します。

```
UOW state..&CclUOW=0004899C  &CclUOWI=00203BD0
&CclConnI=00000000  uowId=0  &CclFlowI=00000000
```

uowId

unsigned long uowId() const

UOW の ID を戻します。0 は、UOW が完了しているか、まだ開始していないことを示します。つまり、非アクティブ状態です。

第3部 付録

付録. 例外オブジェクト

例外オブジェクトはすべて以下の情報を提供します。

- クラス名
- メソッド名
- 例外コード
- 例外テキスト
- 異常終了コード (ECI のみ)
- 原点

クラス名には、CclFlowI などのように後ろに「I」を含む場合があります。これは既知のクラスについて、内部に含まれるクラスを意味します。すなわち CclFlowI は CclFlow に含まれています。内部クラスが報告される場合は、報告されたメソッドは外部のものではなく、内部メソッドです。

原点は、クラス・ライブラリー内で例外が生成された正確なポイントを定義する固有の値です。これらは主に保守用です。

より重要な情報の項目は例外コード、例外テキスト、および異常終了コード (ECI のみ) です。次にこれらの例外コード、テキスト、およびそれらが ECI、EPI、または両方と関連するかどうかをまとめてあります。

表 3. 例外コード

一覧	テキスト	説明	ECI	EPI
Ccl::noError	no error	エラーは生じなかった。	O	O
Ccl::bufferOverflow	buffer overflow	拡張可能ではない CclBuf オブジェクトを拡張しようとした。	O	
Ccl::multipleInstance	multiple instance	複数のオブジェクトを作成しようとした。	O	
Ccl::activeFlow	flow is active	現行の流れはまだアクティブであり、この流れが非アクティブになるまでこの流れは使用できない。	O	

表 3. 例外コード (続き)

一覧	テキスト	説明	ECI	EPI
Ccl::activeUOW	UOW is active	現行の UOW はまだアクティブであり、バックアウトまたはコミットが必要である。	O	
Ccl::syncType	sync error	メソッド呼び出しの同期タイプが誤り。	O	O
Ccl::threadCreate	thread create error	内部スレッド作成エラー。	O	O
Ccl::threadWait	thread wait error	内部スレッド待機エラー。	O	
Ccl::threadKill	thread kill error	内部スレッド強制終了エラー。	O	
Ccl::dataLength	data length invalid	CommArea > 32768 バイトまたはインバウンド 3270 データ・ストリームが端末バッファー・サイズに対して大き過ぎる。	O	O
Ccl::noCICS	no CICS	クライアントが選択不可能である、またはサーバー・インプリメンテーションが選択不可能である、または作業論理単位は開始されているが、指定した CICS サーバーが選択不可能である。リソースは更新されない。	O	O

表 3. 例外コード (続き)

一覧	テキスト	説明	ECI	EPI
Ccl::CICSDied	CICS died	作業論理単位が開始または継続されるはずであったが、その CICS サーバーはもはや選択不可能である。これがアクティブ UOW を使用したリンク呼び出しであれば、変更はバックアウトされる。これが UOW コミットまたはバックアウトの場合、アプリケーションは変更がコミットされたかバックアウトされたかを判別できず、今後の手作業によるリカバリーのために、この条件をログに記録しなければならない。	O	
Ccl::noReply	no reply	未解決の応答はない。	O	
Ccl::transaction	transaction abend	ECI プログラムの異常終了。	O	
Ccl::systemError	system error	不明な内部エラーが生じた。	O	O
Ccl::resource	resource shortage	サーバー・インプリメンテーションまたはクライアントが要求を完了するのに十分なリソースを持っていなかった。すなわち不十分な SNA セッション。	O	O
Ccl::maxUOWs	exceeded max UOWs	作業論理単位を作成しようとしたが、アプリケーションはすでに構成がサポートする数と同じだけ未解決の作業論理単位を持っている。	O	
Ccl::unknownServer	unknown server	要求されたサーバーは探し出せない。	O	O

表 3. 例外コード (続き)

一覧	テキスト	説明	ECI	EPI
Ccl::security	security error	ユーザーは、サーバーが予測している有効な組み合わせのユーザー ID とパスワードを提供しなかった。	O	O
Ccl::maxServers	exceeded max servers	ユーザーの構成で許可された以上のサーバーに要求を開始しようとした。ユーザーのクライアントまたはサーバー用の資料で、使用できるサーバー数を制御する方法を調べてください。	O	O
Ccl::maxRequests	exceeded max requests	要求を満足する十分な通信リソースがない。ユーザーのクライアントまたはサーバー用の資料で、通信リソースを制御する方法を調べてください。	O	O
Ccl::rolledBack	rolled back	作業論理単位をコミットしようとしたが、サーバーはその変更をコミットできず、その代わりバックアウトした。	O	
Ccl::parameter	parameter error	指定したパラメーターの誤り。	O	O
Ccl::invalidState	invalid object state	メソッドを呼び出すのにオブジェクトは正しい状態ではない。すなわち端末オブジェクトはまだサーバー状態で、データ送信の試みがなされた。	O	O
Ccl::transId	invalid transaction	疑似会話トランザクションにヌルのトランザクション ID が提供されたかまたは戻された。		O
Ccl::initEPI	EPI not initialised	EPI オブジェクトがないまたは正しい初期設定に失敗した。		O

表 3. 例外コード (続き)

一覧	テキスト	説明	ECI	EPI
Ccl::connect	connection failed	端末の追加時に予期しないエラーが生じた。		O
Ccl::dataStream	3270 datastream error	非サポート・データ・ストリーム		O
Ccl::invalidMap	map/screen mismatch	マップ定義と画面が一致しない。		O
Ccl::cclClass	CICS class error	不明な内部クラス・エラーが生じた。	O	O
Ccl::startTranFailure	Start Transaction Failure	トランザクションの開始が失敗。		O
Ccl::timeout	Timeout Occurred	サーバーからの応答が返る前にタイムアウトが発生した。	O	O
Ccl::noPassword	Password is Null	オブジェクトのパスワードがヌルである。	O	O
Ccl::noUserid	Userid is Null	オブジェクトのユーザー ID がヌルである。	O	O
Ccl::nullNewPassword	A NULL new password was supplied	提供されたパスワードがヌルである。	O	O
Ccl::pemNotSupported	PEM is not supported on the server	CICS サーバーはパスワード満了管理機能をサポートしない。このメソッドを使用することはできない。	O	O
Ccl::pemNotActive	PEM is not active on the server	パスワード満了管理機能ではない。	O	O
Ccl::passwordExpired	Password has expired	パスワードが満了した。情報は戻らない。	O	O
Ccl::passwordInvalid	Password is invalid	パスワードが無効である。	O	O
Ccl::passwordRejected	New password was rejected	パスワードが定義された標準に準拠しないのでパスワード変更が失敗した。	O	O
Ccl::useridInvalid	Userid unknown at server	ユーザー ID が不明。	O	O
Ccl:invalidTermid	Termid is invalid	端末 ID が無効。		O

表 3. 例外コード (続き)

一覧	テキスト	説明	ECI	EPI
Ccl:invalidModelid	Modelid is invalid	無効なモデル / 装置 タイプ		O
Ccl:not3270	3270 デバイスではない	3270 デバイスではない		O
Ccl:invalidCCSid	Codepage (CCSid value) is invalid	無効な CCSid		O
Ccl:serverBusy	Server is too busy	CICS サーバーが使用中		O
Ccl:signonNotPossible	Signon Capable terminal is not possible	端末をサインオン可能としてインストールすることをサーバーが許可しない。		O

用語集

AID. アテンション ID (Attention Identifier)

ATI. 自動トランザクション開始 (Automatic Transaction Initiation)

BMS. 基本マッピング・サポート

COMMAREA. CICS 通信域 (CICS Communications Area)

ECI. 外部呼び出しインターフェース (External Call Interface)

EPI. 外部表示インターフェース (External Presentation Interface)

ESI. 外部セキュリティー・インターフェース (External Security Interface)

ISC. システム間連絡 (Inter-Systems Communication)

OLE. オブジェクト・リンキング & エンベッディング (Object Linking and Embedding)

UOW. 作業単位 (Unit of Work)

参考文献

以下の資料を参考にご利用ください。

C++ プログラミング

ご使用の C++ コンパイラーとともに提供される資料を参照してください。

以下に、一般に入手可能な IBM 以外の資料をリストします。このリストはすべてを網羅するものではなく、また、弊社が以下の資料を特に勧めするというわけではありません。ここにリストした以外の資料をお近くの図書館や書店で入手することも可能です。

- Lippman, Stanley B., *C++ Primer* : Addison-Wesley Publishing Company.
- Stroustrup, Bjarne, *The C++ Programming Language* : Addison-Wesley Publishing Company.
- Ellis, Margaret A. and Bjarne Stroustrup, *The Annotated C++Reference Manual* : Addison-Wesley Publishing Company.

CICS Transaction Gateway および CICS ユニバーサル・クライアントのライブラリー

この章では、すべての CICS Transaction Gateway、CICS ユニバーサル・クライアント、および関連資料をリストし、それらが使用できるさまざまな形式を説明します。

この章で扱う見出し項目は、次のとおりです。

CICS Transaction Gateway 資料

- *CICS Transaction Gateway: Windows® Gateway 管理*、*SC88-8984*

  この資料では、CICS® Transaction Gateway (Windows® 版) の管理について説明します。

- *CICS Transaction Gateway: AIX® Gateway 管理*、*SC88-8985*

  この資料では、CICS® Transaction Gateway (AIX® 版) の管理について説明します。

- *CICS Transaction Gateway: Solaris Gateway 管理*、*SC88-8986*

  この資料では、CICS® Transaction Gateway (Solaris 版) の管理について説明します。

- *CICS Transaction Gateway: Linux Gateway 管理*、*SC88-8989*

  この資料では、CICS® Transaction Gateway (Linux 版) の管理について説明します。

- *CICS Transaction Gateway: HP-UX Gateway 管理*、*SC88-8988*

  この資料では、CICS® Transaction Gateway (HP-UX 版) の管理について説明します。

- *CICS Transaction Gateway: OS/390® Gateway 管理*、*SC88-8987*

この資料では、CICS® Transaction Gateway (OS/390® 版) の管理について説明します。

- *CICS Transaction Gateway: Gateway Messages*

  このオンライン・ブックでは、CICS Transaction Gateway によって生成されることがあるエラー・メッセージをリストし説明します。

  この資料をオーダーすることはできません。

- *CICS Transaction Gateway: Gateway プログラミング*、SC88-8990

  この資料では、 CICS Transaction Gateway を使用した Java™ プログラミングの概要を提供します。

  プログラミング参照情報を含む追加の HTML ページもあります。

CICS ユニバーサル・クライアント資料

- *CICS Transaction Gateway: Windows® クライアント管理*、*SC88-8991*

  この資料では、CICS ユニバーサル・クライアント (Windows 版) の管理について説明します。

- *CICS Transaction Gateway: AIX® クライアント管理*、*SC88-8992*

  この資料では、CICS ユニバーサル・クライアント (AIX 版) の管理について説明します。

- *CICS Transaction Gateway: Solaris クライアント管理*、*SC88-8993*

  この資料では、CICS ユニバーサル・クライアント (Solaris 版) の管理について説明します。

- *CICS Transaction Gateway: Linux クライアント管理*、*SC88-8995*

  この資料では、CICS ユニバーサル・クライアント (Linux 版) の管理について説明します。

- *CICS Transaction Gateway: HP-UX クライアント管理*、*SC88-8994*

  この資料では、CICS ユニバーサル・クライアント (HP-UX 版) の管理について説明します。

- *CICS Transaction Gateway: Client Messages*

  このオンライン・ブックでは、 CICS ユニバーサル・クライアントによって生成されることがあるエラー・メッセージとトレース・メッセージをリストし説明します。

  この資料をオーダーすることはできません。

- *CICS Transaction Gateway: C++ プログラミング*、SC88-8996

  この資料では、 C++ 言語を使用して ECI と EPI 用のオブジェクト指向プログラムを作成する方法を説明します。

- *CICS Transaction Gateway COM オートメーション・プログラミング*、SC88-8997

  この資料では、コンポーネント・オブジェクト・モデル (COM) 規格に従って、 ECI と EPI 用のオブジェクト指向プログラムを作成する方法を説明します。

CICS ファミリー資料

- *CICS® ファミリー: クライアント / サーバー・プログラミング*、SC88-8998

  この資料では、 CICS クライアント / サーバー・プログラミング — すなわち外部呼び出しインターフェース (ECI)、外部表示インターフェース (EPI)、および外部セキュリティー・インターフェース (ESI) と関連したプログラミング・インターフェ

ースを説明します。このブックは CICS サーバー・システムと通信する目的で、クライアント・アプリケーションを開発するアプリケーション設計者とプログラマーを対象としています。

資料のファイル名

表4 は、CICS Transaction Gateway および CICS ユニバーサル・クライアントの資料のソフトコピー・ファイル名を示しています。

表 4. CICS Transaction Gateway および CICS ユニバーサル・クライアントの資料のファイル名

資料タイトル	ファイル名
CICS Transaction Gateway: Client Messages	CCLIAB
CICS Transaction Gateway: AIX® クライアント管理	CCLIAD
CICS Transaction Gateway: Windows® クライアント管理	CCLIAF
CICS Transaction Gateway: Solaris クライアント管理	CCLIAG
CICS Transaction Gateway: Linux クライアント管理	CCLIAR
CICS Transaction Gateway: HP-UX クライアント管理	CCLIAT
CICS Transaction Gateway: OS/390® Gateway 管理	CCLIAI
CICS Transaction Gateway: Gateway Messages	CCLIAJ
CICS Transaction Gateway: Gateway プログラミング	CCLIAK
CICS Transaction Gateway: Windows® Gateway 管理	CCLIAL
CICS Transaction Gateway: AIX® Gateway 管理	CCLIAN
CICS Transaction Gateway: Solaris Gateway 管理	CCLIAO
CICS Transaction Gateway: Linux Gateway 管理	CCLIAS
CICS Transaction Gateway: HP-UX Gateway 管理	CCLIAU
CICS Transaction Gateway: C++ プログラミング	CCLIAP
CICS Transaction Gateway COM オートメーション・プログラミング	CCLIAQ
CICS® ファミリー: クライアント / サーバー・プログラミング	DFHZAD
注: このテーブルのファイル名には二桁の接尾部を含んでいません。	

サンプル構成資料

多くのサンプル構成資料が PDF 形式で使用可能です。

これらの資料では、たとえば、さまざまな構成プロトコルを使用して、CICS サーバーと通信するためにCICS ユニバーサル・クライアントを構成するためのステップバイステップの指針を提供します。それらは、CICS Transaction Gateway および CICS ユニバーサル・クライアント・ライブラリーの情報を拡張する詳細な指示を提供します。

サンプル構成文書が選択可能になる場合は、Web サイトからそれらをダウンロードすることができます。次に進み、

`www.ibm.com/software/ts/cics/`

Library のリンクに従ってください。

その他の資料

次のインターナショナル・テクニカル・サポート編成 (ITSO) レッドブック™資料には、クライアント / サーバー構成の多くの例が記載されています。

- *Revealed! CICS Transaction Gateway with more CICS Clients Unmasked*、*SG24-5277*

  この資料は以下の資料を置き換えます。
- *CICS Clients Unmasked*、*GG24-2534*

多くのソースから ITSO レッドブックを入手することができます。最新の情報については、以下を参照してください。

`www.ibm.com/redbooks/`

以下で CICS プロダクトの情報を見ることができます。

`www.ibm.com/software/ts/cics/`

オンライン資料の表示

CICS Transaction Gateway と CICS ユニバーサル・クライアントが提供するすべての資料を弊社のオンライン・ライブラリーでアクセスすることができます (英語のみ)。オンライン・ライブラリーを使用するためには適切な Web ブラウザーと Adobe Acrobat Reader が必要です (また、これらを構成する必要もあります)。

オンライン・ライブラリーを開くには、**Documentation** アイコンを選択すると、ライブラリーのホーム・ページが表示されます。

オンライン・ライブラリーを使用すると、以下にリンクできます。

- PDF 形式の CICS Transaction Gateway および CICS ユニバーサル・クライアント資料。
- ハイパーテキスト・マークアップ言語 (HTML) ファイル形式のプログラミング解説書資料 (CICS Transaction Gateway 用にのみ提供)。
- README ファイル。
- PDF 形式のサンプル構成資料。
- CICS Web サイト。

Acrobat Reader の使用についての情報も提供されています。

資料の関連バージョンが提供されることもあります。以下の弊社の Web サイトをチェックしてください。

`www.ibm.com/software/ts/cics/`

Library のリンクに従ってください。

注: 一部の資料は、他の言語に翻訳されています。それらはオンライン・ライブラリーには含まれていませんが、独立した PDF ファイルとして、上記の Web サイトから入手することができます。

PDF 資料の表示

PDF 情報は以下を行うため強力な機能を提供します。

- 情報のナビゲート。 PDF 資料内、および他の PDF 資料および Web ページにはハイパーテキスト・リンクがあります。
- 特定情報の検索。
- PostScript プリンターでの PDF 資料の全印刷または部分印刷。

Adobe Web サイトで Acrobat Reader についてさらに知ることができます。

`www.adobe.com/acrobat/`

特記事項

本書はアメリカ合衆国で提供されている製品およびサービス用に作成されたものであり、本書に記載の製品、サービス、またはフィーチャーが日本においては提供されていない場合があります。日本で利用可能な製品、サービス、およびフィーチャーについては、日本 IBM の営業担当員にお尋ねください。本書で IBM 製品、プログラム、またはサービスに言及していても、その IBM 製品、プログラム、またはサービスのみが使用可能であることを意味するものではありません。これらに代えて、IBM の知的所有権を侵害することのない機能的に同等のプログラムまたは製品を使用することができます。ただし、IBM 以外の製品、プログラムまたはサービスの操作性の評価および検証は、お客様の責任で行っていただきます。

IBM は、本書に記載されている内容に関して特許権 (特許出願中のものを含む。) を保有している場合があります。本書の提供は、お客様にこれらの特許権について実施権を許諾することを意味するものではありません。実施権の許諾については、下記の宛先に書面にてご照会ください。

〒106-0032 東京都港区六本木 3 丁目 2-31
AP事業所
IBM World Trade Asia Corporation
Intellectual Property Law & Licensing

以下の保証は、国または地域の法律に沿わない場合は、適用されません。

IBM およびその直接または間接の子会社は、本書を特定物として現存するままの状態で提供し、商品性の保証、特定目的適合性の保証および法律上の瑕疵担保責任を含むすべての明示もしくは黙示の保証責任を負わないものとします。国または地域によっては、法律の強行規定により、保証責任の制限が禁じられる場合、強行規定の制限を受けるものとします。

本書は定期的に見直され、必要な変更 (たとえば、技術的に不適確な表現や誤植など) は、本書の次版に組み込まれます。IBM は、随時、この文書に記載されている製品またはプログラムに対して、改良または変更を行うことがあります。

本書において IBM 以外の Web サイトに言及している場合がありますが、便宜のため記載しただけであり、決してそれらの Web サイトを推奨するもので

はありません。それらの Web サイトにある資料は、この IBM 製品の資料の一部ではありません。それらの Web サイトは、お客様の責任でご使用ください。

本プログラムのライセンス保持者で、(i) 独自に作成したプログラムとその他のプログラム (本プログラムを含む) との間での情報交換、および (ii) 交換された情報の相互利用を可能にすることを目的として、本プログラムに関する情報を必要とする方は、下記に連絡してください。

IBM United Kingdom Laboratories, MP151,
Hursley Park, Winchester, Hampshire,
England, SO21 2JN

本プログラムに関する上記の情報は、適切な使用条件の下で使用することができますが、有償の場合もあります。

本書で説明されているライセンス・プログラムまたはその他のライセンス資料は、IBM 所定のプログラム契約の契約条項、IBM プログラムのご使用条件、またはそれと同等の条項に基づいて、IBM より提供されます。

IBM 以外の製品に関する情報は、その製品の供給者、出版物、もしくはその他の公に利用可能なソースから入手したものです。IBM は、それらの製品のテストは行っておりません。また、IBM 以外の製品に関するパフォーマンスの正確性、互換性、またはその他の要求は確証できません。IBM 以外の製品の性能に関する質問は、それらの製品の供給者にお願いします。

著作権使用許諾:

本書には、様々なオペレーティング・プラットフォームでのプログラミング手法を例示するサンプル・アプリケーション・プログラムがソース言語で掲載されています。お客様は、IBM のアプリケーション・プログラミング・インターフェースに準拠したアプリケーション・プログラムの開発、使用、販売、配布を目的として、いかなる形式においても、IBM に対価を支払うことなくこれを複製し、改変し、配布することができます。このサンプル・プログラムは、あらゆる条件下における完全なテストを経ていません。従って IBM は、これらのサンプル・プログラムについて信頼性、利便性もしくは機能性があることをほのめかしたり、保証することはできません。

商標

以下は、IBM Corporation の商標です。

AIX	IBM	VisualAge
CICS	OS/2	

Microsoft、Windows、Windows NT、および Windows ロゴは、Microsoft Corporation の米国およびその他の国における商標です。

UNIX は、The Open Group がライセンスしている米国およびその他の国における登録商標です。

索引

日本語, 数字, 英字, 特殊文字の順に配列されています。なお, 濁音と半濁音は清音と同等に扱われています。

[ア行]

[カ行]

[サ行]

[タ行]

[ハ行]

[マ行]

[ラ行]

[数字]

A

B

C

D

E

F

G

H

I

P

Q

R

S

T

U

V

W

X

Y

[特殊文字]

プログラム番号: 5724-A75

Printed in Japan

SC88-8996-00

日本アイ・ビー・エム株式会社
〒106-8711　東京都港区六本木3-2-12