

CICS Transaction Server for z/
OSバージョン 5 リリース 6

C++ OO クラス・ライブラリー



注記

本書および本書で紹介する製品をご使用になる前に、[製品の特記事項](#)に記載されている情報をお読みください。

本書は、IBM® CICS® Transaction Server for z/OS®, バージョン 5 リリース 6 (製品番号 5655-Y305655-BTA)、および新しい版で明記されていない限り、以降のすべてのリリースおよびモディフィケーションに適用されます。

お客様の環境によっては、資料中の円記号がバックスラッシュと表示されたり、バックスラッシュが円記号と表示されたりする場合があります。

原典：

CICS Transaction Server for z/OS
Version 5 Release 5
C++ OO Class Libraries

発行：

日本アイ・ビー・エム株式会社

担当：

トランスレーション・サービス・センター

© Copyright International Business Machines Corporation 1974, 2020.

目次

本書について.....	ix
第 1 章インストールとセットアップ.....	1
オブジェクト指向 CICS の準備.....	1
インストール済みの内容.....	1
ヘッダー・ファイル.....	1
ダイナミック・リンク・ライブラリー.....	2
サンプル・ソース・コード.....	2
CICS Transaction Server for z/OS のその他のデータ・セット.....	3
第 2 章 CICS ファウンデーション・クラスの使用.....	5
C++ オブジェクト.....	5
オブジェクトの作成.....	5
オブジェクトを使用.....	6
オブジェクトの削除.....	6
ファウンデーション・クラスの概要.....	6
基本クラス.....	6
リソース識別クラス.....	7
リソース・クラス.....	9
サポート・クラス.....	10
CICS リソースの使用.....	11
バッファ・オブジェクト.....	12
IccBuf クラス.....	13
CICS Service の使用.....	15
ファイル制御.....	15
プログラム制御.....	19
トランザクションの非同期の開始.....	21
一時データ.....	24
一時記憶.....	25
端末管理.....	27
日時サービス.....	29
コンパイル、実行、およびデバッグ.....	30
CICS ファウンデーション・クラス・プログラムのコンパイル.....	30
プログラムの実行.....	32
プログラムのデバッグ.....	32
条件、エラー、および例外.....	33
ファウンデーション・クラスの異常終了コード.....	33
C++ 例外およびファウンデーション・クラス.....	33
CICS 条件.....	35
プラットフォームの相違.....	37
ポリモフィック動作.....	39
ポリモフィック動作の例.....	42
ストレージ管理.....	43
パラメーター受け渡し規則.....	44
「read」メソッドから返される IccBuf 参照内のデータの有効範囲.....	44
第 3 章ファウンデーション・クラス: 参照.....	45
EXEC CICS 呼び出しとファウンデーション・クラス・メソッドのマッピング.....	46
EXEC CICS 呼び出しへのファウンデーション・クラス・メソッドのマッピング.....	51
Icc 構造.....	58

関数.....	58
列挙.....	60
IccAbendData クラス.....	62
IccAbendData コンストラクター (protected).....	62
public メソッド.....	62
継承された public メソッド.....	66
継承された protected メソッド.....	66
IccAbsTime クラス.....	66
IccAbsTime コンストラクター.....	67
public メソッド.....	67
継承された public メソッド.....	70
継承された protected メソッド.....	71
IccAlarmRequestId クラス.....	71
IccAlarmRequestId コンストラクター.....	71
public メソッド.....	72
継承された public メソッド.....	73
継承された protected メソッド.....	73
IccBase クラス.....	73
IccBase コンストラクター (protected).....	73
public メソッド.....	74
protected メソッド.....	75
列挙.....	75
IccBuf クラス.....	77
IccBuf コンストラクター.....	77
public メソッド.....	78
継承された public メソッド.....	86
継承された protected メソッド.....	86
列挙.....	86
IccClock クラス.....	87
IccClock コンストラクター.....	87
public メソッド.....	87
継承された public メソッド.....	90
継承された protected メソッド.....	90
列挙.....	90
IccCondition 構造体.....	92
列挙.....	92
IccConsole クラス.....	93
IccConsole コンストラクター (protected).....	93
public メソッド.....	93
継承された public メソッド.....	96
継承された protected メソッド.....	96
列挙.....	96
IccControl クラス.....	97
IccControl コンストラクター (protected).....	97
public メソッド.....	97
継承された public メソッド.....	100
継承された protected メソッド.....	101
IccConvId クラス.....	101
IccConvId コンストラクター.....	101
public メソッド.....	102
継承された public メソッド.....	102
継承された protected メソッド.....	102
IccDataQueue クラス.....	103
IccDataQueue コンストラクター.....	103
public メソッド.....	103
継承された public メソッド.....	105
継承された protected メソッド.....	105
IccDataQueueId クラス.....	106

IccDataQueueId コンストラクター	106
public メソッド	106
継承された public メソッド	107
継承された protected メソッド	107
IccEvent クラス	107
IccEvent コンストラクター	107
public メソッド	108
継承された public メソッド	109
継承された protected メソッド	109
IccException クラス	109
IccException コンストラクター	109
public メソッド	110
継承された public メソッド	111
継承された protected メソッド	111
列挙	112
IccFile クラス	112
IccFile コンストラクター	112
public メソッド	113
継承された public メソッド	122
継承された protected メソッド	123
列挙	123
IccFileId クラス	124
IccFileId コンストラクター	124
public メソッド	125
継承された public メソッド	125
継承された protected メソッド	125
IccFileIterator クラス	126
IccFileIterator コンストラクター	126
public メソッド	126
継承された public メソッド	127
継承された protected メソッド	128
IccGroupId クラス	128
IccGroupId コンストラクター	128
public メソッド	129
継承された public メソッド	129
継承された protected メソッド	130
IccJournal クラス	130
IccJournal コンストラクター	130
public メソッド	131
継承された public メソッド	133
継承された protected メソッド	134
列挙	134
IccJournalId クラス	134
IccJournalId コンストラクター	134
public メソッド	135
継承された public メソッド	136
継承された protected メソッド	136
IccJournalTypeId クラス	136
IccJournalTypeId コンストラクター	136
public メソッド	137
継承された public メソッド	137
継承された protected メソッド	137
IccKey クラス	138
IccKey コンストラクター	138
public メソッド	138
継承された public メソッド	141
継承された protected メソッド	141
列挙	141

IccLockId クラス.....	141
IccLockId コンストラクター	141
public メソッド.....	142
継承された public メソッド.....	142
継承された protected メソッド.....	143
IccMessage クラス.....	143
IccMessage コンストラクター	143
public メソッド.....	143
継承された public メソッド.....	144
継承された protected メソッド.....	144
IccPartnerId クラス.....	145
IccPartnerId コンストラクター	145
public メソッド.....	145
継承された public メソッド.....	146
継承された protected メソッド.....	146
IccProgram クラス.....	146
IccProgram コンストラクター	146
public メソッド.....	147
継承された public メソッド.....	149
継承された protected メソッド.....	150
列挙.....	150
IccProgramId クラス.....	150
IccProgramId コンストラクター	150
public メソッド.....	151
継承された public メソッド.....	151
継承された protected メソッド.....	151
IccRBA クラス.....	152
IccRBA コンストラクター	152
public メソッド.....	152
継承された public メソッド.....	153
継承された protected メソッド.....	153
IccRecordIndex クラス.....	154
IccRecordIndex コンストラクター (protected).....	154
public メソッド.....	154
継承された public メソッド.....	155
継承された protected メソッド.....	155
列挙.....	155
IccRequestId クラス.....	155
IccRequestId コンストラクター	155
public メソッド.....	156
継承された public メソッド.....	156
継承された protected メソッド.....	157
IccResource クラス.....	157
IccResource コンストラクター (protected).....	157
public メソッド.....	157
継承された public メソッド.....	162
継承された protected メソッド.....	162
列挙.....	162
IccResourceId クラス.....	163
IccResourceId コンストラクター (protected).....	163
public メソッド.....	164
protected メソッド.....	164
継承された public メソッド.....	164
継承された protected メソッド.....	165
IccRRN クラス.....	165
IccRRN コンストラクター	165
public メソッド.....	165
継承された public メソッド.....	166

継承された protected メソッド.....	167
IccSemaphore クラス.....	167
IccSemaphore コンストラクター.....	167
public メソッド.....	168
継承された public メソッド.....	169
継承された protected メソッド.....	169
列挙.....	169
IccSession クラス.....	170
IccSession コンストラクター (public).....	170
IccSession コンストラクター (protected).....	171
public メソッド.....	171
継承された public メソッド.....	179
継承された protected メソッド.....	179
列挙.....	179
IccStartRequestQ クラス.....	180
IccStartRequestQ コンストラクター (protected).....	180
public メソッド.....	181
継承された public メソッド.....	185
継承された protected メソッド.....	186
列挙.....	186
IccSysId クラス.....	186
IccSysId コンストラクター.....	186
public メソッド.....	187
継承された public メソッド.....	187
継承された protected メソッド.....	188
IccSystem クラス.....	188
IccSystem コンストラクター (protected).....	188
public メソッド.....	188
継承された public メソッド.....	192
継承された protected メソッド.....	193
列挙.....	193
IccTask クラス.....	193
IccTask コンストラクター (protected).....	194
public メソッド.....	194
継承された public メソッド.....	201
継承された protected メソッド.....	202
列挙.....	202
IccTempStore クラス.....	204
IccTempStore コンストラクター.....	204
public メソッド.....	205
継承された public メソッド.....	207
継承された protected メソッド.....	208
列挙.....	208
IccTempStoreId クラス.....	208
IccTempStoreId コンストラクター.....	208
public メソッド.....	209
継承された public メソッド.....	209
継承された protected メソッド.....	210
IccTermId クラス.....	210
IccTermId コンストラクター.....	210
public メソッド.....	210
継承された public メソッド.....	211
継承された protected メソッド.....	211
IccTerminal クラス.....	211
IccTerminal コンストラクター (protected).....	212
public メソッド.....	212
継承された public メソッド.....	225
継承された protected メソッド.....	225

列挙.....	225
IccTerminalData クラス.....	226
IccTerminalData コンストラクター (protected).....	227
public メソッド.....	227
継承された public メソッド.....	232
継承された protected メソッド.....	232
IccTime クラス.....	233
IccTime コンストラクター (protected).....	233
public メソッド.....	233
継承された public メソッド.....	234
継承された protected メソッド.....	235
列挙.....	235
IccTimeInterval クラス.....	235
IccTimeInterval コンストラクター.....	235
public メソッド.....	236
継承された public メソッド.....	236
継承された protected メソッド.....	237
IccTimeOfDay クラス.....	237
IccTimeOfDay コンストラクター.....	237
public メソッド.....	238
継承された public メソッド.....	238
継承された protected メソッド.....	239
IccTPNameId クラス.....	239
IccTPNameId コンストラクター.....	239
public メソッド.....	240
継承された public メソッド.....	240
継承された protected メソッド.....	241
IccTransId クラス.....	241
IccTransId コンストラクター.....	241
public メソッド.....	241
継承された public メソッド.....	242
継承された protected メソッド.....	242
IccUser クラス.....	242
IccUser コンストラクター.....	242
public メソッド.....	243
継承された public メソッド.....	245
継承された protected メソッド.....	246
IccUserId クラス.....	246
IccUserId コンストラクター.....	246
public メソッド.....	246
継承された public メソッド.....	247
継承された protected メソッド.....	247
IccValue 構造.....	247
列挙型.....	247
main 関数.....	248
特記事項.....	251
索引.....	257

本書について

本書では、アプリケーション・プログラマーが EXEC CICS API を使用して CICS サービスにアクセスすることを可能にする CICS C++ ファウンデーション・クラスの使用方法について説明しています。

使用されている用語や表記について詳しくは、IBM Knowledge Center の [CICS 資料で使用されている表記規則および用語](#)を参照してください。

本書の日付

本書は、2020 年 5 月 28 日に作成されました。

第1章 インストールとセットアップ

このセクションでは、ご使用の CICS サーバーにインストールされている CICS ファウンデーション・クラスについて説明します。

オブジェクト指向 CICS の準備

この後のトピックについて理解するには、オブジェクト指向の概念とテクノロジー、C++ 言語、および CICS についてよく理解している必要があります。

ここでは、これらのテーマについての概要は説明していません。

インストール済みの内容

CICS ファウンデーション・クラス・パッケージは、複数のファイルやデータ・セットから成ります。

CICS ファウンデーション・クラス・パッケージは、複数のファイルやデータ・セットから成ります。これらは以下を含みます。

- ヘッダー・ファイル
- 実行可能ファイル (DLL のもの)
- サンプル
- その他の CICS Transaction Server for z/OS ファイル

このセクションでは、CICS C++ ファウンデーション・クラスを構成するファイルについてと、それらが CICS サーバー上のどこにあるかについて説明しています。

ヘッダー・ファイル

ヘッダー・ファイルは、CICS C++ ファウンデーション・クラス・プログラムをコンパイルするために必要な C++ クラス定義です。

C++ ヘッダー・ファイル	このヘッダーに定義されているクラス
ICCABDEH	IccAbendData
ICCBASEH	IccBase
ICCBUFEH	IccBuf
ICCCLKEH	IccClock
ICCCNDEH	IccCondition (struct)
ICCCONEH	IccConsole
ICCCTLEH	IccControl
ICCDATEH	IccDataQueue
ICCEH	2 ページの『1』 を参照してください。
ICCEVTEH	IccEvent
ICCEXCEH	IccException
ICCFILEH	IccFile
ICCFLIEH	IccFileIterator
ICCGLBEH	Icc (struct) (グローバル関数)
ICCJRNEH	IccJournal
ICCMSGEH	IccMessage

C++ ヘッダー・ファイル	このヘッダーに定義されているクラス
ICCPRGEH	IccProgram
ICCRECEH	IccRecordIndex、IccKey、IccRBA、および IccRRN
ICCRESEH	IccResource
ICCRIDEH	IccResourceId + サブクラス (IccConvId など)
ICCSEMEH	IccSemaphore
ICCSESEH	IccSession
ICCSRQEH	IccStartRequestQ
ICCSYSEH	IccSystem
ICCTIMEH	IccTime、IccAbsTime、IccTimeInterval、IccTimeOfDay
ICCTMDEH	IccTerminalData
ICCTMPEH	IccTempStore
ICCTRMEH	IccTerminal
ICCTSKEH	IccTask
ICCUSREH	IccUser
ICCVALEH	IccValue (struct)

注:

1. リストされているすべてのヘッダー・ファイルを含む単一のヘッダーは、ICCEH という名前です。
2. ICCMAIN ファイルには、C++ ヘッダー・ファイルも提供されます。これには、ファウンデーション・クラス・プログラムの作成時に使用する **main** 関数スタブが含まれています。
3. ヘッダー・ファイルは、CICSTS56.CICS.SDFHC370 にあります。

場所

PDS: CICSTS56.CICS.SDFHC370。

ダイナミック・リンク・ライブラリー

ダイナミック・リンク・ライブラリーは、CICS C++ ファウンデーション・クラス・プログラムをサポートするために必要なランタイム環境です。

場所

PDS 内の ICCFCDLL モジュール: CICSTS56.CICS.SDFHLOAD。

サンプル・ソース・コード

クラスを使用してオブジェクト指向アプリケーションを作成する方法を理解できるように、サンプルが用意されています。

場所

PDS: CICSTS56.CICS.SDFHSAMP。

サンプル・アプリケーションの実行

メンバー DFHCURDS で定義されたリソースがインストール済みであれば、いくつかのサンプル・アプリケーションをすぐに実行できるはずです。

サンプル・プログラムは、CICSTS56.CICS.SDFHSAMP ライブラリーにソース・コードとして提供されています。サンプル・プログラムを実行する前に、コンパイル、プリリンク、およびリンクする必要があります。

す。これを行うには、データ・セット CICSTS56.CICS.SDFHPROC の ICCFCCL プロシージャを使用します。

ICCFCLL には、CICS ユーザー・アプリケーションをコンパイル、プリリンク、およびリンクするのに必要なジョブ制御言語が含まれています。ICCFCLL を使用する前に、ご使用のシステムの規格に合わせていくらかカスタマイズする必要があることに気付くかもしれません。[プログラムのコンパイル](#)も参照してください。

ICC\$BUF、ICC\$CLK および ICC\$HEL などのサンプル・プログラムには、追加の CICS リソース定義は不要で、すぐに正常に実行できます。

他のサンプル・プログラム (特に DTP サンプル・プログラム ICC\$SES1 および ICC\$SES2) は、追加の CICS リソース定義が必要です。これらの追加の要件については、サンプル・プログラムのソースの前書き部分を参照してください。

CICS Transaction Server for z/OS のその他のデータ・セット

CICSTS56.CICS.SDFHSDCK には、以下のメンバーが含まれます

- ICCFCIMP - インポート制御ステートメントが含まれる 'sidedeck'

CICSTS56.CICS.SDFHPROC には、以下のメンバーが含まれます

- ICCFCC - CFC ユーザー・プログラムをコンパイルするための JCL
- ICCFCCL - CFC ユーザー・プログラムをコンパイル、プリリンク、およびリンクするための JCL
- ICCFCGL - CFC ライブラリーを使用する XPLINK プログラムをコンパイルおよびリンクするための JCL。
- ICCFCL - CFC ユーザー・プログラムを プリリンクおよびリンクするための JCL

CICSTS56.CICS.SDFHLOAD には、以下のメンバーが含まれます

- DFHCURDS - CICS システム定義に必要なプログラム定義。
- DFHCURDI - CICS システム定義に必要なプログラム定義。

第 2 章 CICS ファウンデーション・クラスの使用

このセクションでは、CICS ファウンデーション・クラスについて、およびそれらの使用方法について説明します。ユーザー・インターフェースの公式なリストについては、[ファウンデーション・クラス: リファレンス](#)を参照してください。

C++ オブジェクト

このセクションでは、オブジェクトを作成、使用、および削除する方法について説明します。

このセクションでは、オブジェクトを作成、使用、および削除する方法について説明します。このコンテキストでは、オブジェクトはクラスのインスタンスです。オブジェクトを、基本クラスまたは抽象基本クラスのインスタンスにすることはできません。本書のリファレンス部分で説明しているすべての具象 (非基本) クラスのオブジェクトを作成できます。

オブジェクトの作成

クラスにコンストラクターがある場合は、そのクラスのオブジェクトの作成時にコンストラクターが実行されます。このコンストラクターは一般的に、オブジェクトの状態を初期化します。ファウンデーション・クラスのコンストラクターには多くの場合、プログラマーがオブジェクト作成時に指定する必要がある必須の位置パラメーターがあります。

C++ オブジェクトは、以下のいずれかの方法で作成できます。

1. オブジェクトが C++ スタック上に作成される場合は、自動。以下に例を示します。

```
{
ClassX objX
ClassY objY(parameter1);
} //objects deleted here
```

ここで、objX および objY が自動的にスタック上に作成されます。オブジェクトの存続期間は、オブジェクトが作成されたコンテキストによって制限されます。オブジェクトがその有効期間を超えると、自動的に削除されます (つまり、デストラクターが実行され、ストレージが解放されます)。

2. オブジェクトが C++ ヒープ上に作成される場合は、動的。以下に例を示します。

```
{
ClassX* pObjX = new ClassX;
ClassY* pObjY = new ClassY(parameter1);
} //objects NOT deleted here
```

ここで、オブジェクトそのものではなく、オブジェクトのポインターを操作します。オブジェクトの存続期間は、それが作成されたときの有効期間を超えます。前のサンプルでは、オブジェクトが有効期間を超えると、ポインター (pObjX および pObjY) は「失われます」が、ポインターが指していたオブジェクトは存在し続けます。オブジェクトは、以下のように明示的に削除しない限り、存在しています。

```
{
ClassX* pObjX = new ClassX;
ClassY* pObjY = new ClassY(parameter1);
:
pObjX->method1();
pObjY->method2();
:
delete pObjX;
delete pObjY;
}
```

本書のほとんどのサンプルでは、自動ストレージが使用されています。オブジェクトを明示的に削除することを覚えている必要がないため、自動ストレージを使用することが **推奨** されますが、CICS C++ ファウンデーション・クラス・プログラムにはどちらのスタイルを使用しても構いません。ファウンデーション・クラスおよびストレージ管理の詳細については、『43 ページの『ストレージ管理』』を参照してください。

オブジェクトを使用

クラスの `public` メソッドは、そのクラスのオブジェクトに対して呼び出すことができます。

クラスの `public` メソッドは、そのクラスのオブジェクトに対して呼び出すことができます。次の例では、オブジェクト `obj` を作成してから、そのオブジェクトに対してメソッド `doSomething` を呼び出します。

```
ClassY obj("TEMP1234");  
obj.doSomething();
```

また、動的オブジェクト作成の使用時に、これを実行することもできます。

```
ClassY* pObj = new ClassY("parameter1");  
pObj->doSomething();
```

オブジェクトの削除

オブジェクトが破棄されたら、そのデストラクター関数 (~ (チルド) が先頭に付いたクラスと同じ名前を持つ) が自動的に呼び出されます。(デストラクターを明示的に呼び出すことはできません。)

オブジェクトが自動的に作成された場合は、有効範囲を超えたときに、自動的に破棄されます。

オブジェクトが動的に作成された場合は、明示的な `delete` 演算子を使用されるまで、存在し続けます。

ファウンデーション・クラスの概要

このトピックでは、ファウンデーション・クラスで実行できる内容について概説します。

単純な開始方法については、『[ICC\\$HEL: C++ Hello World サンプル](#)』を参照してください。このセクションでは、CICS C++ ファウンデーション・クラス・ライブラリーについて概説するため、カテゴリーを順に示していきます。

ファウンデーション・クラスの詳細については、『[ファウンデーション・クラス・リファレンス](#)』を参照してください。

CICS ファウンデーション・クラスに属するすべてのクラスに接頭部 **Icc** が付きます。

基本クラス

すべてのクラスは、**IccBase** から直接または間接的に継承します。

```
IccBase  
IccRecordIndex  
IccResource  
IccControl  
IccTime  
IccResourceId
```

図 1. 基本クラス

すべてのリソース識別クラス (**IccTermId**、**IccTransId** など) が、**IccResourceId** クラスから継承します。これらは、通常、CICS テーブル・エントリーです。

すべての CICS リソース (実際には、**IccResource** から継承し、CICS サービスへのアクセスを必要とするすべてのクラス)。

基本クラスにより、クラスのカテゴリーの共通インターフェースを定義できます。これらのインターフェースは、IBM 提供のファウンデーション・クラスを作成するために使用され、アプリケーション・プログラマーが独自で派生させたクラスを作成するために使用できます。

IccBase

他のすべてのファウンデーション・クラスの基本。これにより、メモリー管理が可能になり、オブジェクトのタイプを検出するためにオブジェクトを調べることができます。

IccControl

アプリケーション・プログラムがサブクラスを設定し、**run** メソッドの実装を提供する必要がある、抽象基本クラス。

IccResource

CICS リソースまたはサービスにアクセスするすべてのクラスの基本クラス。[9 ページの『リソース・クラス』](#)を参照してください。

IccResourceId

IccFileId および **IccTempStoreId** などの、すべてのテーブル・エントリー (リソース名) クラスの基本クラス。

IccTime

時間情報を保管するクラス (**IccAbsTime**、**IccTimeInterval**、および **IccTimeOfDay**) の基本クラス。

リソース識別クラス

リソース識別クラスは以下のとおりです。

```
IccBase
  IccResourceId
  IccConvId
  IccDataQueueId
  IccFileId
  IccGroupId
  IccJournalId
  IccJournalTypeId
  IccLockId
  IccPartnerId
  IccProgramId
  IccRequestId
  IccAlarmRequestId
  IccSysId
  IccTempStoreId
  IccTermId
  IccTPNameId
  IccTransId
  IccUserId
```

図 2. リソース識別クラス

CICS リソース識別クラスは、CICS リソース ID (一般的には、RDO リソース定義に指定されているリソースの名前) を定義します。例えば、**IccFileId** オブジェクトは CICS ファイル名を表します。すべての具象リソース識別クラスに、以下のプロパティーが適用されます。

- クラスの名前は **Id** で終わります。
- クラスは、**IccResourceId** クラスのサブクラスです。
- コンストラクターは、指定されたリソース ID が CICS 標準に従っているかどうかを検査します。例えば、**IccFileId** オブジェクトには、1 から 8 バイトの文字フィールドが含まれている必要があります。9 バイトのフィールドを指定しても、許可されません。

リソース識別クラスは、型検査を改善します。パラメーターとして **IccFileId** オブジェクトを予期するメソッドは、代わりに **IccProgramId** オブジェクトが提供されると、それを受け入れません。リソース名を表す文字ストリングが代わりに使用されると、コンパイラーが妥当性を検査できません。つまり、ストリングがファイル名であるか、プログラム名であるかを検査できません。

『9 ページの『リソース・クラス』』で説明しているリソース・クラスの多くには、リソース識別クラスが含まれています。例えば、**IccFile** オブジェクトには **IccFileId** オブジェクトが含まれています。CICS リソース上で機能させるためには、リソース識別オブジェクトではなく、リソース・オブジェクトを使用する必要があります。例えば、ファイルからレコードを読み取る場合は、**IccFileId** ではなく **IccFile** を使用する必要があります。

クラス	CICS リソース
IccAlarmRequestId	アラーム要求
IccConvId	会話 (conversation)
IccDataQueueId	一時データ・キュー
IccFileId	ファイル
IccGroupId	グループ
IccJournalId	ジャーナル
IccJournalTypeId	ジャーナル・タイプ
IccLockId	(適用されない)
IccPartnerId	APPC パートナー定義ファイル
IccProgramId	プログラム
IccRequestId	要求
IccSysId	リモート・システム
IccTempStoreId	一時記憶域キュー
IccTermId	端末
IccTPNameId	リモート APPC TP 名
IccTransId	トランザクション
IccUserId	ユーザー

リソース・クラス

CICS リソース・クラスはすべて、**IccResource** 基本クラスから継承します。

IccBase
IccResource
IccAbendData
IccClock
IccConsole
IccControl
IccDataQueue
IccFile
IccFileIterator
IccJournal
IccProgram
IccSemaphore
IccSession
IccStartRequestQ
IccSystem
IccTask
IccTempStore
IccTerminal
IccTerminalData
IccUser

図 3. リソース・クラス

これらのクラスは、以下のように、主要な CICS リソースの動作をモデル化します。

- 端末は **IccTerminal** によってモデル化されます。
- プログラムは **IccProgram** によってモデル化されます。
- 一時記憶域キューは **IccTempStore** によってモデル化されます。
- 一時データ・キューは **IccDataQueue** によってモデル化されます。

CICS リソースに対する操作によって、CICS 条件が発生することがあります。**IccResource** の **condition** メソッド ([『IccResource method: condition』](#) ページを参照) がリソースの問い合わせを行うことがあります。

(CICS サービスにアクセスするクラスは、すべて、**IccResource** から派生したものである必要があります)。

クラス	CICS リソース
IccAbendData	タスク異常終了データ
IccClock	CICS 日時サービス
IccConsole	CICS コンソール
IccControl	実行中のプログラムの制御
IccDataQueue	一時データ・キュー
IccFile	ファイル
IccFileIterator	ファイル・イテレーター (ファイルの参照)
IccJournal	ユーザーまたはシステム・ジャーナル
IccProgram	プログラム (実行中のプログラムの外部)
IccSemaphore	セマフォ (サービスのロック)
IccSession	セッション

クラス	CICS リソース
IccStartRequestQ	開始要求キュー (非同期トランザクション開始)
IccSystem	CICS システム
IccTask	現行タスク
IccTempStore	一時記憶域キュー
IccTerminal	現行タスクに属する端末
IccTerminalData	IccTerminal の属性
IccTime	時間指定
IccUser	ユーザー (セキュリティ属性)

サポート・クラス

サポート・クラスは以下のとおりです。

IccBase
IccBuf
IccEvent
IccException
IccMessage
IccRecordIndex
IccKey
IccRBA
IccRRN
IccResource
IccTime
IccAbsTime
IccTimeInterval
IccTimeOfDay

図 4. サポート・クラス

これらのクラスは、リソース・クラスを補完するツールです。これによりアプリケーション・プログラマーの負担が軽減され、オブジェクト・モデルに値が追加されます。

リソース・クラス	説明
IccAbsTime	絶対時刻 (1900 年 1 月 1 日以降のミリ秒)
IccBuf	データ・バッファー (データ域の操作を容易にする)
IccEvent	イベント (CICS コマンドの結果)
IccException	ファウンデーション・クラス例外 (C++ 例外処理モデルをサポートする)
IccTimeInterval	時間間隔 (例: 5 分)
IccTimeOfDay	時刻 (例: 6 時 5 分)

IccAbsTime、**IccTimeInterval**、および **IccTimeOfDay** クラスにより、アプリケーション・プログラマーが、アプリケーション・プログラム内のオブジェクトとして時間測定を簡単に指定できるようになります。**IccTime** は基本クラスです。**IccAbsTime**、**IccTimeInterval**、および **IccTimeOfDay** は **IccTime** から派生します。

以下のようなシグニチャーを持つ **IccTask** クラスのメソッド **delay** について考えてみます。

```
void delay(const IccTime& time, const IccRequestId*  
reqId = 0);
```

1 分 7 秒の遅延 (つまり、時間間隔) を要求するには、アプリケーション・プログラマーは以下を実行します。

```
IccTimeInterval time(0, 1, 7);  
task()->delay(time);
```

注 : IccControl クラスにはタスク・メソッドが用意されており、アプリケーションのタスク・オブジェクトのポインターを返します。

また、12 時 10 分 (ランチタイム?) までの遅延を要求するには、アプリケーション・プログラマーが以下を実行します。

```
IccTimeOfDay lunchtime(12, 10);  
task()->delay(lunchtime);
```

IccBuf クラスにより、バッファ (ファイル・レコード・バッファ、一時データ・レコード・バッファ、COMMAREA など) を容易に操作できます (**IccBuf** クラスの詳細については、『[12 ページの『バッファ・オブジェクト』](#)』を参照してください)。

IccMessage クラスは主に **IccException** クラスによって使用され、例外がスローされた理由を示す説明をカプセル化します。また、アプリケーション・プログラマーは **IccMessage** を使用すると、独自のメッセージ・オブジェクトを作成することもできます。

エラーが検出されると、ファウンデーション・クラス内の多くのメソッドから **IccException** オブジェクトがスローされます。

IccEvent クラスを使用すると、プログラマーは、特定の CICS イベント (コマンド) に関連する情報にアクセスできます。

CICS リソースの使用

CICS リソース (ファイル、プログラムなど) を使用するには、まず適切なオブジェクトを作成してから、オブジェクトに対するメソッドを呼び出す必要があります。

リソース・オブジェクトの作成

リソース・オブジェクトを作成する場合は、実際の CICS リソース (ファイルやプログラムなど) の表現を作成します。CICS リソースを作成するわけではありません。このオブジェクトは、アプリケーションから見たリソースです。オブジェクトの破棄にも同じことが当てはまります。

リソース・オブジェクトを作成するときは、付随するリソース識別オブジェクトを使用します。以下に例を示します。

```
IccFileId id("XYZ123");  
IccFile file(id);
```

これにより、以下のような誤った処理が実行されないように C++ コンパイラーで防止できるようになります。

```
IccDataQueueId id("WXYZ");  
IccFile file(id); //gives error at compile time
```

オブジェクト作成時に、リソースのテキスト名を代わりに使用することも可能です。

```
IccFile file("XYZ123");
```

singleton クラス

IccFile など、多くのリソース・クラスは、単一プログラム内に複数のリソース・オブジェクトを作成するために使用できます。

```
IccFileId id1("File1");
IccFileId id2("File2");
IccFile file1(id1);
IccFile file2(id2);
```

しかし、一部のリソース・クラスは、プログラマーがそのクラスのインスタンスを **1つ** だけ作成できるように設計されており、それらのクラスは singleton クラスと呼ばれています。以下のファウンデーション・クラスは singleton です。

- **IccAbendData** は、タスクの異常終了に関する情報を提供します。
- **IccConsole** またはその派生クラスは、オペレーター・メッセージ用のシステム・コンソールを表します。
- **IccControl** またはその派生クラス (**IccUserControl** など) は、実行中のプログラムを制御します。
- **IccStartRequestQ** またはその派生クラスは、アプリケーション・プログラムが CICS トランザクション (タスク) を非同期に開始できるようにします。
- **IccSystem** またはその派生クラスは、そのアプリケーションを実行している CICS システムのアプリケーション・ビューです。
- **IccTask** またはその派生クラスは、実行中のプログラムを実行している CICS タスクを表します。
- **IccTerminal** またはその派生クラスは、使用している基本機能が 3270 端末であれば、タスクの端末を表します。

singleton クラスの複数のオブジェクトを作成しようとすると、エラーになり、C++ 例外がスローされます。

これらの各 singleton クラスには、**instance** というクラス・メソッドが用意されています。これは、要求されたオブジェクトへのポインターを返し、まだそのポインターが存在しない場合は作成します。以下に例を示します。

```
IccControl* pControl = IccControl::instance();
```

リソース・オブジェクトでのメソッドの呼び出し

public メソッドはいずれも、リソース・クラスのオブジェクトで呼び出すことができます。

以下に例を示します。

```
IccTempStoreId id("TEMP1234");
IccTempStore temp(id);
temp.writeItem("Hello TEMP1234");
```

メソッド **writeItem** は、渡されたストリングの内容 ("Hello TEMP1234") を CICS 一時記憶域キュー「TEMP1234」に書き込みます。

バッファ・オブジェクト

ファウンデーション・クラスは、**IccBuf** オブジェクトを広範囲で使用します。このバッファ・オブジェクトは、データまたはレコードを処理するタスクを単純化します。

本書の以降の説明を読み進めるための前提条件として、これらのオブジェクトの使用方法について理解しておく必要があります。

データを CICS に渡す処理 (例えば、データ・レコードの書き込み) およびデータを CICS から受け取る処理 (例えば、データ・レコードの読み取り) に関する各 CICS リソース・クラスで、**IccBuf** クラスが使用されます。このようなクラスの例は、**IccConsole**、**IccDataQueue**、**IccFile**、**IccFileIterator**、**IccJournal**、**IccProgram**、**IccSession**、**IccStartRequestQ**、**IccTempStore**、および **IccTerminal** です。

IccBuf クラス

本書のリファレンス部分で詳しく説明している **IccBuf** では、データ域の一般的な操作を実行できます。

このクラスは多くの方法で利用できるため、オブジェクトの動作に影響する **IccBuf** コンストラクターがいくつかあります。これから、**IccBuf** オブジェクトの 2 つの重要な属性について説明します。

データ域の所有権

IccBuf には、データ域がオブジェクトの内部と外部のどちらに割り振られているかを示す属性があります。

この属性の値は 'internal' と 'external' のいずれかです。これを問い合わせるには、**dataAreaOwner** メソッドを使用します。

バッファの内部/外部所有権

DataAreaOwner = external である場合は、アプリケーション・プログラマーが、**IccBuf** オブジェクトの基となっているストレージの妥当性を確認する必要があります。ストレージが、オブジェクトに適用された特定のメソッドに対して無効または不適切である場合は、予測不能な結果が生じます。

データ域の拡張性

この属性は、**IccBuf** オブジェクトが作成されたら、そのオブジェクト内のデータ域の長さを伸ばすことができるかどうかを定義します。

この属性の値は 'fixed' と 'extensible' のいずれかです。これを問い合わせるには、**dataAreaType** メソッドを使用します。

'fixed' であるオブジェクトのデータ域サイズを大きくすることはできないため、**IccBuf** オブジェクトに割り当てられたデータの長さ (例えば、ファイル・レコード) は、データ域の長さを超えてはなりません。データ域の長さを超えると、C++ 例外がスローされます。

注: 定義では、'extensible' バッファは 'internal' でもある **必要があります**。

IccBuf コンストラクター

IccBuf コンストラクターには、**IccBuf** オブジェクトの作成に使用される形式がいくつかあります。

以下に例を示します。

```
IccBuf buffer;
```

これにより、初期の長さがゼロである 'internal' および 'extensible' データ域が作成されます。データがオブジェクトに割り当てられると、割り当てられているデータが収まるように、データ域長が自動的に延長されます。

```
IccBuf buffer(50);
```

これにより、初期の長さが 50 バイトである 'internal' および 'extensible' データ域が作成されます。データ長は、データがオブジェクトに割り当てられるまで、ゼロのままです。50 バイトのデータがオブジェクトに割り当てられると、データ長とデータ域長の両方が値 50 を返します。50 バイトを超えるデータがオブジェクトに割り当てられると、データが収まるように、データ域長が自動的に (つまり、ユーザーが操作しなくても) 延長されます。

```
IccBuf buffer(50, IccBuf::fixed);
```

これにより、長さが 50 バイトである 'internal' および 'fixed' データ域が作成されます。50 バイトを超えるデータをオブジェクトに割り当てようとする、データが切り捨てられ、例外がスローされて、エラー状態がアプリケーションに通知されます。

```
struct MyRecordStruct
{
    short id;
    short code;
    char data(30);
    char rating;
};
MyRecordStruct myRecord;
IccBuf buffer(sizeof(MyRecordStruct), &myRecord);
```

これにより、myRecord という名前の 'external' データ域を使用する **IccBuf** オブジェクトが作成されます。定義上は、'external' データ域は 'fixed' でもあります。データを割り当てるには、**IccBuf** オブジェクトのメソッドを使用するか、または myRecord 構造を直接使用します。

```
IccBuf buffer("Hello World");
```

これにより、長さがストリング "Hello World" の長さと同じ 'internal' および 'extensible' データ域が作成されます。このストリングは、オブジェクトのデータ域にコピーされます。この初期データ割り当ては、用意されているいずれかの操作メソッド (**insert**、**cut**、または **replace**) を使用して変更できます。

```
IccBuf buffer("Hello World");  
buffer << " out there";  
IccBuf buffer2(buffer);
```

ここでは、コピー・コンストラクターによって、最初のバッファとほとんど同じ属性が適用された 2 番目のバッファが作成されます。異なるのは、データ域所有権属性です。2 番目のオブジェクトには必ず、最初のオブジェクト内のデータ域のコピーである 'internal' データ域が含まれます。上記の例では、buffer2 には "Hello World out there" が含まれ、データ域長とデータ長はいずれも 21 となります。

IccBuf メソッド

IccBuf オブジェクトを操作するには、用意されている多数のメソッドを使用します。例えば、データをバッファに付加したり、バッファ内のデータを変更したり、バッファからデータを切り取ったり、データをバッファの中央へ挿入したりすることができます。

演算子 **const char***、**=**、**+=**、**==**、**!=**、および **<<** は、クラス **IccBuf** で多重定義されています。**IccBuf** 属性を照会できるメソッドもあります。詳細については、リファレンス・セクションを参照してください。

IccResource サブクラスの処理

IccResource サブクラスの処理を示すには、**IccTempstore** クラスを使用して CICS 一時記憶域にキュー項目を書き込むことを考えてみてください。

```
IccTempStore store("TEMP1234");  
IccBuf buffer(50);
```

作成される **IccTempStore** オブジェクトは、"TEMP1234" という名前の CICS 一時記憶域キューのアプリケーション・ビューです。作成される **IccBuf** オブジェクトのデータ域は 50 バイトです ('extensible' となる場合もあります)。

```
buffer = "Hello Temporary Storage Queue";  
store.writeItem(buffer);
```

文字ストリング "Hello Temporary Storage Queue" がバッファにコピーされます。これが可能なのは、**operator=** メソッドが **IccBuf** クラスで多重定義されているためです。

IccTempStore オブジェクトはその **writeItem** メソッドを呼び出し、最初のパラメーターとして **IccBuf** オブジェクトに対する参照を渡します。**IccBuf** オブジェクトの内容が、CICS 一時記憶域キューに書き込まれます。

ここで、CICS リソースからアプリケーション・プログラムの **IccBuf** オブジェクトヘレコードを読み取るという反対の操作について考えてみましょう。

```
buffer = store.readItem(5);
```

readItem メソッドは、CICS 一時記憶域キュー内の 5 番目の項目の内容を読み取り、**IccBuf** 参照としてデータを返します。

C++ コンパイラーは、上記のコード行を 2 つのメソッド呼び出し (**IccTempStore** クラスに定義されている **readItem**、および **IccBuf** クラスに多重定義された **operator=**) に解決します。この 2 番目のメソッドは、返された **IccBuf** 参照の内容を取得し、そのデータをバッファにコピーします。

ファウンデーション・クラスを使用してレコードを読み書きする上記のスタイルは一般的なスタイルです。最後の例では、上記の例と同様のスタイルを使用してコードを記述する方法を示します。ただし今回は、CICS 一時データ・キューにアクセスします。


```
IccDataQueue queue("DATQ");  
IccBuf buffer(50);  
buffer = queue.readItem();  
buffer << "Some extra data";  
queue.writeItem(buffer);
```

IccDataQueue オブジェクトの **readItem** メソッドが呼び出され、**IccBuf** に対する参照が返されます。これは、**buffer** オブジェクトに割り当てられます (**IccBuf** クラスで多重定義されている **operator=** メソッドを使用)。文字ストリング "Some extra data" がバッファに付加されます (**IccBuf** クラスで多重定義されている **operator chevron** « メソッドを使用)。その後、**writeItem** メソッドによって、変更されたこのバッファが CICS 一時データ・キューに書き込まれます。

以下のセクションのサンプルでは、この構文の例をさらに示し、ファウンデーション・クラスを使用して CICS サービスにアクセスする方法について説明します。

IccBuf クラスの詳細については、リファレンス・セクションを参照してください。また、用意されているサンプル **ICC\$BUF** も役立ちます。

CICS Service の使用

このセクションでは、CICS サービスの使用方法について説明します。サービスについて順に考えてみましょう。

ファイル制御

ファイル制御クラス **IccFile**、**IccFileId**、**IccKey**、**IccRBA**、および **IccRRN** を使用すると、ファイル内のレコードを読み取り、書き込み、更新、および削除できます。

さらに、**IccFileIterator** クラスを使用すると、ファイル内のすべてのレコードをブラウズできます。

IccFile オブジェクトは、ファイルを表すために使用されます。**IccFileId** オブジェクトを使用して、ファイルを名前で識別する方法は便利ですが、必須ではありません。

アプリケーション・プログラムは、個々のレコードの形式でそのデータを読み書きします。読み取りまたは書き込みの各要求は、メソッド呼び出しから出されます。レコードにアクセスするには、ファイルと特定のレコードの両方をプログラムが識別する必要があります。

VSAM (または VSAM に類似した) ファイルは、以下のタイプです。

KSDS

キー順: 各レコードがキー (レコード内の事前定義位置にあるフィールド) で識別されます。各キーは、ファイル内で固有である必要があります。

ファイル内のレコードの論理順序は、キーによって決まります。物理位置は、VSAM が保持する索引に組み込まれます。

レコードをブラウズすると、それらの論理順序で見つかります。

ESDS

入力順: 各レコードが相対バイト・アドレス (RBA) で識別されます。

レコードは、最初にファイルにロードされた順序で ESDS に保持されます。新しいレコードは必ず末尾に追加されます。レコードは、削除したり、その長さを変更したりすることはできません。

レコードをブラウズすると、最初に書き込まれた順序で見つかります。

RRDS ファイル

相対レコード: レコードが固定長スロットに書き込まれます。レコードは、それを保持するスロットの相対レコード番号 (RRN) で識別されます。

レコードの読み取り

読み取り操作には、2 つのクラスを使用します。操作を実行するための **IccFile** と、ファイル・アクセス・タイプが KSDS、ESDS、RRDS のいずれかに応じて、特定のレコードを識別するための **IccKey**、**IccRBA**、**IccRRN** のいずれかです。

IccFile クラスの **readRecord** メソッドは、レコードを読み取ります。

KSDS レコードの読み取り

レコードを読み取る前に、**IccFile** の **registerRecordIndex** メソッドを使って、**IccKey** クラスのオブジェクトとファイルを関連付ける必要があります。

レコードにアクセスするには、**IccKey** オブジェクトに保持されているキーを使用する必要があります。「完全」キーは、物理ファイルのキーの長さと同じ長さの文字ストリングです。各レコードは、それぞれの完全なキーにより個別に特定されます。

また、キーを「総称」にすることもできます。総称キーは完全キーより短く、一連のレコードを検索するために使用されます。**IccKey** クラスには、キーの設定や変更を行うことができるメソッドがあります。

IccFile クラスには、**isReadable**、**keyLength**、**keyPosition**、**recordIndex**、および **recordLength** の各メソッドがあり、これらは KSDS レコードの読み取りに役立ちます。

ESDS レコードの読み取り

レコードの先頭にアクセスするには、**IccRBA** オブジェクト内に保持されている相対バイト・アドレス (RBA) を使用する必要があります。

レコードを読み取る前に、**IccFile** の **registerRecordIndex** メソッドを使って、**IccRBA** クラスのオブジェクトとファイルを関連付ける必要があります。

IccFile クラスには、**isReadable**、**recordFormat**、**recordIndex**、および **recordLength** の各メソッドがあり、これらは ESDS レコードの読み取りに役立ちます。

RRDS レコードの読み取り

レコードにアクセスするには、**IccRRN** オブジェクト内に保持されている相対レコード番号 (RRN) を使用する必要があります。

レコードを読み取る前に、**IccFile** の **registerRecordIndex** メソッドを使って、**IccRRN** クラスのオブジェクトとファイルを関連付ける必要があります。

IccFile クラスには、**isReadable**、**recordFormat**、**recordIndex**、および **recordLength** の各メソッドがあり、これらは RRDS レコードの読み取りに役立ちます。

レコードの書き込み

レコードの書き込みは、「レコードの追加」とも呼ばれます。

このトピックでは、これまで書き込まれたことがないレコードの書き込みについて説明します。既に存在するレコードの書き込みは、そのレコードがそれまでに「更新」モードになっている場合を除いて許可されません。詳しくは、[17 ページの『レコードの更新』](#)を参照してください。

レコードを書き込む前に、**IccFile** の **registerRecordIndex** メソッドを使って、**IccKey**、**IccRBA**、または **IccRRN** クラスのオブジェクトとファイルを関連付ける必要があります。**IccFile** クラスの **writeRecord** メソッドは、レコードを書き込みます。

書き込み操作には、2つのクラスを使用します。操作を実行するための **IccFile** と、ファイル・アクセス・タイプが KSDS、ESDS、RRDS のいずれかに応じて、特定のレコードを識別するための **IccKey**、**IccRBA**、**IccRRN** のいずれかです。

書き込むレコードが複数あるときは、データの大量挿入を使用して、書き込み速度を向上させることができます。この大量挿入の開始と終了は、**IccFile** の **beginInsert** および **endInsert** メソッドを呼び出して行います。

KSDS レコードの書き込み

レコードにアクセスするには、**IccKey** オブジェクトに保持されているキーを使用する必要があります。

「完全」キーは、レコードを一意的に識別する文字ストリングです。各レコードは、それぞれの完全なキーにより個別に特定されます。

IccFile クラスの **writeRecord** メソッドは、レコードを書き込みます。

IccFile クラスには、**isAddable**、**keyLength**、**keyPosition**、**recordIndex**、**recordLength**、および **registerRecordIndex** の各メソッドがあり、これらは KSDS レコードの書き込みに役立ちます。

ESDS レコードの書き込み

レコードの先頭にアクセスするには、**IccRBA** オブジェクト内に保持されている相対バイト・アドレス (RBA) を使用する必要があります。

IccFile クラスには、**isAddable**、**recordFormat**、**recordIndex**、**recordLength**、および **registerRecordIndex** の各メソッドがあり、これらは ESDS レコードの書き込みに役立ちます。

RRDS レコードの書き込み

新しい ESDS レコードを追加するには、**writeRecord** メソッドを使用します。

IccFile クラスには、**isAddable**、**recordFormat**、**recordIndex**、**recordLength**、および **registerRecordIndex** の各メソッドがあり、これらは RRDS レコードの書き込みに役立ちます。

レコードの更新

レコードの更新は、「レコードの再書き込み」とも呼ばれます。

レコードを更新する前に、まず 'update' モードで **readRecord** メソッドを使用して読み取る必要があります。これによりレコードがロックされるため、どのユーザーも変更できなくなります。

レコードを更新するには、**rewriteRecord** メソッドを使用します。**IccFile** オブジェクトは、処理中のレコードを記憶しますが、この情報が再び渡されることはありません。

例については、『[コード・フラグメント: 「更新用にレコードを読み取る」](#)』を参照してください。

レコードを変更するときに、KSDS ファイル内の基本キーを変更してはなりません。ファイル定義に可変長レコードを使用できる場合には、レコードの長さが変更されることがあります。

ESDS、RRDS、または固定長の KSDS ファイル内のレコードの長さを、更新時に変更してはなりません。

固定長レコードを含むファイルとして CICS に定義されているファイルの場合は、更新するレコードの長さが、元の長さと同じである必要があります。更新されたレコードの長さは、VSAM に定義されている最大以下である必要があります。

レコードの削除

ESDS ファイルからレコードを削除することは絶対にできません。

通常レコードの削除

IccFile クラスの **deleteRecord** メソッドは、レコードが「更新」モードの状態であるという効果によってロックされていない場合に、1 つ以上のレコードを削除します。

削除するレコードは、**IccKey** または **IccRRN** オブジェクトによって定義されます。

ロックされたレコードの削除

IccFile クラスの **deleteLockedRecord** メソッドは、**readRecord** メソッドによって「更新」モードの状態にあるという効果により、それまでにロックされているレコードを削除します。

レコードのブラウズ

ブラウズ、つまりファイルの順次読み取りでは、別のクラス **IccFileIterator** を使用します。

このクラスのオブジェクトは、**IccFile** オブジェクトのほか、**IccKey**、**IccRBA**、**IccRRN** のいずれかのオブジェクトに関連付けられている必要があります。この関連付けを行った後は、その他のオブジェクトをさらに参照することなく、**IccFileIterator** オブジェクトを使用できます。

readNextRecord メソッドを使用して順方向へ、または **readPreviousRecord** メソッドを使用して逆方向へ、いずれかのブラウズを実行できます。 **reset** メソッドは、**IccKey** オブジェクトまたは **IccRBA** オブジェクトで指定されたレコードを指すように **IccFileIterator** オブジェクトをリセットします。

ファイルのブラウズの例は、[コード・フラグメント「すべてのレコードをキーの昇順にリストする \(List all records in assending order of key\)」](#)のページに示されています。

ファイル制御の例

このサンプル・プログラムは、**IccFile** クラスと **IccFileIterator** クラスの使用法を示しています。

このサンプルのソースは、[C++ サンプル・プログラムのファイル ICC\\$FIL](#)にあります。ここでは、ソース・ファイルには見られる端末の入出力を省略して、コードを示しています。

```
#include "icceh.hpp"
#include "iccmmain.hpp"
```

最初の 2 行は、ファウンデーション・クラスのヘッダー・ファイルと、アプリケーション・プログラムの稼働環境をセットアップする標準 **main** 関数を組み込んでいます。

```
const char* fileRecords[] =
{
//NAME KEY PHONE USERID
"BACH, J S 003 00-1234 BACH ",
"BEETHOVEN, L 007 00-2244 BEET ",
"CHOPIN, F 004 00-3355 CHOPIN ",
"HANDEL, G F 005 00-4466 HANDEL ",
"MOZART, W A 008 00-5577 WOLFGANG "
};
```

これは、サンプル・プログラムによって使用される複数行のデータを定義します。

```
void IccUserControl::run()
{
```

IccUserControl クラスの **run** メソッドに、この例のユーザー・コードが含まれています。端末を使用するため、このサンプルは端末オブジェクトの作成と関連する画面の消去から始まります。

```
    short recordsDeleted = 0;
    IccFileId id("ICCKFILE");
    IccKey key(3,IccKey::generic);
    IccFile file( id );
    file.registerRecordIndex( &key );
    key = "00";
    recordsDeleted = file.deleteRecord();
```

最初に **key** オブジェクトと **file** オブジェクトが作成され、これらを使用して、KSDS ファイル「ICCKFILE」内の「00」で始まるキーを持つレコードがすべて削除されます。 **key** は 3 バイトの総称キーとして定義されますが、このインスタンスでは、先頭 2 バイトのみが使用されます。

```
    IccBuf buffer(40);
    key.setKind( IccKey::complete );
    for (short j = 0; j < 5; j++)
    {
        buffer = fileRecords[j];
        key.assign(3, fileRecords[j]+15);
        file.writeRecord( buffer );
    }
```

この次のフラグメントは、指定されたすべてのデータを、ファイルのレコードに書き込みます。このデータは、この目的のために作成される **IccBuf** オブジェクトによって渡されます。 **setKind** メソッドを使用して、**key** を「generic」から「complete」に変更します。

これらの呼び出し間の **for** ループでは、すべてのデータをループしながら、**IccBuf** の **operator=** メソッドを使用してバッファにデータを渡してから、**writeRecord** を使用してファイル内のレコードに渡します。途中で、**assign** を使用して、各レコードのキーがデータ内で出現する文字ストリング (15 文字目から 3 文字) になるように設定します。

```

IccFileIterator fIterator( &file,
                           &key );
key = "000";
buffer = fIterator.readNextRecord();
while (fIterator.condition() == IccCondition::NORMAL)
{
term->sendLine("- record read: [%s]",(const char*) buffer);
buffer = fIterator.readNextRecord();
}

```

ここに示すループは、**sendLine** を使用して、すべてのレコードをキーの昇順で端末にリストします。ここでは **IccFileIterator** オブジェクトを使用してレコードをブラウズします。この例にはありませんが、キーの最小値が設定された場合は、CICS に依存して、キー・シーケンスで最初のレコードを検索して、このループが開始されます。

ループは、NORMAL 以外の条件が返されるまで続行されます。

```

key = "FFFF";
fIterator.reset( &key );
buffer = fIterator.readPreviousRecord();
while (fIterator.condition() == IccCondition::NORMAL)
{
buffer = fIterator.readPreviousRecord();
}

```

次のループは最後のループとほぼ同じですが、レコードがキーの逆順でリストされます。

```

key = "008";
buffer = file.readRecord( IccFile::update );
buffer.replace( 4, "5678", 23);
file.rewriteRecord( buffer );

```

このフラグメントは、更新のためにレコードを読み取り、他のプログラムが変更できないようにそのレコードをロックします。次に、バッファ内のレコードを変更し、更新したレコードをファイルに書き込みます。

```

buffer = file.readRecord();

```

実際に更新したことを示すため、同じレコードを再び読み取り、端末に送信します。

```

return;
}

```

run が終了し、制御が CICS に戻ります。

このサンプルからの予期される出力については、[C++ サンプル・プログラム](#)を参照してください。

プログラム制御

このセクションでは、現在実行されているプログラム以外のプログラムにアクセスし、それを使用する方法について説明します。

プログラム制御では、リソース・クラスの 1 つである **IccProgram** クラスを使用します。

プログラムは、**IccProgram** オブジェクトを使用して、ロード、アンロード、およびリンクできます。**IccProgram** オブジェクトを問い合わせると、プログラムに関する情報を取得できます。詳細については、[IccProgram クラス](#)を参照してください。

以下に示す例では、1 つのプログラムがほかの 2 つのプログラムを順に呼び出します。これらの間でデータは COMMAREA 経由で受け渡されます。1 つのプログラムがローカルであることが想定され、2 番目のプログラムはリモート CICS システム上にあります。プログラムは 2 つのファイル ICC\$PRG1 および ICC\$PRG2 に含まれています。これらのファイルの位置と、これらのサンプル・プログラムから予期される出力については、[C++ サンプル・プログラム](#)を参照してください。

これらのサンプル内の端末入出力のほとんどが、以下のコードでは省略されています。

```
#include "icceh.hpp"
#include "iccmmain.hpp"
void IccUserControl::run()
{
```

両方のプログラムのコードが、まずファウンデーション・クラスのヘッダー・ファイルと **main** メソッドのスタブから始まります。ユーザー・コードは、各プログラムの **IccUserControl** クラスの **run** メソッド内にあります。

```
IccSysId sysId( "ICC2" );
IccProgram icc$prg2( "ICC$PRG2" );
IccProgram remoteProg( "ICC$PRG3" );
IccBuf commArea( 100, IccBuf::fixed );
```

最初のプログラム (ICC\$PRG1) では、リモート領域を表す **IccSysId** オブジェクト、およびこのプログラムから呼び出されるローカル・プログラムとリモート・プログラムを表す 2 つの **IccProgram** オブジェクトが作成されます。100 バイトの固定長バッファ・オブジェクトも作成され、プログラム間の通信領域として使用されます。

```
icc$prg2.load();
if (icc$prg2.condition() == IccCondition::NORMAL)
{
term->sendLine( "Loaded program: %s <%s> Length=%ld Address=%x",
icc$prg2.name(),
icc$prg2.conditionText(),
icc$prg2.length(),
icc$prg2.address() );
icc$prg2.unload();
}
```

その後プログラムは、プログラム ICC\$PRG2 のプロパティをロードし問い合わせようとしています。

```
commArea = "DATA SET BY ICC$PRG1";
icc$prg2.link( &commArea );
```

通信域バッファは、ICC\$PRG1 のリンク先の最初のプログラム (ICC\$PRG2) に渡されるデータを格納するように設定されます。ICC\$PRG1 は、ICC\$PRG2 の実行中には中断状態になります。

呼び出されるプログラム ICC\$PRG2 は単純なプログラムで、その骨子は以下のとおりです。

```
IccBuf& commArea = IccControl::commArea();
commArea = "DATA RETURNED BY ICC$PRG2";
return;
```

ICC\$PRG2 は、渡された通信域へのアクセスを取得します。その後、この通信域のデータを変更し、呼び出し側のプログラムに制御を返します。

この時点で最初のプログラム (ICC\$PRG1) が、以下のように別のシステムで別のプログラムを呼び出します。

```
remoteProg.setRouteOption( sysId );
commArea = "DATA SET BY ICC$PRG1";
remoteProg.link( &commArea );
```

setRouteOption は、このオブジェクトの呼び出しがリモート・システムにルーティングされるように要求します。通信域が再設定され (ICC\$PRG2 によって変更されているため)、リモート・プログラム (システム ICC2 上の ICC\$PRG3) にリンクされます。

呼び出されたプログラムは、CICS 一時記憶域を使用しますが、以下の 3 行について考えてみましょう。

```
IccBuf& commArea = IccControl::commArea();
commArea = "DATA RETURNED BY ICC$PRG3";
return;
```

リモート・プログラム (ICC\$PRG3) が再度、渡された通信域へのアクセスを取得します。この通信域のデータを変更し、呼び出し側のプログラムに制御を返します。

```
return;  
};
```

最後に、呼び出し側プログラムそのものが終了し、制御を CICS に返します。

トランザクションの非同期の開始

IccStartRequestQ クラスを使用すると、プログラムは別の CICS トランザクション・インスタンスを非同期で開始できます(さらに、オプションでデータを開始済みトランザクションに渡すことができます)。

開始済みトランザクションでは同じクラスを使用して、開始要求を発行したタスクから渡されたデータへのアクセスを取得します。最後に、開始要求を後から取り消すことができます。

トランザクションの開始

以下のいずれかのメソッドを使用して、どのデータを、開始されるトランザクションへ送信するかを設定できます。

- **registerData** または **setData**
- **setQueueName**
- **setReturnTermId**
- **setReturnTransId**

実際の開始は、**start** メソッドを使用して要求します。

開始データへのアクセス

開始されるトランザクションは、**retrieveData** メソッドを呼び出すことにより、自己の開始データにアクセスできます。

このメソッドはすべての開始データ属性を **IccStartRequestQ** オブジェクトに保管します。個々の属性には、以下のメソッドを使用してアクセスできます。

- **data**
- **queueName**
- **returnTermId**
- **returnTransId**

満了していない開始要求のキャンセル

満了していない開始要求(つまり、まだ到達していない将来の時点の開始要求)は、**cancel** メソッドを使用してキャンセルすることができます。

トランザクション開始の例

システム ICC1 の端末 PEO1 でトランザクション ISR1 を開始します。

CICS システム	ICC1	ICC2
トランザクション	ISR1/ITMP	ISR2
プログラム	ICC\$SRQ1/ICC\$TMP	ICC\$SRQ2
端末	PEO1	PEO2

これは、2つの開始要求を発行し、最初の要求は満了前に取り消されます。2番目の要求は、システム ICC2 の端末 PEO2 でトランザクション ISR2 を開始します。このトランザクションは、その開始データにアクセスし、元の端末(システム ICC1 の PEO1)でトランザクション ITMP を開始して、トランザクションが終了します。

プログラムとその予期される出力は、C++ サンプル・プログラムに、ファイル ICC\$SRQ1 および ICC\$SRQ2 として用意されています。ここでは、端末の入出力要求を省略してコードを示しています。

トランザクション ISR1 は、システム ICC1 でプログラム ICC\$SRQ1 を実行します。最初にこのプログラムについて考えてみます。

```
#include "icceh.hpp"
#include "iccmmain.hpp"
void IccUserControl::run()
{
```

これらの行では、ファウンデーション・クラスのヘッダー・ファイルと、アプリケーション・プログラムのクラス・ライブラリーをセットアップするために必要な **main** 関数を組み込んでいます。

IccUserControl クラスの **run** メソッドに、この例のユーザー・コードが含まれています。

```
    IccRequestId req1;
    IccRequestId req2("REQUEST1");
    IccTimeInterval ti(0,0,5);
    IccTermId remoteTermId("PE02");
    IccTransId ISR2("ISR2");
    IccTransId ITMP("ITMP");
    IccBuf buffer;
    IccStartRequestQ* startQ = startRequestQ();
```

ここでは、以下のようないくつかのオブジェクトを作成しています。

req1

特定の開始要求を識別するために作動可能な、空の **IccRequestId** オブジェクト。

req2

ユーザーが提供する ID 「REQUEST1」を格納する **IccRequestId** オブジェクト。

ti

0 時 0 分 5 秒を表す **IccTimeInterval** オブジェクト。

remoteTermId

IccTermId オブジェクト。トランザクションを開始するリモート・システムの端末を表します。

ISR2

IccTransId オブジェクト。リモート・システムで開始するトランザクションを表します。

ITMP

IccTransId オブジェクト。開始されたトランザクションが、このプログラムの端末で開始するトランザクションを表します。

buffer

開始データを保持する **IccBuf** オブジェクト。

最後に、**IccControl** クラスの **startRequestQ** メソッドが、単一インスタンス (singleton) クラス **IccStartRequestQ** にポインターを返します。

```
    startQ->setRouteOption( "ICC2" );
    startQ->registerData( &buffer );
    startQ->setReturnTermId( terminal()->name() );
    startQ->setReturnTransId( ITMP );
    startQ->setQueueName( "startqnm" );
```

このコード・フラグメントでは、開始要求を発行するときに渡される開始データを作成します。

setRouteOption は、リモート・システム ICC2 で開始要求を発行することを指示します。 **registerData** メソッドは、開始データを含む **IccBuf** オブジェクトを関連付けます (**IccBuf** オブジェクトの内容は、開始要求を発行するまで抽出されません)。 **setReturnTermId** メソッドと **setReturnTransId** メソッドにより、開始要求側がトランザクションと端末名を、開始されるトランザクションに渡すことができます。開始されるトランザクションが別の端末、この例の場合は発信元の端末で、指定されたとおりに **別の** トランザクションを開始できるよう、これらのフィールドが一般的に使用されます。

setQueueName は、開始されるトランザクションに渡すことができる、もう 1 つの情報です。

```
    buffer = "This is a greeting from program
'icc$srq1'!!";
    req1 = startQ->start( ISR2, &remoteTermId, &ti );
    startQ->cancel( req1 );
```


ここで、開始要求で渡すデータを設定します。時間間隔 ti (5 秒) の経過後、トランザクション **ISR2** を開始します。要求 ID は、*req1* に保管されます。5 秒経過する前に (つまり、即時に) 開始要求を取り消します。

```
req1 = startQ->start( ISR2, &remoteTermID,  
&ti, &req2 );  
return;  
}
```

時間間隔 ti (5 秒) の経過後、トランザクション **ISR2** を再度開始します。今度はその要求を満了させるため、トランザクション **ISR2** はリモート・システムで開始されます。それと同時に、CICS に制御を返して終了します。

次に、開始されるプログラム **ICC\$SRQ2** について考えてみます。

```
IccBuf buffer;  
IccRequestId req("REQUESTX");  
IccTimeInterval ti(0,0,5);  
IccStartRequestQ* startQ = startRequestQ();
```

ここでは、**ICC\$SRQ1** の場合と同様に、いくつかのオブジェクトを作成します。

buffer

呼び出し元 (**ICC\$SRQ1**) から渡された開始データを保持する **IccBuf** オブジェクト。

req

呼び出し元の端末で発行する開始を識別する **IccRequestId** オブジェクト。

ti

0 時 0 分 5 秒を表す **IccTimeInterval** オブジェクト。

IccControl クラスの **startRequestQ** メソッドは、singleton クラス **IccStartRequestQ** を指すポインターを返します。

```
if ( task()->startType() != IccTask::startRequest )  
{  
term->sendLine(  
"This program should only be started via the StartRequestQ");  
task()->abend( "OOPS" );  
}
```

ここでは、**IccTask** クラスの **startType** メソッドを使用して、**ICC\$SRQ2** が **start** メソッドによって開始されていて、他の方法 (端末でのトランザクション名の入力など) では開始されていないことを確認します。意図したとおりに開始されていなかった場合は、異常終了し、「OOPS」という異常終了コードが返されます。

```
startQ->retrieveData();
```

ICC\$SRQ1 によって渡された開始データを取得し、以降のアクセス用に **IccStartRequestQ** オブジェクト内に保管します。

```
buffer = startQ->data();  
term->sendLine( "Start buffer contents = [%s]", buffer.dataArea() );  
term->sendLine( "Start queue= [%s]", startQ->queueName() );  
term->sendLine( "Start rtrn = [%s]",  
startQ->returnTransId().name());  
term->sendLine( "Start rtrm = [%s]", startQ->returnTermId().name() );
```

この開始データ・バッファは、**IccBuf** オブジェクトにコピーされます。その他の開始データ項目 (キュー、**returnTransId**、および **returnTermId**) は端末に表示されます。

```
task()->delay( ti );
```

5 秒間遅延します (つまり、スリープして何も実行しません)。

```
startQ->setRouteOption( "ICC1" );
```

setRouteOption は、呼び出し元のシステム (**ICC1**) での開始をシグナル通知します。

```
startQ->start(
startQ->returnTransId(),startQ->returnTermId());
return;
```

ITMP (ICC\$SRQ1 によって returnTransId 開始情報で渡されたトランザクションの名前) というトランザクションを発信元の (このトランザクションを開始したときに ICC\$SRQ1 が完了した) 端末で開始します。開始要求を発行したら、ICC\$SRQ1 は、制御を CICS に戻して、終了します。

最後に、トランザクション ITMP が最初の端末で実行されます。これで、トランザクションを非同期で開始する、このデモンストレーションは終了です。

一時データ

一時データ・クラス **IccDataQueue** および **IccDataQueueId** を使用すると、データを後続の処理のために一時データ・キューに保管できます。

以下のことができます。

- 一時データ・キューからデータを読み取る (**readItem** メソッド)
- 一時データ・キューにデータを書き込む (**writeItem** メソッド)
- 一時データ・キューを削除する (**empty** メソッド)

IccDataQueue オブジェクトは、一時記憶域キューを表すために使用します。**IccDataQueueId** オブジェクトは、キューを名前で識別するために使用します。**IccDataQueueId** オブジェクトが初期化されたら、キューの名前を使用する代わりに、このオブジェクトを使用してキューを識別できます。これにより、C++ コンパイラでさらにエラーを検出できます。

IccDataQueue クラスで使えるメソッドは、**IccTempStore** クラスのメソッドに類似しています。これらの詳細については、[『25 ページの『一時記憶』』](#)を参照してください。

データの読み取り

キューから項目を読み取るには、**readItem** メソッドを使用します。

このメソッドは、情報を含んでいる **IccBuf** オブジェクトへの参照を返します。

データの書き込み

IccDataQueue の **writeItem** メソッドは、データの新しい項目をキューに追加し、指定されたバッファからデータを取得します。

キューの削除

empty メソッドは、キューに入っているすべての項目を削除します。

一時データ管理の例

このサンプル・プログラムは、**IccDataQueue** クラスと **IccDataQueueId** クラスの使用法を示しています。

このプログラムのソースと、このプログラムからの予期される出力は、ファイル ICC\$DAT として [C++ サンプル・プログラム](#) にあります。ここでは、端末の入出力要求を省略してコードを示しています。

```
#include "icceh.hpp"
#include "iccmmain.hpp"
```

最初の 2 行は、ファウンデーション・クラスのヘッダー・ファイルと、アプリケーション・プログラムの稼働環境をセットアップする標準 **main** 関数を組み込んでいます。

```
const char* queueItems[] =
{
    "Hello World - item 1",
    "Hello World - item 2",
    "Hello World - item 3"
};
```

これは、サンプル・プログラム用のバッファを定義しています。

```
void IccUserControl::run()
{
```

IccUserControl クラスの **run** メソッドに、この例のユーザー・コードが含まれています。

```
    short itemNum =1;
    IccBuf buffer( 50 );
    IccDataQueueId id( "ICCQ" );
    IccDataQueue queue( id );
    queue.empty();
```

このフラグメントは最初に、「ICCQ」を含む、IccDataQueueId のタイプの識別オブジェクトを作成します。次に、一時データ・キュー「ICCQ」を表す **IccDataQueue** オブジェクトを作成し、データを空にします。

```
    for (short i=0 ; i<3 ; i++)
    {
        buffer = queueItems[i];
        queue.writeItem( buffer );
    }
```

このループでは、3つのデータ項目を一時データ・オブジェクトに書き込みます。このデータは、この目的のために作成された **IccBuf** オブジェクトによって渡されます。

```
        buffer = queue.readItem();
        while ( queue.condition() == IccCondition::NORMAL )
        {
            buffer = queue.readItem();
        }
```

3つのレコードの書き出しが完了したら、今度はこれらを読み戻し、正常に書き込まれたことを示します。

```
    return;
}
```

run が終了し、制御が CICS に戻ります。

一時記憶

一時記憶域クラス **IccTempStore** および **IccTempStoreId** を使用すると、データを一時記憶域キューに保管できます。

以下のことができます。

- 一時記憶域キューから項目を読み取る (**readItem** メソッド)
- 一時記憶域キューの終わりに新規項目を書き込む (**writeItem** メソッド)
- 一時記憶域キュー内の項目を更新する (**rewriteItem** メソッド)
- 一時記憶域キュー内の次の項目を読み取る (**readNextItem** メソッド)
- 一時データをすべて削除する (**empty** メソッド)

IccTempStore オブジェクトは、一時記憶域キューを表すために使用します。**IccTempStoreId** オブジェクトは、キューを名前で識別するために使用します。**IccTempStoreId** オブジェクトが初期化されたら、キューの名前を使用する代わりに、このオブジェクトを使用してキューを識別できます。これにより、C++ コンパイラでさらにエラーを検出できます。

IccTempStore クラスで使えるメソッドは、**IccDataQueue** クラスのメソッドに類似しています。これらの詳細については、『[24 ページの『一時データ』](#)』を参照してください。

項目の読み取り

IccTempStore の **readItem** メソッドは、指定された項目を一時記憶域キューから読み取ります。

このメソッドは、情報を含んでいる **IccBuf** オブジェクトへの参照を返します。

項目の書き込み

項目の書き込みは、項目の「追加」とも呼ばれます。

このセクションでは、これまで書き込まれたことのない項目の書き込みについて説明します。**rewriteItem** メソッドを使用して、既に存在する項目の書き込みを行うことができます。詳しくは、『[26 ページの『項目の更新』](#)』を参照してください。

IccTempStore の **writeItem** メソッドは、新しい項目をキューの末尾に追加し、指定されたバッファーからデータを取得します。これが正常に実行された場合は、追加されたレコードの項目番号が返されます。

項目の更新

項目の更新は、項目の「再書き込み」とも呼ばれます。

一時記憶域キュー内の指定した項目を更新するには、**IccTempStore** クラスの **rewriteItem** メソッドを使用します。

項目の削除

一時記憶域キュー内の個々の項目を削除することはできません。

IccTempStore オブジェクトに関連付けられているすべての一時データを削除するには、**IccTempStore** クラスの **empty** メソッドを使用します。

一時記憶域の例

このサンプル・プログラムは、**IccTempStore** クラスと **IccTempStoreId** クラスの使用法を示しています。

このプログラムと、このプログラムからの予期される出力は、ファイル **ICC\$TMP** として [C++ サンプル・プログラム](#) にあります。このサンプルは、ここでは、端末の入出力要求を省略して示しています。

```
#include "icceh.hpp"
#include "iccmmain.hpp"
#include <stdlib.h>
```

最初の 3 行は、ファウンデーション・クラスのヘッダー・ファイル、アプリケーション・プログラムの稼働環境をセットアップする標準 **main** 関数、および標準ライブラリーを組み込んでいます。

```
const char* bufferItems[] =
{
    "Hello World - item 1",
    "Hello World - item 2",
    "Hello World - item 3"
};
```

これは、サンプル・プログラム用のバッファーを定義しています。

```
void IccUserControl::run()
{
```

IccUserControl クラスの **run** メソッドに、この例のユーザー・コードが含まれています。

```
    short itemNum = 1;
    IccTempStoreId id("ICCSTORE");
    IccTempStore store( id );
    IccBuf buffer( 50 );
    store.empty();
```

このフラグメントは最初に、フィールド「ICCSTORE」を含む識別オブジェクト **IccTempStoreId** 作成します。次に、一時記憶域キュー「ICCSTORE」を表す **IccTempStore** オブジェクトを作成し、レコードを空にします。

```
for (short j=1 ; j <= 3 ; j++)
{
    buffer = bufferItems[j-1];
    store.writeItem( buffer );
}
```

このループでは、3つのデータ項目を一時記憶域オブジェクトに書き込みます。このデータは、この目的のために作成された **IccBuf** オブジェクトによって渡されます。

```
buffer = store.readItem( itemNum );
while ( store.condition() == IccCondition::NORMAL )
{
    buffer.insert( 9, "Modified " );
    store.rewriteItem( itemNum, buffer );
    itemNum++;
    buffer = store.readItem( itemNum );
}
```

この次のフラグメントでは、項目を読み戻して変更し、一時記憶域キューに再書き込みします。最初に、**readItem** メソッドを使用して、一時記憶域オブジェクトからバッファを読み取ります。**IccBuf** クラスの **insert** メソッドを使用してバッファ・オブジェクトのデータが変更された後、**rewriteItem** メソッドによってバッファが上書きされます。ループでは、次のバッファ項目の読み取りが続行されます。

```
itemNum = 1;
buffer = store.readItem( itemNum );
while ( store.condition() == IccCondition::NORMAL )
{
    term->sendLine( " - record #%d = [%s]", itemNum,
        (const char*)buffer );
    buffer = store.readNextItem();
}
```

このループでは、一時記憶域キュー項目を再び読み取り、これらの項目が更新されたことを表示します。

```
return;
}
```

run が終了し、制御が CICS に戻ります。

端末管理

端末管理クラス **IccTerminal**、**IccTermId**、および **IccTerminalData** を使用すると、CICS タスクに属する端末へのデータの送信、端末からのデータの受信、および端末に関する情報の検索を実行できます。

IccTerminal オブジェクトは、CICS タスクに属する端末を表すために使用されます。トランザクションの基本装置が 3270 端末である場合にのみ、このオブジェクトを作成できます。**IccTermId** クラスは、端末を識別するために使用します。**IccTerminal** が所有する **IccTerminalData** には、端末特性に関する情報が含まれます。

端末へのデータの送信

IccTerminal クラスの **send** および **sendLine** メソッドは、画面へのデータの書き込みに使用されます。

set... メソッドを使用して、これを実行できます。また、**erase** メソッドを使用して現在端末で表示されているデータを消去し、**freeKeyboard** メソッドを使用してキーボードを解放し、入力を受け取れるように準備します。

端末からのデータの受信

IccTerminal クラスの **receive** メソッドと **receive3270data** メソッドは、端末からデータを受信するために使用されます。

端末に関する情報の検索

端末の特性と端末の現在の状態に関する情報を検索できます。

data オブジェクトは、端末の特性に関する情報を含む **IccTerminalData** オブジェクトを指し示します。**IccTerminalData** 内のメソッドを使用すると、例えば、画面の縦の長さや、端末が消去書き込み代替をサポートしているかなどを検出できます。**IccTerminal** のメソッドの中には、ある画面が保持する行数などの特性に関する情報を取得するメソッドがあります。

その他に、端末の現在の状態に関する情報を取得できるメソッドもあります。これらのメソッドには、現在の行番号を返す **line** や、現行カーソル位置を返す **cursor** があります。

端末管理の例

このサンプル・プログラムは、**IccTerminal**、**IccTermId**、および **IccTerminalData** の各クラスの使用法を示しています。

このプログラムと、このプログラムからの予期される出力は、ファイル ICC\$TRM として [C++ サンプル・プログラム](#) にあります。

```
#include "icceh.hpp"
#include "iccmmain.hpp"
```

最初の 2 行は、ファウンデーション・クラスのヘッダー・ファイルと、アプリケーション・プログラムの稼働環境をセットアップする標準 **main** 関数を組み込んでいます。

```
void IccUserControl::run()
{
    IccTerminal& term = *terminal();
    term.erase();
```

IccUserControl クラスの **run** メソッドに、この例のユーザー・コードが含まれています。端末を使用するため、このサンプルは端末オブジェクトの作成と関連する画面の消去から始まります。

```
    term.sendLine( "First part of the line..." );
    term.send( "... a continuation of the line." );
    term.sendLine( "Start this on the next line" );
    term.sendLine( 40, "Send this to column 40 of current line" );
    term.send( 5, 10, "Send this to row 5, column 10" );
    term.send( 6, 40, "Send this to row 6, column 40" );
```

このフラグメントは、**send** および **sendLine** メソッドを使用して、データを端末に送信する方法を示します。これらのメソッドはすべて、ストリング・リテラル (const char*) の代わりに、**IccBuf** 参照 (const IccBuf&) を取ることができます。

```
    term.setNewLine();
```

これで、画面にブランク行が送信されます。

```
    term.setColor( IccTerminal::red );
    term.sendLine( "A Red line of text." );
    term.setColor( IccTerminal::blue );
    term.setHighlight( IccTerminal::reverse );
    term.sendLine( "A Blue, Reverse video line of text." );
```

setColor メソッドは、画面のテキストの色を設定するため、**setHighlight** メソッドは、強調表示を設定するために使用されます。

```
    term << "A cout sytle interface... " <<
endl;
    term << "you can " << "chain input together; "
<< "use different types, eg numbers: " << (short)123 <<
" "
<< (long)4567890 << " " << (double)123456.7891234
<< endl;
    term << "... and everything is buffered till you issue a flush."
<< flush;
```

このフラグメントは、`iostream` に似たインターフェース **endl** を使用して、次の行でデータを開始する方法を示します。パフォーマンスを高めるために、画面にデータを送信する **flush** が発行されるまで、データを端末のバッファに入れることができます。

```
term.send( 24,1, "Program 'icc$trm' complete: Hit PF12
to End" );
term.waitForAID( IccTerminal::PF12 );
term.erase();
```

waitForAID メソッドは、**erase** メソッドを呼び出して表示をクリアする前に、指定されたキーが押されるまで端末を待機させます。

```
return;
}
```

run が終了し、制御が CICS に戻ります。

日時サービス

IccClock クラスは、CICS の日時サービスへのアクセスを制御します。

IccAbsTime は、絶対時刻 (1900 年の年初から経過したミリ秒単位の時間) に関する情報を保持します。これは、他の形式に日時に変換できます。**IccClock** オブジェクトと **IccAbsTime** オブジェクトで使用可能なメソッドは、非常によく似ています。

日時サービスの例

このサンプル・プログラムは、**IccClock** クラスの使用法を示しています。

このプログラムのソースと、このプログラムからの予期される出力は、ファイル `ICC$CLK` として [C++ サンプル・プログラム](#) にあります。このサンプルは、ここでは、端末の入出力要求を省略して示しています。

```
#include "icceh.hpp"
#include "iccmmain.hpp"
void IccUserControl::run()
{
```

最初の 2 行は、ファウンデーション・クラスのヘッダー・ファイルと、アプリケーション・プログラムの稼働環境をセットアップする標準 **main** 関数を組み込んでいます。

IccUserControl クラスの **run** メソッドに、この例のユーザー・コードが含まれています。

```
IccClock clock;
```

これで、`clock` オブジェクトが作成されます。

```
term->sendLine( "date() = [%s]",
clock.date() );
term->sendLine( "date(DDMMYY) = [%s]",
clock.date(IccClock::DDMMYY) );
term->sendLine( "date(DDMMYY,':') = [%s]",
clock.date(IccClock::DDMMYY,':') );
term->sendLine( "date(MMDDYY) = [%s]",
clock.date(IccClock::MMDDYY) );
term->sendLine( "date(YYDDD) = [%s]",
clock.date(IccClock::YYDDD));
```

ここでは、**date** メソッドを使用して、*format* 列挙型で指定されている形式で日付を返します。形式は順に、システム、DDMMYY、DD:MM:YY、MMDDYY、および YYDDD です。フィールド区切りに使用する文字は、*dateSeparator* 文字で指定されます (指定されていない場合、デフォルトは何も設定されません)。

```
term->sendLine( "daysSince1900() = %ld",
clock.daysSince1900());
term->sendLine( "dayOfWeek() = %d",
clock.dayOfWeek());
if ( clock.dayOfWeek() == IccClock::Friday )
term->sendLine( 40, "Today IS Friday" );
else
term->sendLine( 40, "Today is NOT Friday" );
```

このフラグメントは、**daysSince1900** および **dayOfWeek** メソッドの使用を示します。**dayOfWeek** は、曜日を示す列挙型を返します。その日が金曜日であれば、画面に「本日は金曜日です (Today IS Friday)」というメッセージが送信され、それ以外の場合は、「本日は金曜日ではありません (Today is NOT Friday)」のメッセージが送信されます。

```
term->sendLine( "dayOfMonth() = %d",
clock.dayOfMonth());
term->sendLine( "monthOfYear() = %d",
clock.monthOfYear());
```

これは、**IccClock** クラスの **dayOfMonth** メソッドと **monthOfYear** メソッドを示しています。

```
term->sendLine( "time() = [%s]",
clock.time() );
term->sendLine( "time('-') = [%s]",
clock.time('-') );
term->sendLine( "year() = [%ld]",
clock.year());
```

現在時刻は端末へ送信されますが、最初は分離文字を付けずに (つまり、HHMMSS 形式で) 送信され、次に桁を分離する「-」を付けて (つまり、HH-MM-SS 形式で) 送信されます。年が送信されます (例えば、1996)。

```
return;
};
```

run が終了し、制御が CICS に戻ります。

コンパイル、実行、およびデバッグ

このセクションでは、CICS ファウンデーション・クラス・プログラムをコンパイル、実行、およびデバッグする方法について説明します。

CICS ファウンデーション・クラス・プログラムのコンパイル

CICS ファウンデーション・クラス・プログラムをコンパイルし、リンクするには、プログラムのソース、コンパイラー、ヘッダー・ファイル、およびダイナミック・リンク・ライブラリーにアクセスする必要があります。

以下のアイテムにアクセスする必要があります。

- コンパイルするプログラムのソース

C++ プログラム・ソース・コードには、クラスファウンデーション・クラス・ヘッダーの **#include** ステートメント、およびファウンデーション・クラスの **main()** プログラム・スタブが必要です。

```
#include "icceh.hpp"
#include "iccmmain.hpp"
```

- IBM C++ コンパイラー
- ファウンデーション・クラスのヘッダー・ファイル ([『ヘッダー・ファイル』](#) を参照)
- ファウンデーション・クラスのダイナミック・リンク・ライブラリー (DLL)。ICCFCDLL モジュールは、CICSTS56.CICS.SDFHLOAD にあります。

ファウンデーション・クラスを使用するときは、コンパイルの前に「EXEC CICS」API を変換する必要はないことに注意してください。

以下の JOB ステートメントのサンプルでは、ICC\$HEL という名前のプログラムをコンパイル、プリリンク、およびリンクする方法を示しています。

```
//ICC$HEL JOB 1,user_name,MSGCLASS=A,CLASS=A,NOTIFY=userid
//PROCLIB JCLLIB ORDER=(
CICSTS56.CICS
.SDFHPROC)
//ICC$HEL EXEC ICCFCCL,INFILE=
indatasetname
(ICC$HEL),OUTFILE=
outdatasetname
(ICC$HEL)
//
```

ヘッダー・ファイル

ヘッダー・ファイルは、CICS C++ ファウンデーション・クラス・プログラムをコンパイルするために必要な C++ クラス定義です。

C++ ヘッダー・ファイル	このヘッダーに定義されているクラス
ICCABDEH	IccAbendData
ICCBASEH	IccBase
ICCBUFEH	IccBuf
ICCCLKEH	IccClock
ICCCNDEH	IccCondition (struct)
ICCCONEH	IccConsole
ICCCTLEH	IccControl
ICCDATEH	IccDataQueue
ICCEH	32 ページの『1』 を参照してください。
ICCEVTEH	IccEvent
ICCEXCEH	IccException
ICCFILEH	IccFile
ICCFLIEH	IccFileIterator
ICCGLBEH	Icc (struct) (グローバル関数)
ICCJRNEH	IccJournal
ICCMSGEH	IccMessage
ICCPRGEH	IccProgram
ICCRECEH	IccRecordIndex、IccKey、IccRBA、および IccRRN
ICCRESEH	IccResource
ICCRIDEH	IccResourceId + サブクラス (IccConvId など)
ICCSEMEH	IccSemaphore
ICCSESEH	IccSession
ICCSRQEH	IccStartRequestQ
ICCSYSEH	IccSystem
ICCTIMEH	IccTime、IccAbsTime、IccTimeInterval、IccTimeOfDay

C++ ヘッダー・ファイル	このヘッダーに定義されているクラス
ICCTMDEH	IccTerminalData
ICCTMPEH	IccTempStore
ICCTRMEH	IccTerminal
ICCTSKEH	IccTask
ICCUSREH	IccUser
ICCVALEH	IccValue (struct)

注:

1. リストされているすべてのヘッダー・ファイルを含む単一のヘッダーは、ICCEH という名前です。
2. ICCMAIN ファイルには、C++ ヘッダー・ファイルも提供されます。これには、ファウンデーション・クラス・プログラムの作成時に使用する **main** 関数スタブが含まれています。
3. ヘッダー・ファイルは、CICSTS56.CICS.SDFHC370 にあります。

プログラムの実行

コンパイルされリンクされた (つまり、実行可能な) ファウンデーション・クラス・プログラムを実行するには、以下を実行する必要があります。

1. 実行可能プログラムを CICS で使用できるようにします。このとき、プログラムが適切なディレクトリまたはロード・ライブラリー内にあることも確認します。また、サーバーによっては、CICS プログラム定義を作成してからでないと (CICS リソース定義機能を使用)、プログラムを実行できない場合があります。
2. CICS 端末にログオンします。
3. プログラムを実行します。

プログラムのデバッグ

ファウンデーション・クラス・プログラムを正常にコンパイル、リンク、および実行試行した後、デバッグが必要な場合があります。

CICS ファウンデーション・クラス・プログラムをデバッグするために使用できるオプションは、以下のよう
に 3 つあります。

- シンボリック・デバッガーを使用する
- トレースをアクティブにしてファウンデーション・クラス・プログラムを実行する
- CICS 実行診断機能を使用してファウンデーション・クラス・プログラムを実行する

シンボリック・デバッガー

シンボリック・デバッガーを使用すると、CICS ファウンデーション・クラス・プログラムのソースをステップスルー・デバッグできます。デバッグ・ツールは、IBM C/C++ の機能として組み込まれています。シンボリック・デバッガーを使用して CICS ファウンデーション・クラス・プログラムをデバッグするには、デバッグ情報を実行可能プログラムに追加するフラグを指定してプログラムをコンパイルします。CICS Transaction Server for z/OS の場合、このフラグは TEST(ALL) です。

詳しくは、[Debug Tool for z/OS](#) を参照してください。

トレース

デバッグ目的でトレース・ファイルを記述するように、CICS ファウンデーション・クラスを構成できます。

例外のトレースは常にアクティブです。CETR トランザクションは、すべての CICS プログラム (C++ クラスを使用して開発されたプログラムを含む) の補助トレースおよび内部トレースを制御します。

実行診断機能

実行診断機能 (EDF) を使用すると、各 **EXEC CICS** 呼び出しで停止しながら、CICS プログラムをステップスルーできます。表示画面には、CICS ファウンデーション・クラス・タイプのインターフェースではなく、**EXEC CICS** プロシージャ呼び出しインターフェースが表示されます。

EDF を有効にするには、ソース・コード内でプリプロセッサ・マクロ `ICC_EDF` を使用してから、`ICCMAIN` ファイルを組み込みます。

```
#define ICC_EDF //switch EDF on
#include "iccmain.hpp"
```

または、コンパイラ CPARM で適切なフラグを使用して、`ICC_EDF` を宣言します。

条件、エラー、および例外

このセクションでは、検出される可能性のあるさまざまなエラー状態に対応するために、ファウンデーション・クラスがどのように設計されているかについて説明します。

ファウンデーション・クラスの異常終了コード

重大なエラー (オブジェクトを作成するためのストレージが不足している場合など) が発生した場合は、ファウンデーション・クラスが CICS タスクを即時に終了します。

CICS ファウンデーション・クラスの異常終了コードはすべて、`ACLx` という形式です。アプリケーションが終了し、「`ACL`」で始まる異常終了コードが出された場合は、『[CICS メッセージ](#)』を参照してください。

C++ 例外およびファウンデーション・クラス

C++ 例外は、予約語 **try**、**throw**、および **catch** を使用して管理されます。

詳細については、コンパイラの資料または参考文献に記載されているいずれかの C++ 資料を参照してください。

以下に、サンプル `ICC$EXC1` を示します ([C++ サンプル・プログラム](#)を参照)。

```
#include "icceh.hpp"
#include "iccmain.hpp"
class Test {
public:
void tryNumber( short num ) {
IccTerminal* term = IccTerminal::instance();
*term << "Number passed = " << num << endl <<
flush;
if ( num > 10 ) {
*term << ">>Out of Range - throwing exception" << endl
<< flush;
throw "!!Number is out of range!!";
}
}
};
```

最初の 2 行は、ファウンデーション・クラスのヘッダー・ファイルと、アプリケーション・プログラムの稼働環境をセットアップする標準 **main** 関数を組み込んでいます。

クラス **Test** を宣言します。これには、**public** メソッド **tryNumber** が 1 つ含まれています。このメソッドは、10 より大きい整数が渡された場合に例外がスローされるように、インラインで実装されています。また、情報を CICS 端末に書き込みます。

```

void IccUserControl::run()
{
    IccTerminal* term = IccTerminal::instance();
    term->erase();
    *term << "This is program 'icc$exc1' ..." << endl;
    try {
        Test test;
        test.tryNumber( 1 );
        test.tryNumber( 7 );
        test.tryNumber( 11 );
        test.tryNumber( 6 );
    }
    catch( const char* exception ) {
        term->setLine( 22 );
        *term << "Exception caught: " << exception << endl;
        << flush;
    }
    term->send( 24,1,"Program 'icc$exc1' complete: Hit PF12 to End" );
    term->waitForAID( IccTerminal::PF12 );
    term->erase();
    return;
}

```

IccUserControl クラスの **run** メソッドに、この例のユーザー・コードが含まれています。

端末表示を消去し、テキストを書き込んだ後、**try** ブロックを開始します。**try** ブロックは、任意の行数の C++ コードに有効範囲を設定できます。

ここで、**Test** オブジェクトを作成し、唯一のメソッドである **tryNumber** を、さまざまなパラメーターを指定して呼び出します。最初の 2 回の呼び出し (1、7) は成功したものの、3 回目 (11) では **tryNumber** によって例外がスローされます。例外によって、プログラムの実行フローが現在の **try** ブロックから外れるため、4 番目の **tryNumber** の呼び出し (6) は実行されません。

その後、**try** ブロックから離れ、適切な **catch** ブロックを検索します。適切な **catch** ブロックは、スローされている例外のタイプと互換性のある引数を含むブロック (ここでは **char***) です。**catch** ブロックによって、CICS 端末にメッセージが書き込まれ、**catch** ブロックの後の行で実行が再開されます。

この CICS プログラムの出力は以下のとおりです。

```

This is program 'icc$exc1' ...
      Number passed = 1
      Number passed = 7
      Number passed = 11
      >>Out of Range - throwing exception
      Exception caught: !!Number is out of range!!
      Program 'icc$exc1' complete: Hit PF12 to End

```

前のサンプルのように、CICS C++ ファウンデーション・クラスは **char*** 例外をスローしませんが、代わりに **IccException** オブジェクトをスローします。

IccException にはいくつかのタイプがあります。**type** メソッドは、タイプを示す列挙型を返します。各タイプについて以下で説明します。

objectCreationError

オブジェクトを作成する試みが無効でした。これは例えば、**IccTask** などの singleton クラスの 2 番目のインスタンスを作成しようとした場合に発生します。

invalidArgument

無効な引数を指定してメソッドが呼び出されました。これは例えば、含まれているデータが多すぎる **IccBuf** オブジェクトがアプリケーション・プログラムによって **IccTempStore** クラスの **writeItem** メソッドに渡される場合に発生します。

また、長すぎるストリングを使用して **IccResourceId** のサブクラス (例えば、**IccTermId**) を作成しようとした場合にも発生します。

次のサンプルは、C++ サンプル・プログラムにあります (ファイル ICC\$EXC2)。ここに示すサンプルでは、端末 IO 要求の多くを除いてあります。

```

#include "icceh.hpp"
#include "iccmmain.hpp"
void IccUserControl::run()
{
    try
    {
        IccTermId id1( "1234" );
        IccTermId id2( "12345");
    }
    catch( IccException& exception )
    {
        terminal()->send( 21, 1, exception.summary() );
    }
    return;
}

```

前の例では、最初の **IccTermId** オブジェクトは正常に作成されますが、2 番目のオブジェクトでは、4 バイトしか許可されないのに 5 バイトのストリング "12345" が使用されているため、**IccException** がスローされています。このサンプル・プログラムからの予期される出力については、[C++ サンプル・プログラム](#)を参照してください。

invalidMethodCall

メソッドを呼び出すことはできません。これは一般的に、オブジェクトが現在の状態では呼び出しを受け入れられないことが原因です。例えば、読み取る **レコードを指定** するための **IccRecordIndex** オブジェクトが既にファイルに関連付けられている場合は、**IccFile** オブジェクト上の **readRecord** 呼び出しのみが受け入れられます。

CICSCondition

IccCondition 構造体にリストされている CICS 条件がオブジェクトで発生しました。このオブジェクトは例外をスローするように構成されていました。

familyConformanceError

このプログラムではファミリー・サブセットの制約が有効になっており、サポートされているすべてのプラットフォームで無効な操作が試行されました。

internalError

CICS ファウンデーション・クラスによって、内部エラーが検出されました。サービス担当者に連絡してください。

CICS 条件

CICS ファウンデーション・クラスは、アプリケーション実行時に発生する条件を処理するための高性能なフレームワークを提供します。

CICS リソースにアクセスすると、[ファウンデーション・クラス・リファレンス](#)に記載されているように、多数の CICS 条件が発生する可能性があります。

条件は、呼び出し側アプリケーションに返されるエラーまたは情報を表します。決定要因は、多くの場合、条件が発生したコンテキストです。

アプリケーション・プログラムは、さまざまな方法で CICS 条件を処理できます。各 CICS リソース・オブジェクト (プログラム、ファイル、データ・キューなど) は、必要に応じてさまざまな方法で CICS 条件を処理できます。

リソース・オブジェクトは、検出可能な各条件に対して、以下のいずれかのアクションを実行するように構成できます。

noAction

手動条件処理

callHandleEvent

自動条件処理

throwException

例外処理

abendTask

重大エラー処理

手動条件処理 (noAction)

これは、(任意のリソース・オブジェクト用の) すべての CICS 条件のデフォルトの処理です。

つまり、**condition** メソッドを使用して、条件を手動で処理する必要があるということです。以下に例を示します。

```
IccTempStore temp("TEMP1234");
IccBuf buf(40);
temp.setActionOnCondition(IccResource::noAction,
IccCondition::QIDERR);
buf = temp.readNextItem();
switch (temp.condition())
{
case IccCondition::QIDERR:
//do whatever here
:
default:
//do something else here
}
```

自動条件処理 (callHandleEvent)

以下のように、QIDERR など任意の CICS 条件に対してこれを活動化します。

```
IccTempStore temp("TEMP1234");
temp.setActionOnCondition(IccResource::callHandleEvent,
IccCondition::QIDERR);
```

オブジェクト「temp」の任意のメソッドに対する呼び出しにより、CICS で QIDERR 状態が発生した場合、**handleEvent** メソッドが自動的に呼び出されます。**handleEvent** メソッドは仮想メソッドにすぎないため、オブジェクトが **IccTempStore** のサブクラスに属していて **handleEvent** メソッドがオーバーライドされている場合のみ、この呼び出しが役立ちます。

IccTempStore のサブクラスを作成し、コンストラクターを宣言して、**handleEvent** メソッドをオーバーライドします。

```
class MyTempStore : public IccTempStore
{
public:
MyTempStore(const char* storeName) : IccTempStore(storeName) {}
HandleEventReturnOpt handleEvent(IccEvent& event);
};
```

ここで、**handleEvent** メソッドを実装します。

```
IccResource::HandleEventReturnOpt
MyTempStore::handleEvent(IccEvent& event)
{
switch (event.condition())
{
case ...
:
case IccCondition::QIDERR:
//Handle QIDERR condition here.
:
//
default:
return rAbendTask;
}
}
```

このコードは、特定の CICS 条件の「callHandleEvent」に対して構成されている任意の **MyTempStore** オブジェクトで呼び出されます。

例外処理 (throwException)

以下のように、QIDERR など任意の CICS 条件に対してこれを活動化します。

```
IccTempStore temp("TEMP1234");
temp.setActionOnCondition(IccResource::throwException,
IccCondition::QIDERR);
```

例外処理は、**try**、**throw**、および **catch** を使用する C++ 例外処理モデルによるものです。以下に例を示します。

```
try
{
    buf = temp.readNextItem();
    ...
}
catch (IccException& exception)
{
    //Exception handling code
    ...
}
```

try ブロックの内部にあるメソッドのいずれかでオブジェクト「temp」の QIDERR 状態を発生させた場合に例外がスローされます。例外がスローされると、C++ では、スタックをアンwindし、適切な **catch** ブロックで実行を再開します。try ブロック内で再開することはできません。さらに詳細な例については、サンプル ICC\$EXC3 を参照してください。

注：この例以外に多くの理由で、ファウンデーション・クラスから例外がスローされることがあります。詳しくは、[33 ページの『C++ 例外およびファウンデーション・クラス』](#)を参照してください。

重大エラー処理 (abendTask)

このオプションにより、CICS では、特定の条件が発生したときにタスクを終了することができます。

以下のように、QIDERR など任意の CICS 条件に対してこれを活動化します。

```
IccTempStore temp("TEMP1234");
temp.setActionOnCondition(IccResource::abendTask,
IccCondition::QIDERR);
```

CICS がオブジェクト「temp」に対して QIDERR 状態を発生させた場合、CICS タスクが ACL3 異常終了で終了します。

プラットフォームの相違

CICS ファウンデーション・クラスは、以下で説明するように、それが実行される特定の CICS プラットフォームから独立するように設計されています。ただし、プラットフォームによっていくつかの相違点があります。これらの相違点、およびそれぞれのコピー方法について、以下で説明します。

注：このセクションでは、詳しく説明するために、他の CICS プラットフォームに言及しています。それらのプラットフォームにおける CICS ファウンデーション・クラスのテクノロジー・リリースがありました。

アプリケーションは以下のいずれかのモードで実行できます。

fsAllowPlatformVariance

CICS ファウンデーション・クラスを使用して記述されたアプリケーションは、ターゲット CICS サーバーで使用可能なすべての機能にアクセスできます。

fsEnforce

アプリケーションが使用できる機能は、すべての CICS サーバー (z/OS と UNIX) 間で使用できる CICS 機能に制限されます。

デフォルトではプラットフォームの差異が認められますが、代わりにすべての CICS プラットフォームに共通の機能のみをアプリケーションで使用するよう強制することもできます。

クラス・ヘッダーはすべてのプラットフォームで同じで、あらゆる CICS プラットフォーム上のファウンデーション・クラスで使用可能なすべての CICS 機能を「サポート」(つまり定義) します。各プラットフォームの制約事項については、『[ファウンデーション・クラス・リファレンス](#)』で説明しています。プラットフォームの差異は以下のレベルで存在します。

- オブジェクト・レベル
- メソッド・レベル
- パラメーター・レベル

オブジェクト・レベル

一部のオブジェクトは特定のプラットフォームでサポートされません。

例えば、**IccConsole** オブジェクトは、CICS(r) for AIX® でコンソール・サービスをサポートしていない場合、CICS(r) for AIX(r) では作成できません。

CICS(r) for AIX(r) で **IccConsole** オブジェクトを作成しようとすると、タイプ「platformError」の **IccException** オブジェクトがスローされますが、他のプラットフォームでは許容されます。

```
IccConsole* cons = console(); //No good on CICS for AIX
```

「fsEnforce」を選択してアプリケーションを初期化した場合 ([initializeEnvironment](#) を参照)、前のどちらの例でも、タイプ「familyConformanceError」の **IccException** オブジェクトがすべてのプラットフォームでスローされます。

IccConsole クラスや **IccJournal** クラスのオブジェクトとは異なり、ほとんどのオブジェクトはどの CICS サーバー・プラットフォーム上でも作成できます。ただし、メソッドの使用が制限される場合があります。[ファウンデーション・クラス・リファレンス](#)に、プラットフォームのすべての制約事項が記載されています。

メソッド・レベル

あるプラットフォームで正常に実行されるメソッドが、別のプラットフォームでは問題を引き起こす場合があります。

例えば、**IccControl** クラスのメソッド **programId** について考えてみます。

```
void IccUserControl::run()
{
    if (strcmp(programId.name(), "PROG1234") == 0)
        //do something
}
```

ここで、メソッド **programId** は CICS TS for z/OS では正しく実行されますが、CICS(r) for AIX(r) ではタイプ「platformError」の **IccException** オブジェクトがスローされます。

また、ファミリー・サブセットの適用によってアプリケーションを初期化した場合 (**Icc** 構造体の **initializeEnvironment** 機能を参照)、どのような CICS サーバー・プラットフォームでも、メソッド **programId** でタイプ「familyConformanceError」の **IccException** オブジェクトをスローします。

パラメーター・レベル

このレベルでは、メソッドはすべてのプラットフォームでサポートされますが、特定の定位置パラメーターはプラットフォームの制限を受けます。

IccTask クラスのメソッド **abend** について考えます。

```
task()->abend();
1

task()->abend("WXYZ");
2

task()->abend("WXYZ", IccTask::respectAbendHandler);
3

task()->abend("WXYZ", IccTask::ignoreAbendHandler);
4

task()->abend("WXYZ", IccTask::ignoreAbendHandler,
5
IccTask::suppressDump);
```

異常終了 1 から 4 は、すべての CICS サーバー・プラットフォームで正常に実行されます。

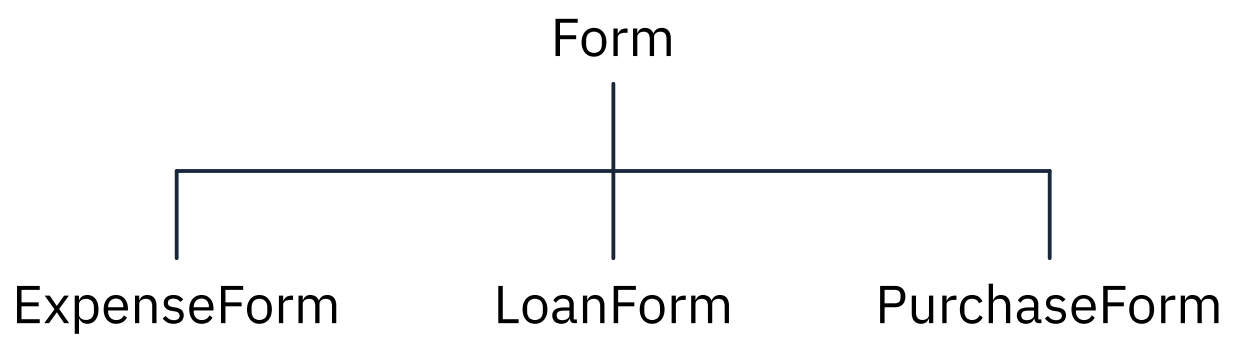
ファミリー・サブセットの適用がオフの場合は、CICS(r) for AIX(r) プラットフォームでは **abend** 5 がタイプ「platformError」の **IccException** オブジェクトをスローしますが、CICS Transaction Server for z/OS プラットフォームではスローしません。

ファミリー・サブセットの適用がオンの場合は、ターゲット CICS プラットフォームにかかわらず、abend
5 がタイプ「familyConformanceError」の **IccException** オブジェクトをスローします。

ポリモフィック動作

ポリモフィズム (ポリ = 多数、モアフィ = フォーム) は、多くの異なるフォームのオブジェクトを同じフォームとして扱う機能です。

ポリモフィズムは、継承および仮想関数を使用して C++ で実現します。一般的な Form を特殊化した 3 つのフォーム (ExpenseForm、LoanForm、PurchaseForm) について考えてみましょう。



各フォームは、何らかの時点で印刷する必要があります。プロシージャ型プログラミングでは、`print` 関数をコーディングして異なる 3 つのフォームを処理するか、または 3 つの異なる関数 (`printExpenseForm`、`printLoanForm`、`printPurchaseForm`) を記述します。

C++ では、以下のように、はるかに簡潔にこれを行えます。

```
class Form {
public:
    virtual void print();
};
class ExpenseForm : public Form {
public:
    virtual void print();
};
class LoanForm : public Form {
public:
    virtual void print();
};
class PurchaseForm : public Form {
public:
    virtual void print();
};
```

これらの指定変更された各関数は、各フォームが正しく印刷されるように実装されます。この時点で、フォーム・オブジェクトを使用するアプリケーションでは、以下を実行できます。

```
Form* pForm[10]
//create Expense/Loan/Purchase Forms...
for (short i=0 ; i < 9 ; i++)
    pForm->print();
```

ここで、Expense、Loan、および Purchase Form を組み合わせて 10 個のオブジェクトを作成します。ただし、基本クラス **Form** のポインターを扱うため、所有しているフォーム・オブジェクトの種類を認識する必要はありません。正しい **print** メソッドが自動的に呼び出されます。

ファウンデーション・クラスでは、制限付きのポリモフィック動作を使用できます。基本クラス **IccResource** には、以下の 3 つの仮想関数が定義されています。

```
virtual void clear();
virtual const IccBuf& get();
virtual void put(const IccBuf&
    buffer
);
```

これらのメソッドは、可能な場合は **IccResource** のサブクラス内に実装されています。

クラス	clear	get	put
IccConsole	×	×	✓
IccDataQueue	✓	✓	✓
IccJournal	×	×	✓
IccSession	×	✓	✓
IccTempStore	✓	✓	✓
IccTerminal	✓	✓	✓

これらの仮想メソッドは、テーブル内のメソッドを除き、**IccResource** のサブクラスでは サポートされていません。

注：基本クラス **IccResource** 内の **clear**、**get**、および **put** のデフォルト実装から、例外がスローされるため、ユーザーはサポートされていないメソッドを呼び出さずに済みます。

ポリモアフィック動作の例

以下のサンプルは、ICC\$RES2 ファイルとして samples ディレクトリーにあります。

ここでは、端末の入出力要求を省略して示しています。[C++ サンプル・プログラム](#)を参照してください。

```
#include "icceh.hpp"
#include "iccmain.hpp"
char* dataItems[] =
{
    "Hello World - item 1",
    "Hello World - item 2",
    "Hello World - item 3"
};
void IccUserControl::run()
{
```

ここでは、ファウンデーション・クラスのヘッダーと **main** 関数を組み込みます。**dataItems** には、いくつかのサンプル・データ項目が含まれています。**IccUserControl** クラスの **run** メソッドのアプリケーション・コードを作成します。

```
    IccBuf buffer( 50 );
    IccResource* pObj[2];
```

データ項目を保持するために、**IccBuf** オブジェクト (最初に 50 バイト) を作成します。**IccResource** オブジェクトへの 2 つのポインターの配列を宣言します。

```
    pObj[0] = new IccDataQueue("ICQ");
    pObj[1] = new IccTempStore("ICTEMP");
```

IccResource から派生するクラスを持つ 2 つのオブジェクト、**IccDataQueue** と **IccTempStore** を作成します。

```
    for ( short index=0; index <= 1 ; index++ )
    {
        pObj[index]->clear();
    }
```

両方のオブジェクトに対して、**clear** メソッドを呼び出します。これは、アプリケーション・プログラムに対して透過的な方法で、オブジェクト別にそれぞれ処理されます。これがポリモアフィック動作です。

```
    for ( index=0; index <= 1 ; index++ )
    {
        for (short j=1 ; j <= 3 ; j++)
        {
            buffer = dataItems[j-1];
            pObj[index]->put( buffer );
        }
    }
```

ここで、各リソース・オブジェクトに 3 つのデータ項目を **put** します。さらに、**put** メソッドがオブジェクト・タイプに適切な方法で要求に応答します。

```
    for ( index=0; index <= 1 ; index++ )
    {
        buffer = pObj[index]->get();
        while (pObj[index]->condition() == IccCondition::NORMAL)
        {
            buffer = pObj[index]->get();
        }
        delete pObj[index];
    }
    return;
}
```

get メソッドを使用して、各リソース・オブジェクトからデータ項目を再読み込みします。リソース・オブジェクトを削除し、CICS に制御を戻します。

ストレージ管理

C++ オブジェクトは通常、スタックまたはヒープに保管されます。

スタック上のオブジェクトは、有効範囲を超えると、自動的に破棄されますが、ヒープ上のオブジェクトは破棄されません。

CICS ファウンデーション・クラスによって内部的に作成されるオブジェクトの多くは、スタック上ではなくヒープ上に作成されます。これにより、一部の CICS サーバー環境では問題が発生することがあります。

CICS Transaction Server for z/OS では、CICS および Language Environment® によって、すべてのタスク・ストレージが管理されるため、タスク終了 (正常終了または異常終了) 時にストレージが解放されます。

CICS for AIX 環境では、ヒープに割り振られたストレージは、タスク終了時に自動的に解放されません。これは、アプリケーション・プログラマーがヒープ上のオブジェクトを明示的に削除することを忘れた場合や、さらに深刻な状況として、タスクが異常終了した場合などに、「メモリー・リーク」につながるおそれがあります。

この問題は、CICS ファウンデーション・クラスでは解消されています。基本ファウンデーション・クラス **IccBase** で、演算子 **new** および **delete** を指定します。これらを構成するには、動的ストレージ割り振り要求を CICS タスク・ストレージにマップします。これにより、タスク終了時に **すべて** のストレージが自動的に解放されます。この方法の欠点は、パフォーマンス上の問題です。ファウンデーション・クラスは一般的に、単一の大規模な割り振り要求ではなく、小規模のストレージ割り振り要求を大量に発行します。

この機能は、ファウンデーション・クラスの使用前に発行する必要がある **Icc::initializeEnvironment** 呼び出しの影響を受けます。(この関数は、デフォルトの **main** 関数から呼び出されます。[CICS C++ main 関数](#) を参照してください。)

initializeEnvironment 関数に渡される最初のパラメーターは、以下の 3 つの値のいずれかを取る列挙です。

cmmDefault

デフォルト・アクションは、以下のようにプラットフォームに依存します。

z/OS

「**cmmNonCICS**」と同じです (『**cmmNonCICS**』セクションを参照)。

UNIX

「**cmmCICS**」と同じです (『**cmmCICS**』セクションを参照)。

cmmNonCICS

IccBase クラス内の **new** 演算子および **delete** 演算子は、動的ストレージ割り振り要求を CICS タスク・ストレージにマップしません。代わりに、C++ のデフォルトの **new** 演算子と **delete** 演算子が呼び出されます。

cmmCICS

IccBase クラス内の **new** 演算子および **delete** 演算子は、動的ストレージ割り振り要求を CICS タスク・ストレージ (タスクの正常終了または異常終了時に自動的に解放される) にマップします。

ファウンデーション・クラスに提供されるデフォルトの **main** 関数は、「**cmmDefault**」の enum を使用して **initializeEnvironment** を呼び出します。これは、プログラム内で以下のように変更できます。このとき、提供された「ヘッダー・ファイル」**ICCMAN** を変更する必要はありません。

```
#define ICC_CLASS_MEMORY_MGMT Icc::cmmNonCICS
#include "iccmmain.hpp"
```

または、コンパイル時に **DEV(ICC_CLASS_MEMORY_MGMT)** オプションを設定します。

パラメーター受け渡し規則

ファウンデーション・クラス・メソッド呼び出しでオブジェクトの受け渡しに使用される規則は、オブジェクトが必須である場合は参照による受け渡し、オブジェクトがオプションである場合はポインターによる受け渡しです。

例えば、**IccStartRequestQ** クラスのメソッド **start** を確認してみましょう。このメソッドには、以下のシグニチャーがあります。

```
const IccRequestId& start( const IccTransId&
transId,
const IccTime* time=0,
const IccRequestId* reqId=0 );
```

上記の規則から、**IccTransId** オブジェクトが必須である一方、**IccTime** オブジェクトと **IccRequestId** オブジェクトはどちらもオプションであることがわかります。これによりアプリケーションは、以下のいずれかの方法でこのメソッドを使用できます。

```
IccTransId trn("ABCD");
IccTimeInterval int(0,0,5);
IccRequestId req("MYREQ");
IccStartRequestQ* startQ = startRequestQ();
startQ->start( trn );
startQ->start( trn, &int );
startQ->start( trn, &int, &req );
startQ->start( trn, 0, &req );
```

「read」メソッドから返される IccBuf 参照内のデータの有効範囲

IccResource の多くのサブクラスには、**const IccBuf** 参照を返す「read」メソッド (例えば、**IccFile::readRecord**、**IccTempStore::readItem**、および **IccTerminal::receive**) があります。

IccBuf 参照から独自の **IccBuf** オブジェクトへデータをコピーするのではなく、**IccBuf** オブジェクトの参照を保持する場合は、注意が必要です。例えば、以下について考えてみます。

```
IccBuf buf(50);
IccTempStore store("TEMPSTOR");
buf = store.readNextItem();
```

ここでは、**IccTempStore::readNextItem** から返された **IccBuf** 参照内のデータが、アプリケーション独自の **IccBuf** オブジェクトに **即時** にコピーされるため、データが後から無効になっても影響はありません。ただし、アプリケーションは次のようになる可能性があります。

```
IccTempStore store("TEMPSTOR");
const IccBuf& buf = store.readNextItem();
```

ここでは、**IccTempStore::readNextItem** から返された **IccBuf** 参照が、アプリケーション独自のストレージに **コピーされない** ため、注意が必要です。

注：有効なデータが含まれていない **IccBuf** オブジェクトの参照の使用を回避するために、このスタイルのプログラミングを使用することは、お勧めしません。

返された **IccBuf** 参照には一般的に有効なデータが含まれていますが、以下のいずれかの条件に該当する場合は除きます。

- **IccResource** オブジェクトで別の「read」メソッド (上記の例では、別の **readNextItem** または **readItem** メソッド) が呼び出される。
- リソース更新がコミットされる (メソッド **IccTask::commitUOW** を参照)。
- タスクが (正常または異常) 終了する。

第3章 ファウンデーション・クラス: 参照

このセクションでは、CICS の一部として提供されるファウンデーション・クラスおよび構造の参照情報を示します。クラスと構造をアルファベット順に記載します。これらのクラスと構造には、オブジェクト指向 CICS プログラムを作成するために必要なすべての機能が組み込まれています。

すべてのクラスおよび構造が、固有の接頭部 **Icc** で始まっています。この接頭部を使用して、独自のクラスを作成しないでください。

Icc 構造には、広範囲に適用できる機能と列挙が含まれています。**IccValue** 構造は、従来の CICS プログラムで使用されていたすべての CVDA 値の列挙で構成されています。

各クラスの説明は、**IccBase** クラス (他のすべてのクラスの基礎) からの派生方法を示した単純な図で始まっています。この後に、簡単な説明と、そのクラスが含まれているヘッダー・ファイルの名前が示されています。また、そのクラスを使用するサンプル・ソース・ファイルがある場合は、併せて示してあります。

各クラスまたは構造の説明には、以下のセクションがあります (ただし、該当する場合)。

1. 継承図
2. クラスの要旨
3. クラスが定義されているヘッダー・ファイル。ご使用のシステム上での C++ ヘッダー・ファイルの場所については、[ヘッダー・ファイルを参照してください](#)。
4. クラスを示すサンプル・プログラム。ご使用のシステムで提供される C++ サンプル・プログラムの場所については、『[C++ サンプル・プログラム](#)』を参照してください。
5. Icc... コンストラクター
6. public メソッド (アルファベット順)
7. protected メソッド (アルファベット順)
8. 継承された public メソッド (表形式)
9. 継承された protected メソッド (表形式)
10. 列挙

各メソッド (コンストラクターを含む) には、最初に正式な関数プロトタイプを示してあります。このプロトタイプは、呼び出しによって返される内容、およびパラメーターの内容を示してあります。その後で、パラメーターについて順に説明しています。重複を避けるために、継承されたメソッドは、それが派生したクラス (およびそれらの説明箇所) を示してあります。

名前の規則は次のとおりです。

1. 変数名は *variable* と示してあります。
2. クラスの名前、構造、列挙、およびメソッドは **method** と示してあります。
3. 列挙のメンバーは「enumMember」と示してあります。
4. 提供されているすべてのクラスおよび構造の名前は **Icc** で始まります。
5. 複合名には分離文字は含まれていませんが、2 番目以降の語の先頭は大文字になっています (例えば、**IccJournalTypeId**)。
6. クラスと構造の名前、および列挙型は、大文字で始まります。その他の名前は小文字で始まります。

これらのクラスの使用法の詳細については、『[CICS ファウンデーション・クラスの使用法](#)』を参照してください。

EXEC CICS 呼び出しとファウンデーション・クラス・メソッドのマッピング

以下の表は、EXEC CICS API を使用して行われる CICS 呼び出しと、ファウンデーション・クラスからの同等の呼び出しの間の対応を示しています。

EXEC CICS	クラス	メソッド
ABEND	IccTask	異常終了
ADDRESS COMMAREA	IccControl	commArea
ADDRESS CWA	IccSystem	workArea
ADDRESS EIB	EIB への直接アクセスはありません。適切なクラスに適切なメソッドを使用してください。	
ADDRESS TCTUA	IccTerminal	workArea
ADDRESS TWA	IccTask	workArea
ALLOCATE	IccSession	allocate
ASKTIME	IccClock	更新
ASSIGN ABCODE	IccAbendData	abendCode
ASSIGN ABDUMP	IccAbendData	isDumpAvaliable
ASSIGN ABPROGRAM	IccAbendData	programName
ASSIGN ALTSCRNHT	IccTerminalData	alternateHeight
ASSIGN ALTSCRNWD	IccTerminalData	alternateWidth
ASSIGN APLKYBD	IccTerminalData	isAPLKeyboard
ASSIGN APLTEXT	IccTerminalData	isAPLText
ASSIGN ASRAINTRPT	IccAbendData	ASRAInterrupt
ASSIGN ASRAKEY	IccAbendData	ASRAKeyType
ASSIGN ASRAPSW	IccAbendData	ASRAPSW
ASSIGN ASRAREGS	IccAbendData	ASRARegisters
ASSIGN ASRASPC	IccAbendData	ASRASpaceType
ASSIGN ASRASTG	IccAbendData	ASRAStorageType
ASSIGN APPLID	IccSystem	applName
ASSIGN BTRANS	IccTerminalData	isBTrans
ASSIGN CMDSEC	IccTask	isCommandSecurityOn
ASSIGN COLOR	IccTerminalData	isColor
ASSIGN CWALENG	IccSystem	workArea
ASSIGN DEFSCRNHT	IccTerminalData	defaultHeight
ASSIGN DEFSCRNWD	IccTerminalData	defaultWidth
ASSIGN EWASUPP	IccTerminalData	isEWA
ASSIGN EXTDS	IccTerminalData	isExtended3270
ASSIGN FACILITY	IccTerminal	name
ASSIGN FCI	IccTask	facilityType

EXEC CICS	クラス	メソッド
ASSIGN GCHARS	IccTerminalData	graphicCharSetId
ASSIGN GCODES	IccTerminalData	graphicCharCodeSet
ASSIGN GMMI	IccTerminalData	isGoodMorning
ASSIGN HILIGHT	IccTerminalData	isHighlight
ASSIGN INITPARM	IccControl	initData
ASSIGN INITPARMLEN	IccControl	initData
ASSIGN INVOKINGPROG	IccControl	callingProgramId
ASSIGN KATAKANA	IccTerminalData	isKatakana
ASSIGN NETNAME	IccTerminal	netName
ASSIGN OUTLINE	IccTerminalData	isFieldOutline
ASSIGN ORGABCODE	IccAbendData	originalAbendCode
ASSIGN PRINSYSID	IccTask	principalSysId
ASSIGN PROGRAM	IccControl	programId
ASSIGN PS	IccTerminalData	isPS
ASSIGN QNAME	IccTask	triggerDataQueueId
ASSIGN RESSEC	IccTask	isResourceSecurityOn
ASSIGN RESTART	IccTask	isRestarted
ASSIGN SCRNHHT	IccTerminal	height
ASSIGN SCRNWD	IccTerminal	width
ASSIGN SOSI	IccTerminalData	isSOSI
ASSIGN STARTCODE	IccTask	startType、 isCommitSupported、 isStartDataAvailable
ASSIGN SYSID	IccSystem	sysId
ASSIGN TASKPRIORITY	IccTask	priority
ASSIGN TCTUALENG	IccTerminal	workArea
ASSIGN TEXTKYBD	IccTerminalData	isTextKeyboard
ASSIGN TEXTPRINT	IccTerminalData	isTextPrint
ASSIGN TWALENG	IccTask	workArea
ASSIGN USERID	IccTask	userId
ASSIGN VALIDATION	IccTerminalData	isValidation
CANCEL	IccClock	cancelAlarm
CANCEL	IccStartRequestQ	cancel
CHANGE PASSWORD	IccUser	changePassword
CHANGE TASK	IccTask	setPriority
CONNECT PROCESS	IccSession	connectProcess
CONVERSE	IccSession	converse

EXEC CICS	クラス	メソッド
DELAY	IccTask	delay
DELETE	IccFile	deleteRecord
DELETE	IccFile	deleteLockedRecord
DELETEQ TD	IccDataQueue	empty
DELETEQ TS	IccTempStore	empty
DEQ	IccSemaphore	unlock
DUMP TRANSACTION	IccTask	dump
DUMP TRANSACTION	IccTask	setDumpOpts
ENDBR	IccFileIterator	IccFileIterator (destructor)
ENQ	IccSemaphore	lock
ENQ	IccSemaphore	tryLock
ENTER TRACENUM	IccTask	enterTrace
EXTRACT ATTRIBUTES	IccSession	state、stateText
EXTRACT PROCESS	IccSession	extractProcess
FORMATTIME YYDDD、YYMMDD など	IccClock	date
FORMATTIME DATE	IccClock	date
FORMATTIME DATEFORM	IccSystem	dateFormat
FORMATTIME DAYCOUNT	IccClock	daysSince1900
FORMATTIME DAYOFWEEK	IccClock	dayOfWeek
FORMATTIME DAYOFMONTH	IccClock	dayOfMonth
FORMATTIME MONTHOFYEAR	IccClock	monthOfYear
FORMATTIME TIME	IccClock	time
FORMATTIME YEAR	IccClock	year
FREE	IccSession	free
FREEMAIN	IccTask	freeStorage
GETMAIN	IccTask	getStorage
HANDLE ABEND	IccControl	setAbendHandler、 cancelAbendHandler、 resetAbendHandler
INQUIRE FILE ACCESSMETHOD	IccFile	accessMethod
INQUIRE FILE ADD	IccFile	isAddable
INQUIRE FILE BROWSE	IccFile	isBrowsable
INQUIRE FILE DELETE	IccFileControl	isDeletable
INQUIRE FILE EMPTYSTATUS	IccFile	isEmptyOn
INQUIRE FILE ENABLESTATUS	IccFile	enableStatus

EXEC CICS	クラス	メソッド
INQUIRE FILE KEYPOSITION	IccFile	keyPosition
INQUIRE FILE OPENSTATUS	IccFile	openStatus
INQUIRE FILE READ	IccFile	isReadable
INQUIRE FILE RECORDFORMAT	IccFile	recordFormat
INQUIRE FILE RECORDSIZE	IccFile	recordLength
INQUIRE FILE RECOVSTATUS	IccFile	isRecoverable
INQUIRE FILE TYPE	IccFile	type
INQUIRE FILE UPDATE	IccFile	isUpdatable
ISSUE ABEND	IccSession	issueAbend
ISSUE CONFIRMATION	IccSession	issueConfirmation
ISSUE ERROR	IccSession	issueError
ISSUE PREPARE	IccSession	issuePrepare
ISSUE SIGNAL	IccSession	issueSignal
LINK	IccProgram	link
LINK INPUTMSG INPUTMSGLEN	IccProgram	setInputMessage
LOAD	IccProgram	load
POST	IccClock	setAlarm
READ	IccFile	readRecord
READNEXT	IccFileIterator	readNextRecord
READPREV	IccFileIterator	readPreviousRecord
READQ TD	IccDataQueue	readItem
READQ TS	IccTempStore	readItem
RECEIVE (APPC)	IccSession	受信
RECEIVE (3270)	IccTerminal	receive、receive3270Data
RELEASE	IccProgram	unload
RESETBR	IccFileIterator	reset
RETRIEVE	IccStartRequestQ	retrieveData ¹

注 : **retrieveData** メソッドは、CICS から開始情報を取得し、それを IccStartRequestQ オブジェクトに保管します。その後、この情報には、**data**、**queueName**、**returnTermId**、および **returnTransId** の各メソッドを使用してアクセスすることができます。

RETRIEVE INTO, LENGTH	IccStartRequestQ	データ
RETRIEVE QUEUE	IccStartRequestQ	queueName
RETRIEVE RTRANSID	IccStartRequestQ	returnTransId
RETRIEVE RTERMID	IccStartRequestQ	returnTermId
RETURN	IccControl	main ²

EXEC CICS	クラス	メソッド
<p>注：(C++ 予約語 return) を使用してクラス IccControl のメソッド run から戻ると、EXEC CICS RETURN になります。</p>		
RETURN TRANSID	IccTerminal	setNextTransId ³
RETURN IMMEDIATE	IccTerminal	setNextTransId ³
RETURN COMMAREA LENGTH	IccTerminal	setNextCommArea ³
RETURN INPUTMSG, INPUTMSGLEN	IccTerminal	setNextInputMessage ³
<p>注：この呼び出しは、IccControl::run から戻る前に発行します。</p>		
REWRITE	IccFile	rewriteRecord
SEND (APPC)	IccSession	send、sendInvite、sendLast
SEND (3270)	IccTerminal	send、sendLine
SEND CONTROL CURSOR	IccTerminal	setCursor setLine、setNewLine
SEND CONTROL ERASE	IccTerminal	erase
SEND CONTROL FREEKB	IccTerminal	freeKeyboard
SET FILE ADD BROWSE DELETE ...	IccFile	setAccess
SET FILE EMPTYSTATUS	IccFile	setEmptyOnOpen
SET FILE OPEN STATUS ENABLESTATUS	IccFile	setStatus
SIGNOFF	IccTerminal	signoff
SIGNON	IccTerminal	signon
START TRANSID AT/AFTER	IccStartRequestQ	start ⁴
START TRANSID FROM LENGTH	IccStartRequestQ	setData、registerDataBuffer ⁴
START TRANSID NOCHECK	IccStartRequestQ	setStartOpts ⁴
START TRANSID PROTECT	IccStartRequestQ	setStartOpts ⁴
START TRANSID QUEUE	IccStartRequestQ	setQueueName ⁴
START TRANSID REQID	IccStartRequestQ	start ⁴
START TRANSID TERMID	IccStartRequestQ	start ⁴
START TRANSID USERID	IccStartRequestQ	start ⁴
START TRANSID RTERMID	IccStartRequestQ	setReturnTermId ⁴
START TRANSID RTRANSID	IccStartRequestQ	setReturnTransId ⁴
<p>注：setData、setQueueName、setReturnTermId、setReturnTransId、setStartOpts の各メソッドを使用して、start メソッドで開始要求を発行する前に IccStartRequestQ オブジェクトの状態を設定します。</p>		
STARTBR	IccFileIterator	IccFileIterator (constructor)
SUSPEND	IccTask	suspend

EXEC CICS	クラス	メソッド
SYNCPOINT	IccTask	commitUOW
SYNCPOINT ROLLBACK	IccTask	rollBackUOW
UNLOCK	IccFile	unlockRecord
VERIFY PASSWORD	IccUser	verifyPassword
WAIT CONVID	IccSession	flush
WAIT EVENT	IccTask	waitOnAlarm
WAIT EXTERNAL	IccTask	waitExternal
WAIT JOURNALNUM	IccJournal	wait
WRITE	IccFile	writeRecord
WRITE OPERATOR	IccConsole	write、writeAndGetReply
WRITEQ TD	IccDataQueue	writeItem
WRITEQ TS	IccTempStore	writeItem、rewriteItem

EXEC CICS 呼び出しへのファウンデーション・クラス・メソッドのマッピング

以下の表は、ファウンデーション・クラスを使用して行われる CICS 呼び出しと、同等の EXEC CICS API 呼び出しの間の対応を示しています。

表 1. IccAbendData クラス	
メソッド	EXEC CICS
abendCode	ASSIGN ABCODE
ASRAInterrupt	ASSIGN ASRAINTRPT
ASRAKeyType	ASSIGN ASRAKEY
ASRAPSW	ASSIGN ASRAPSW
ASRARegisters	ASSIGN ASRAREGS
ASRASpaceType	ASSIGN ASRASPC
ASRAStorageType	ASSIGN ASRASTG
isDumpAvailable	ASSIGN ABDUMP
originalAbendCode	ASSIGN ORGABCODE
programName	ASSIGN ABPROGRAM

表 2. IccAbsTime クラス	
メソッド	EXEC CICS
date	FORMATTIME YYDDD/YYMMDD/etc.
dayOfMonth	FORMATTIME DAYOFMONTH
dayOfWeek	FORMATTIME DAYOFWEEK
daysSince1900	FORMATTIME DAYCOUNT
monthOfYear	FORMATTIME MONTHOFYEAR

表 2. <i>IccAbsTime</i> クラス (続き)	
メソッド	EXEC CICS
time	FORMATTIME TIME
year	FORMATTIME YEAR

表 3. <i>IccClock</i> クラス	
メソッド	EXEC CICS
cancelAlarm	CANCEL
date	FORMATTIME YYDDD/YMMDD/etc.
dayOfMonth	FORMATTIME DAYOFMONTH
dayOfWeek	FORMATTIME DAYOFWEEK
daysSince1900	FORMATTIME DAYCOUNT
monthOfYear	FORMATTIME MONTHOFYEAR
setAlarm	POST
time	FORMATTIME TIME
update	ASKTIME
year	FORMATTIME YEAR

表 4. <i>IccConsole</i> クラス	
メソッド	EXEC CICS
write	WRITE OPERATOR
writeAndGetReply	WRITE OPERATOR

表 5. <i>IccControl</i> クラス	
メソッド	EXEC CICS
callingProgramId	ASSIGN INVOKINGPROG
cancelAbendHandler	HANDLE ABEND CANCEL
commArea	ADDRESS COMMAREA
initData	ASSIGN INITPARM & INITPARMLLEN
programId	ASSIGN PROGRAM
resetAbendHandler	HANDLE ABEND RESET
setAbendHandler	HANDLE ABEND PROGRAM

表 6. <i>IccDataQueue</i> クラス	
メソッド	EXEC CICS
empty	DELETEQ TD
readItem	READQ TD
writeItem	WRITEQ TD

表 7. IccFile クラス	
メソッド	EXEC CICS
access	INQUIRE FILE ADD BROWSE DELETE READ UPDATE
accessMethod	INQUIRE FILE ACCESSMETHOD
deleteRecord	DELETE FILE RIDFLD
deleteLockedRecord	DELETE FILE
enableStatus	INQUIRE FILE ENABLESTATUS
isAddable	INQUIRE FILE ADD
isBrowsable	INQUIRE FILE BROWSE
isDeletable	INQUIRE FILE DELETE
isEmptyOnOpen	INQUIRE FILE EMPTYSTATUS
isReadable	INQUIRE FILE READ
isRecoverable	INQUIRE FILE RECOVSTATUS
isUpdatable	INQUIRE FILE UPDATE
keyPosition	INQUIRE FILE KEYPOSITION
openStatus	INQUIRE FILE OPENSTATUS
readRecord	READ FILE
recordFormat	INQUIRE FILE RECORDFORMAT
recordLength	INQUIRE FILE RECORDSIZE
rewriteRecord	REWRITE FILE
setAccess	SET FILE ADD BROWSE DELETE etc.
setEmptyOnOpen	SET FILE EMPTYSTATUS
setStatus	SET FILE OPENSTATUS ENABLESTATUS
type	INQUIRE FILE TYPE
unlockRecord	UNLOCK FILE
writeRecord	WRITE FILE

表 8. IccFileIterator クラス	
メソッド	EXEC CICS
IccFileIterator (constructor)	STARTBR FILE
~IccFileIterator (destructor)	ENDBR FILE
readNextRecord	READNEXT FILE
readPreviousRecord	READPREV FILE
reset	RESETBR FILE

表 9. <i>IccJournal</i> クラス	
メソッド	EXEC CICS
wait	WAIT JOURNALNUM
writeRecord	WRITE JOURNALNUM

表 10. <i>IccProgram</i> クラス	
メソッド	EXEC CICS
link	LINK PROGRAM
load	LOAD PROGRAM
unload	RELEASE PROGRAM

表 11. <i>IccResource</i> クラス	
メソッド	EXEC CICS
condition	(RESP & RESP2)
setRouteOption	(SYSID)

表 12. <i>IccSemaphore</i> クラス	
メソッド	EXEC CICS
lock	ENQ RESOURCE
tryLock	ENQ RESOURCE NOSUSPEND
unlock	DEQ RESOURCE

表 13. <i>IccSession</i> クラス	
メソッド	EXEC CICS
allocate	ALLOCATE
connectProcess	CONNECT PROCESS CONVID
converse	CONVERSE CONVID
extractProcess	EXTRACT PROCESS CONVID
flush	WAIT CONVID
free	FREE CONVID
issueAbend	ISSUE ABEND CONVID
issueConfirmation	ISSUE CONFIRMATION CONVID
issueError	ISSUE ERROR CONVID
issuePrepare	ISSUE PREPARE CONVID
issueSignal	ISSUE SIGNAL CONVID
receive	RECEIVE CONVID
send	SEND CONVID
sendInvite	SEND CONVID INVITE

表 13. IccSession クラス (続き)	
メソッド	EXEC CICS
sendLast	SEND CONVID LAST
state	EXTRACT ATTRIBUTES

表 14. IccStartRequestQ クラス	
メソッド	EXEC CICS
cancel	CANCEL
retrieveData	RETRIEVE
start	START TRANSID

表 15. IccSystem クラス	
メソッド	EXEC CICS
applName	ASSIGN APPLID
beginBrowse	INQUIRE (FILE、TDQUEUE など) START
dateFormat	FORMATTIME DATEFORM
endBrowse	INQUIRE (FILE、TDQUEUE など) END
freeStorage	FREEMAIN
getFile	INQUIRE FILE
getNextFile	INQUIRE FILE NEXT
getStorage	GETMAIN SHARED
operatingSystem	INQUIRE SYSTEM OPSYS
operatingSystemLevel	INQUIRE SYSTEM OPREL
release	INQUIRE SYSTEM RELEASE
releaseText	INQUIRE SYSTEM RELEASE
sysId	ASSIGN SYSID
workArea	ADDRESS CWA

表 16. IccTask クラス	
メソッド	EXEC CICS
abend	ABEND
commitUOW	SYNCPOINT
delay	DELAY
dump	DUMP TRANSACTION
enterTrace	ENTER TRACENUM
facilityType	ASSIGN STARTCODE, TERMCODE, PRINSYSID, FCI
freeStorage	FREEMAIN
isCommandSecurityOn	ASSIGN CMDSEC

表 16. *IccTask* クラス (続き)

メソッド	EXEC CICS
isCommitSupported	ASSIGN STARTCODE
isResourceSecurityOn	ASSIGN RESSEC
isRestarted	ASSIGN RESTART
isStartDataAvailable	ASSIGN STARTCODE
principalSysId	ASSIGN PRINSYSID
priority	ASSIGN TASKPRIORITY
rollBackUOW	SYNCPOINT ROLLBACK
setPriority	CHANGE TASK PRIORITY
startType	ASSIGN STARTCODE
suspend	SUSPEND
triggerDataQueueId	ASSIGN QNAME
userId	ASSIGN USERID
waitExternal	WAIT EXTERNAL / WAITCICS
waitOnAlarm	WAIT EVENT
workArea	ADDRESS TWA

表 17. *IccTempStore* クラス

メソッド	EXEC CICS
empty	DELETEQ TS
readItem	READQ TS ITEM
readNextItem	READQ TS NEXT
rewriteItem	WRITEQ TS ITEM REWRITE
writeItem	WRITEQ TS ITEM

表 18. *IccTerminal* クラス

メソッド	EXEC CICS
erase	SEND CONTROL ERASE
freeKeyboard	SEND CONTROL FREEKB
height	ASSIGN SCRNHT
netName	ASSIGN NETNAME
receive	RECEIVE
receive3270Data	RECEIVE BUFFER
send	SEND
sendLine	SEND
setCursor	SEND CONTROL CURSOR

表 18. IccTerminal クラス (続き)

メソッド	EXEC CICS
setLine	SEND CONTROL CURSOR
setNewLine	SEND CONTROL CURSOR
signoff	SIGNOFF
signon	SIGNON
waitForAID	RECEIVE
width	ASSIGN SCRNEW
workArea	ADDRESS TCTUA

表 19. IccTerminalData クラス

メソッド	EXEC CICS
alternateHeight	ASSIGN ALTSCRNHT
alternateWidth	ASSIGN ALTSCRNEW
defaultHeight	ASSIGN DEFSCRNHT
defaultWidth	ASSIGN DEFSCRNEW
graphicCharSetId	ASSIGN GCHARS
graphicCharCodeSet	ASSIGN GCODES
isAPLKeyboard	ASSIGN APLKYBD
isAPLText	ASSIGN APLTEXT
isBTrans	ASSIGN BTRANS
isColor	ASSIGN COLOR
isEWA	ASSIGN ESASUPP
isExtended3270	ASSIGN EXTDS
isGoodMorning	ASSIGN GMMI
isHighlight	ASSIGN HILIGHT
isKatakana	ASSIGN KATAKANA
isMSRControl	ASSIGN MSRCONTROL
isFieldOutline	ASSIGN OUTLINE
isPS	ASSIGN PS
isSOSI	ASSIGN SOSI
isTextKeyboard	ASSIGN TEXTKYBD
isTextPrint	ASSIGN TEXTPRINT
isValidation	ASSIGN VALIDATION

表 20. <i>IccUser</i> クラス	
メソッド	EXEC CICS
changePassword	CHANGE PASSWORD
verifyPassword	VERIFY PASSWORD

Icc 構造

この構造は、CICS ファウンデーション・クラスのグローバル列挙型および関数を保持します。これらのグローバルは、名前の競合を回避するために、この構造内で定義されます。

ヘッダー・ファイル: ICCGLBEH

関数

Icc 構造内の関数には、以下のものがあります。

boolText

「yes」や「on」などのパラメーターによって記述されるブール値を表すテキストを返します。

```
static const char* boolText (Bool test,
                             BoolSet set = trueFalse)
```

test

この構造で定義されている、*set* で指定される値のセットから選択された 2 つの値のうちの 1 つを持つブール値。

set

この構造で定義されている、どの値のペアから *test* を選択するかを示す列挙型。デフォルトでは、*true* と *false* を使用します。

catchException

これは、アプリケーションがキャッチできない **IccException** オブジェクトを代行受信するために使用される最後の手段の関数です。この関数は、ICCMAN ヘッダー・ファイルにリストされているスタブ・プログラム内の **main** 関数から呼び出すことができます。これについては、[248 ページの『main 関数』](#)で説明されています。すべての OO CICS プログラムで、このスタブまたはそれに類似したものを使用する必要があります。

```
static void catchException(IccException&exception)
```

exception

特定のタイプの例外に関する情報を保持する **IccException** オブジェクトへの参照。

conditionText

条件値に関連付けられたシンボル名を返します。例えば、*condition* に *IccCondition::NORMAL* を指定して **conditionText** が呼び出された場合は「NORMAL」を返し、*condition* に *IccCondition::IOERR* を指定して呼び出された場合は「IOERR」を返すなどです。

static const char* conditionText(IccCondition::Codes *condition*)

condition

IccCondition 構造で定義されている、CICS への呼び出しによって返される条件を示す列挙型。

initializeEnvironment

CICS ファウンデーション・クラスを初期化します。残りのクラス・ライブラリーは、この関数が呼び出された後でのみ、呼び出すことができます。この関数は、スタブ・プログラム内の **main** 関数 (ICCMAN ヘッダー・ファイルにリストされている) から呼び出されます。これについては、『[CICS C++ main 関数](#)』で説明しています。すべての OO CICS プログラムで、このスタブまたはそれに類似したものを使用する必要があります。

**static void initializeEnvironment (ClassMemoryMgmt *mem* = cmmDefault,
FamilySubset *fam* = fsDefault,
Icc::Bool *EDF*)**

mem

ファウンデーション・クラスのメモリー管理ポリシーを示す、この構造に定義された列挙。

fam

すべてのプラットフォームで使用不可の CICS 機能の使用が許可されるかどうかを示す、この構造に定義された列挙。

EDF

EDF トレースを最初からオンにするかどうかを示すブール。

isClassMemoryMgmtOn

この構造で定義されている、クラス・メモリー管理がオンになっているかどうかを示すブール値を返します。

static Bool isClassMemoryMgmtOn()

isEDFOn

この構造で定義されている、EDF トレースがグローバル・レベルでオンになっているかどうかを示すブール値を返します。

static Bool isEDFOn()

この構造の **setEDF** と、[157 ページの『IccResource クラス』](#) および [プログラムのデバッグの IccResource クラス内の isEDFOn](#) および **setEDF** を参照してください。

isFamilySubsetEnforcementOn

この構造で定義されている、すべてのプラットフォームで使用可能ではない CICS 機能を使用することを許可するかどうかを示すブール値を返します。

static Bool isFamilySubsetEnforcementOn()

returnToCICS

この呼び出しは、CICS にプログラム・フローを返します。

static void returnToCICS()

この関数は、ICCMAN ヘッダー・ファイルにリストされているスタブ・プログラム内の **main** 関数によって呼び出されます。これについては、248 ページの『[main 関数](#)』で説明されています。すべての OO CICS プログラムで、このスタブまたはそれに類似したものを使用する必要があります。

setEDF

EDF トレースのオン/オフをグローバル・レベルで設定します。

static void setEDF(Icc::Bool onOff = off)

onOff

この構造で定義されている、EDF トレースが使用可能であるかどうかを示すブール値。EDF は、オブジェクト指向プログラムより EXEC CICS 呼び出しを使用するトレース・プログラムに適しているため、デフォルトがオフです。

unknownException

この関数は、ICCMAN ヘッダー・ファイル内の **main** 関数によって呼び出され、不明な例外を代行受信するために使用されます。

static void unknownException()

248 ページの『[main 関数](#)』 およびこの構造の **catchException** を参照してください。

列挙

このセクションでは、詳しく説明するために、他の CICS プラットフォーム (CICS(r) for AIX など) について言及しています。それらのプラットフォームにおける CICS ファウンデーション・クラスのテクノロジー・リリースがありました。

Bool

ブール値には、以下のように 3 つの同等のペアがあります。

- true、yes、on
- false、no、off

true、yes、および on は 1 になり、false、no、および off はゼロになります。したがって、テスト関数を以下のようにコーディングすることができます。

```
if (task()->isStartDataAvailable())
{
    //do something
}
```

注: 「true」と「false」は、z/OS 1.2 C/C++ コンパイラーのコンパイラー・キーワードであり、このコンパイラーあるいはそれ以降のバージョンを使用する場合は、ICCGLBEH では生成されません。

BoolSet

BoolSet 列挙型は以下のとおりです。

- trueFalse
- yesNo
- onOff

ClassMemoryMgmt

ClassMemoryMgmt 列挙型は以下のとおりです。

cmmDefault

各プラットフォームのデフォルトは以下のとおりです。

z/OS

cmmNonCICS

UNIX

cmmCICS

cmmNonCICS

C++ 環境は、プログラムに必要なメモリー管理を実行します。

z/OS Language Environment では、タスクの終了時、あるいはタスクが異常終了した場合に、CICS タスク用のストレージが解放されるようにする必要があります。

CICS for AIX では、タスクが正常終了あるいは異常終了のどちらであった場合も動的なストレージの解放は行われません。これは、プログラムがメモリー・リークの影響を受けやすいことを意味します。

cmmCICS

IccBase クラス内で定義されている **new** 演算子および **delete** 演算子は、ストレージ割り振りを CICS にマップします。ストレージは、タスクの終了時に自動的に解放されます。

FamilySubset

FamilySubset 列挙型は以下のとおりです。

fsDefault

各プラットフォームのデフォルトは、すべて同じです (fsAllowPlatformVariance)。

fsEnforce

ファミリー・サブセットの規格適合を強制的に実施します。つまり、すべての CICS サーバー (OS/2、AIX、および z/OS) で使用可能ではない CICS 機能の使用は許可しません。

注: CICS OS/2 は、現在はサポートされていません。

fsAllowPlatformVariance

各プラットフォームがそのプラットフォームで 使用可能なすべての CICS 機能にアクセスすることを許可します。

GetOpt

この列挙型は、さまざまなクラスの多くのメソッドで使用されます。これは、オブジェクトによって内部的に保持されている値を呼び出し元に返すか、その前に CICS からその値を更新する必要があるかを示します。

object

値が以前に CICS から取得されてオブジェクト内に保管されている場合は、この保管されている値を返します。それ以外の場合は、CICS から値のコピーを取得し、オブジェクト内に保管します。

CICS

以前の呼び出しによってオブジェクト内に保管された値が既に存在する場合でも、オブジェクトが CICS から新しい値を取得し、それをオブジェクト内に保管することを強制します。

プラットフォーム

プログラムが実行されているオペレーティング・システムを示します。

可能な値は次のとおりです。

- OS2
- UNIX
- MVS™

IccAbendData クラス

これは、プログラム異常終了に関する診断情報を CICS から取得するために使用される singleton クラスです。

```
IccBase
  IccResource
    IccAbendData
```

ヘッダー・ファイル: ICCABDEH

IccAbendData コンストラクター (protected)

IccAbendData クラス内の IccAbendData コンストラクター

```
コンストラクター
IccAbendData()
```

public メソッド

これらは、このクラス内の public メソッドです。

opt パラメーター

多くのメソッドに、*opt* という同じパラメーターがあります。このパラメーターについては、**abendCode** メソッドの下で説明されています。

abendCode

現行の 4 文字の異常終了コードを返します。

```
const char* abendCode(Icc::GetOpt opt = Icc::object)
```

opt

Icc 構造で定義されている、値を CICS から更新するか、既存の値を保持するかを示す列挙型。使用可能な値については、[61 ページの『GetOpt』](#) の **Icc** 構造の **GetOpt** 列挙型の下で説明されています。

条件

INVREQ

ASRAInterrupt

最後にコード ASRA、ASRB、ASRD、または AICA の異常終了が発生した時点での 8 文字の状況ワード (PSW) 割り込み情報を返します。発行トランザクションの実行中に ASRA および ASRB のいずれの異常終了も発生しなかった場合、あるいは最初の異常終了がリモート DPL サーバー・プログラムで発生した場合は、フィールドには 2 進ゼロが入ります。

```
const char* ASRAInterrupt(Icc::GetOpt opt = Icc::object)
```


条件

INVREQ

ASRAKeyType

IccValue で定義されている、最後の ASRA、ASRB、AICA、または AEYD 異常終了時の実行キーを示す列挙型を返します (存在する場合)。

可能な値は次のとおりです。

CICSEXECKEY

最後の ASRA、ASRB、AICA、または AEYD の異常終了時に、タスクは CICS キーで実行されていました。CICS サブシステム・ストレージ保護がアクティブでない場合は、すべてのプログラムが CICS キーで実行されることに注意してください。

USEREXECKEY

最後の ASRA、ASRB、AICA、または AEYD の異常終了時に、タスクはユーザー・キーで実行されました。CICS サブシステム・ストレージ保護がアクティブでない場合は、すべてのプログラムが CICS キーで実行されることに注意してください。

NONCICS

最後の異常終了時の実行キーは、CICS キー (キー 8 またはキー 9) のいずれでもありませんでした。

NOTAPPLIC

ASRA、ASRB、AICA、および AEYD のいずれの異常終了もありませんでした。

IccValue::CVDA ASRAKeyType(Icc::GetOpt *opt* = Icc::object)

条件

INVREQ

ASRAPSW

最後にコード ASRA、ASRB、ASRD、または AICA の異常終了が発生した時点での 8 文字の状況ワード (PSW) を返します。発行トランザクションの実行中に ASRA、ASRB、ASRD、および AICA のいずれの異常終了も発生しなかった場合、あるいは最初の異常終了がリモート DPL サーバーで発生した場合は、フィールドにはヌルが入ります。

const char* ASRAPSW(Icc::GetOpt *opt* = Icc::object)

条件

INVREQ

ASRARegisters

最後の ASRA、ASRB、ASRD、または AICA 異常終了が発生した時点での汎用レジスター 0 から 15 の内容を 64 バイトのデータ域として返します。レジスターの内容は、0、1、...、15 の順序で返されます。発行トランザクションの実行中に ASRA、ASRB、ASRD、および AICA のいずれの異常終了も発生しなかった場合、あるいは最初の異常終了がリモート DPL サーバー・プログラムで発生した場合は、ヌルが返されるので注意してください。

const char* ASRARegisters(Icc::GetOpt opt = Icc::object)

条件

INVREQ

ASRASpaceType

IccValue 構造で定義されている、最後の ASRA、ASRB、AICA、または AEYD 異常終了時に制御権を持っていたスペースのタイプ (ある場合) を示す列挙型を返します。

可能な値は次のとおりです。

SUBSPACE

最後の ASRA、ASRB、AICA、または AEYD の異常終了時に、タスクは自身が所有するサブスペースまたは共通サブスペースのいずれかで実行されていました。

BASESPACE

最後の ASRA、ASRB、AICA、または AEYD の異常終了時に、タスクは基本スペースで実行されていました。トランザクション分離がアクティブでない場合、すべてのタスクが基本スペースで実行されるので注意してください。

NOTAPPLIC

ASRA、ASRB、AICA、および AEYD のいずれの異常終了もありませんでした。

IccValue::CVDA ASRASpaceType(Icc::GetOpt opt = Icc::object)

条件

INVREQ

ASRAStorageType

IccValue 構造で定義されている、最後の ASRA、ASRB、AICA、または AEYD 異常終了時にアドレス指定されていたストレージのタイプ (ある場合) を示す列挙型を返します。

可能な値は次のとおりです。

CICS

CICS キー・ストレージがアドレス指定されています。以下のいずれかに該当する場合、このストレージは、CICS 動的ストレージ域 (CDSA または ECDSA) のいずれか、あるいは読み取り専用動的ストレージ域 (RDSA または ERDSA) のいずれかにあります。

- CICS が、RENTPGM システム 初期設定パラメーターで NOPROTECT オプションを指定して稼働している
- ストレージ保護がアクティブになっていない

USER

ユーザー動的ストレージ域 (RDSA または ERDSA) のいずれかにあるユーザー・キー・ストレージがアドレス指定されています。

READONLY

CICS が RENTPGM システム 初期設定パラメーターで PROTECT オプションを指定して稼働している場合の、読み取り専用動的ストレージ域 (RDSA または ERDSA) のいずれかにある読み取り専用ストレージ。

NOTAPPLIC

以下のいずれかです。

- このタスクで ASRA および AEYD のいずれの異常終了も検出されていません。

- 異常終了の影響を受けたストレージが、CICS によって管理されていません。
- ASRA 異常終了の原因が OC4 異常終了ではありません。
- ASRB または AICA の異常終了が最後の ASRA または AEYD の異常終了以降 に発生した

IccValue::CVDA ASRAStorageType(Icc::GetOpt *opt* = Icc::object)

条件

INVREQ

instance

単一の **IccAbendData** オブジェクトを指すポインターを返します。オブジェクトが存在しない場合は、このメソッドによって作成されます。

static IccAbendData* instance()

isDumpAvailable

Icc 構造で定義されている、ダンプが生成されたかどうかを示すブール値を返します。生成されている場合は、**programName** メソッドを使用して、最新の異常終了について障害が発生したプログラムの名前を検索します。

Icc::Bool isDumpAvailable(Icc::GetOpt *opt* = Icc::object)

条件

INVREQ

originalAbendCode

繰り返し異常終了が発生した場合に、このタスクの最初の異常終了コードを返します。

const char* originalAbendCode(Icc::GetOpt *opt* = Icc::object)

条件

INVREQ

programName

異常終了の原因となったプログラムの名前を返します。

const char* programName(Icc::GetOpt *opt* = Icc::oldValue)

条件

INVREQ

継承された public メソッド

これらは、このクラスによって継承された public メソッドです。

メソッド	クラス
actionOnCondition	IccResource
actionOnConditionAsChar	IccResource
actionsOnConditionsText	IccResource
classType	IccBase
className	IccBase
condition	IccResource
conditionText	IccResource
customClassNum	IccBase
handleEvent	IccResource
id	IccResource
isEDFOn	IccResource
name	IccResource
operator delete	IccBase
operator new	IccBase
setActionOnAnyCondition	IccResource
setActionOnCondition	IccResource
setActionsOnConditions	IccResource
setEDF	IccResource

継承された protected メソッド

これらは、このクラスによって継承された protected メソッドです。

メソッド	クラス
setClassName	IccBase
setCustomClassNum	IccBase

IccAbsTime クラス

このクラスは、絶対時刻 (1900 年の始まり以来の経過時間をミリ秒数で示した時間) に関する情報を保持します。

IccBase
IccResource
IccTime
IccAbsTime

ヘッダー・ファイル: ICCTIMEH

IccAbsTime コンストラクター

IccAbsTime クラス内の IccAbsTime コンストラクター。

コンストラクター (1)

IccAbsTime(const char* *absTime*)

absTime

8 バイトの時刻値 (パック 10 進数フォーマット)。

コンストラクター (2)

コピー・コンストラクター。

IccAbsTime(const IccAbsTime& *time*)

public メソッド

これらは、このクラス内の public メソッドです。

date

日付を文字ストリングとして返します。

**const char* date (IccClock::DateFormat *format* = IccClock::defaultFormat,
char *dateSeparator* = '¥0')**

format

IccClock クラス内で定義されている、日付のフォーマットを示す列挙型。デフォルトではインストール済み環境のデフォルトが使用され、値は CICS 領域の初期設定時に設定されます。

dateSeparator

日付の各フィールドを分離する文字。デフォルトでは、分離文字はありません。

条件

INVREQ

dayOfMonth

月の何日か (1 から 31 の範囲) を返します。

unsigned long dayOfMonth()

条件

INVREQ

dayOfWeek

IccClock クラス内で定義されている、曜日を示す列挙型を返します。

IccClock::DayOfWeek dayOfWeek()

条件

INVREQ

daysSince1900

1900 年の初日以来に経過した日数を返します。

unsigned long daysSince1900()

条件

INVREQ

hours

時分秒の「時」のコンポーネントを返します。

virtual unsigned long hours() const

milliSeconds

1900 年の初日以来に経過したミリ秒数を返します。

long double milliSeconds()

minutes

時分秒の「分」のコンポーネントを返します。

virtual unsigned long minutes() const

monthOfYear

IccClock クラス内で定義されている、暦月を示す列挙型を返します。

IccClock::MonthOfYear monthOfYear()

条件

INVREQ

operator=

1つの **IccAbsTime** オブジェクトを別のオブジェクトに代入します。

IccAbsTime& operator=(const IccAbsTime& *absTime*)

packedDecimal

1900 年の始まり以来の経過時間をミリ秒数で表す 8 桁のパック 10 進数ストリングとして時間を返します。

const char* packedDecimal() const

seconds

時分秒の「秒」のコンポーネントを返します。

virtual unsigned long seconds() const

time

時刻をテキスト・ストリングとして返します。

const char* time(char *timeSeparator* = '¥0')

timeSeparator

時刻の各フィールドを分離する文字。デフォルトでは、時刻分離文字はありません。

条件

INVREQ

timeInHours

その日の始め以来経過した時間数を返します。

unsigned long timeInHours()

timeInMinutes

その日の始め以来経過した分数を返します。

unsigned long timeInMinutes()

timeInSeconds

その日の始め以来経過した秒数を返します。

unsigned long timeInSeconds()

year

4桁の整数として年を返します (1996 など)。

unsigned long year()

条件

INVREQ

継承された public メソッド

これらは、IccAbsTime クラス内で継承された public メソッドです。

メソッド	クラス
actionOnCondition	IccResource
actionOnConditionAsChar	IccResource
actionsOnConditionsText	IccResource
classType	IccBase
className	IccBase
condition	IccResource
conditionText	IccResource
customClassNum	IccBase
handleEvent	IccResource
hours	IccTime
isEDFOn	IccResource
minutes	IccTime
operator delete	IccBase
operator new	IccBase
setActionOnAnyCondition	IccResource
setActionOnCondition	IccResource
setActionsOnConditions	IccResource
setEDF	IccResource
timeInHours	IccTime
timeInMinutes	IccTime
timeInSeconds	IccTime
type	IccTime

継承された protected メソッド

IccAbsTime クラス内で継承された protected メソッドには以下のものがあります。

メソッド	クラス
setClassName	IccBase
setCustomClassNum	IccBase

IccAlarmRequestId クラス

IccAlarmRequestId オブジェクトは、固有のアラーム要求を表します。

IccBase
IccResourceId
IccRequestId
IccAlarmRequestId

これには、要求 ID の 8 文字の名前と、4 バイトのタイマー・イベント制御域へのポインターが含まれます。**IccAlarmRequestId** は、アラームを設定する場合に **IccClock** クラスの **setAlarm** メソッドによって使用され、アラームを待つ場合に **IccTask** タスクの **waitOnAlarm** メソッドによって使用されます。

ヘッダー・ファイル: ICCRIDEH

IccAlarmRequestId コンストラクター

IccAlarmRequestId コンストラクターには、以下のものがあります。

コンストラクター (1)

情報が存在しない新規オブジェクトを作成します。

IccAlarmRequestId()

コンストラクター (2)

情報が設定されたオブジェクトを作成します。

**IccAlarmRequestId (const char* *nam*,
const void* *timerECA*)**

name

8 文字の要求名。

timerECA

4 バイトのタイマー・イベント制御域へのポインター。

コンストラクター (3)

コピー・コンストラクター。

IccAlarmRequestId(const IccAlarmRequestId& *id*)

id

IccAlarmRequestId オブジェクトへの参照。

public メソッド

これらのメソッドは、情報を **IccAlarmRequestId** オブジェクトにコピーするために使用されます。

isExpired

Icc 構造で定義されている、アラームが期限切れになっているかどうかを示すブール値を返します。

Icc::Bool isExpired()

operator= (1)

IccAlarmRequestId& operator=(const IccRequestId& *id*)

id

IccRequestId オブジェクトへの参照。

operator= (2)

IccAlarmRequestId& operator=(const IccAlarmRequestId& *id*)

id

IccAlarmRequestId オブジェクトへの参照。

operator= (3)

IccAlarmRequestId& operator=(const char* *requestName*)

requestName

アラーム要求の 8 文字の名前。

setTimerECA

void setTimerECA(const void* *timerECA*)

timerECA

4 バイトのタイマー・イベント制御域へのポインター。

timerECA

4 バイトのタイマー・イベント制御域へのポインターを返します。

```
const void* timerECA() const
```

継承された public メソッド

これらは、このクラスによって継承された public メソッドです。

メソッド	クラス
classType	IccBase
className	IccBase
customClassNum	IccBase
name	IccResourceId
nameLength	IccResourceId
operator delete	IccBase
operator new	IccBase

継承された protected メソッド

これらは、このクラスによって継承された protected メソッドです。

メソッド	クラス
operator=	IccResourceId
setClassName	IccBase
setCustomClassNum	IccBase

IccBase クラス

IccBase クラスは、すべての CICS ファウンデーション・クラスの派生元となる基本クラスです。

IccBase

(**IccBase** に関連付けられたメソッドについてはここで説明されていますが、実際には、それらのメソッドは派生されたクラスのオブジェクトに対してのみ呼び出すことができます。)

ヘッダー・ファイル: ICCBASEH

IccBase コンストラクター (protected)

IccBase クラス内の IccBase コンストラクター (protected)

コンストラクター

IccBase(ClassType type)

type

どのサブクラス・タイプであるかを示す列挙型。例えば、**IccTempStore** オブジェクトの場合、クラス・タイプは「cTempStore」です。

public メソッド

これらは、このクラス内の public メソッドです。

opt パラメーター

多くのメソッドに、*opt* という同じパラメーターがあります。このパラメーターについては、[62 ページの『abendCode』](#)の **abendCode** メソッドの下で説明しています。

classType

サブクラス・タイプを示す列挙型を返します。例えば、**IccTempStore** オブジェクトの場合、クラス・タイプは「cTempStore」です。使用可能な値は、ページ [ClassType](#) の **ClassType** の下にリストされています。

ClassType classType() const

className

クラスの名前を返します。例えば、**IccTempStore** オブジェクトは、「IccTempStore」を返します。クラス **MyDataQueue** は **IccDataQueue** から継承されると仮定します。**MyDataQueue** が **setClassName("MyDataQueue")** を呼び出した場合、**MyDataQueue::className(IccBase::customName)** は「MyDataQueue」を返し、**MyDataQueue::className(IccBase::baseName)** は「IccDataQueue」を返します。**IccDataQueue** オブジェクトは、両方の *opt* 値に対して「IccDataQueue」を返します。

const char* className(NameOpt opt=customName)

opt

このクラス内で定義されている列挙子であり、クラスのベース名を返すか、派生したクラスによってカスタマイズされた名前を返すかを示します。

customClassNum

アプリケーション設計者が、自分が設計したサブクラスに関連付けた番号を返します。

unsigned short customClassNum() const

operator delete

正規の方法でオブジェクトを破棄します。

void operator delete(void* object)

object

破棄するオブジェクトを指すポインター。

operator new

指定されたサイズの新規オブジェクトを作成します。この演算子を使用すると、ファウンデーション・クラスが CICS ストレージ割り振りを使用することができます (59 ページの『[initializeEnvironment](#)』を参照)。

void* operator new(size_t size)

size

作成されるオブジェクトのサイズ (バイト数)。

protected メソッド

setClassName

クラスの名前を設定します。オブジェクトが属するクラスの名前のストリング表記を取得できるようにすることは、診断目的では有用です。

void setClassName(const char* className)

className

クラスの名前。例えば、**IccTempStore** を特殊化したクラス **MyTempStore** を作成する場合に、**setClassName("MyTempStore")** を呼び出すことがあります。

setCustomClassNum

提供時にはそのクラスのオリジナルの部分ではなかったサブクラスに識別番号を割り当てます。

void setCustomClassNum(unsigned short number)

number

アプリケーション設計者が識別の目的でサブクラスに関連付ける番号。

列挙

IccBase クラス内の列挙型には、以下のものがあります。

ClassType

名前は、クラスの名前から最初の 2 文字を削除することで生成されます。

可能な値は次のとおりです。

- cAbendData
- cAlarmRequestId
- cBuf
- cClock
- cConsole

- cControl
- cConvId
- cCUSTOM
- cDataQueue
- cDataQueueId
- cEvent
- cException
- cFile
- cFileId
- cFileIterator
- cGroupId
- cJournal
- cJournalId
- cJournalTypeId
- cLockId
- cMessage
- cPartnerId
- cProgram
- cProgramId
- cRecordIndex
- cRequestId
- cSemaphore
- cSession
- cStartRequestQ
- cSysId
- cSystem
- cTask
- cTempStore
- cTempStoreId
- cTermId
- cTerminal
- cTerminalData
- cTime
- cTPNameId
- cTransId
- cUser
- cUserId

注 : cCUSTOM を使用すると、IBM 以外の開発者がクラス・ライブラリーを拡張することができます。

NameOpt

列挙型内の NameOpt は以下のとおりです。

74 ページの『[className](#)』を参照してください。

baseName

IBM によってクラスに割り当てられているデフォルト名を返します。

customName

サブクラスから **setClassName** メソッドを使用して割り当てられた名前、あるいは **setClassName** が呼び出されていない場合は *baseName* と同じ名前を返します。

IccBuf クラス

IccBuf クラスは、バッファの一般的な操作に提供されています。

IccBase

IccBuf

このクラスは、CICS に対して呼び出しを行う他のクラスによって使用されますが、このクラス自体は CICS サービスを呼び出しません。[バッファ・オブジェクト](#)を参照してください。

ヘッダー・ファイル: ICCBUFEH

サンプル: ICC\$BUF

IccBuf コンストラクター

IccBuf クラス内の **IccBuf** コンストラクターには、以下のものがあります。

コンストラクター (1)

IccBuf オブジェクトを作成し、その独自のデータ域を指定された長さで割り振り、その領域内のすべてのバイトを NULL に設定します。

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

length

データ域の初期の長さ (バイト単位)。デフォルトの長さは 0 です。

type

データ域を動的に拡張できるかどうかを示す列挙型。可能な値は *extensible* または *fixed* です。デフォルトは *extensible* です。

コンストラクター (2)

拡張できない **IccBuf** オブジェクトを作成し、指定されたデータ域を自身のデータ域として採用します。[バッファの内部/外部所有権](#)に関する警告を参照してください。

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

length

提供されたデータ域の長さ (バイト数)。

dataArea

提供されるデータ域の最初のバイトのアドレス。

コンストラクター (3)

IccBuf オブジェクトを作成し、独自のデータ域を *text* スtringと同じ長さで割り振り、Stringをそのデータ域にコピーします。

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

text

新規の **IccBuf** オブジェクトにコピーされるヌル終了String。

type

データ域を拡張できるかどうかを示す列挙型。可能な値は、**extensible** または **fixed** です。デフォルトは **extensible** です。

コンストラクター (4)

コピー・コンストラクター。指定されたオブジェクトのコピーである新規の **IccBuf** オブジェクトを作成します。作成される **IccBuf** オブジェクトには、常に内部データ域があります。

IccBuf(const IccBuf& *buffer*)

buffer

新規オブジェクトにコピーされる **IccBuf** オブジェクトへの参照。

public メソッド

これらは、このクラス内の public メソッドです。

append (1)

指定されたデータ域のデータをオブジェクト内のデータ域に追加します。

**IccBuf& append (unsigned long *length*,
const void* *dataArea*)**

length

ソース・データ域の長さ (バイト数)

dataArea

ソース・データ域のアドレス。

append (2)

書式制御Stringおよび変数引数の形式で、オブジェクト内のデータ域にデータを追加します。これは、標準Cライブラリー内の **printf** で使用される形式と同じです。オプション・パラメーターが書式制御Stringと整合していることの確認は、アプリケーション・プログラマーが行う必要があるので注意してください。

**IccBuf& append (const char* *format*,
...)**

format

ヌル終了書式制御ストリング

...

オプションのパラメーター。

assign (1)

指定されたデータ域のデータをオブジェクト内のデータ域に代入します。

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

length

ソース・データ域の長さ (バイト数)

dataArea

ソース・データ域のアドレス。

assign (2)

書式制御ストリングおよび変数引数の形式で、オブジェクト内のデータ域にデータを代入します。これは、標準 C ライブラリー内の **printf** で使用される形式と同じです。

**IccBuf& assign (const char* *format*,
...)**

format

書式制御ストリング

...

オプションのパラメーター。

cut

データ域のデータに対して指定された切り取りを行い、**IccBuf** オブジェクトへの参照を返します。

**IccBuf& 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。デフォルトはオフセットなしです。

dataArea

データ域への指定されたオフセットにあるデータのアドレスを返します。

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

offset

データ域へのオフセット。デフォルトはオフセットなしです。

dataAreaLength

データ域の長さ (バイト数) を返します。

unsigned long dataAreaLength() const

dataAreaOwner

データ域が **IccBuf** コンストラクターによって割り振られたか、それ以外の場所から提供されたかを示す列挙型を返します。

DataAreaOwner dataAreaOwner() const

可能な値は、[86 ページの『DataAreaOwner』](#)の下にリストされています。

dataAreaType

DataAreaType dataAreaType() const

データ域を拡張できるかどうかを示す列挙型を返します。可能な値は、[86 ページの『DataAreaType』](#)の下にリストされています。

dataLength

データ域内のデータの長さを返します。この値は、**dataAreaLength**によって返される値を超えてはなりません。

unsigned long dataLength() const

insert

データ域の指定されたオフセットに指定されたデータを挿入し、**IccBuf** オブジェクトへの参照を返します。

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

length

IccBuf オブジェクトに挿入されるデータの長さ (バイト数)。

dataArea

IccBuf オブジェクトに挿入されるソース・データの開始位置。

offset

データを挿入するデータ域のオフセット。デフォルトはオフセットなしです。

isFMHContained

Icc::Bool isFMHContained() const

Icc 構造で定義されている、データ域に FMH (機能管理ヘッダー) が含まれているかどうかを示すブール値を返します。

operator const char*

operator const char*() const

IccBuf オブジェクトをヌル終了ストリングにキャストします。

```
IccBuf data("Hello World");  
cout << (const char*) data;
```

operator= (1)

別のバッファ・オブジェクトからのデータを代入し、**IccBuf** オブジェクトへの参照を返します。

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

buffer

IccBuf オブジェクトへの参照。

operator= (2)

ヌル終了ストリングからのデータを代入し、**IccBuf** オブジェクトへの参照を返します。**assign** メソッドも参照してください。

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

text

IccBuf オブジェクトに代入されるヌル終了ストリング。

operator+=(1)

別のバッファ・オブジェクトからのデータを追加し、**IccBuf** オブジェクトへの参照を返します。

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

buffer

IccBuf オブジェクトへの参照。

operator+=(2)

ヌル終了ストリングからのデータを追加し、**IccBuf** オブジェクトへの参照を返します。**append** メソッドも参照してください。

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

text

IccBuf オブジェクトに追加されるヌル終了ストリング。

operator==

Icc 構造で定義されている、2つの **IccBuf** オブジェクトのバッファに含まれているデータが同じであるかどうかを示すブール値を返します。現行の2つのデータ域の長さが同じで、内容も同じである場合は true です。

Icc::Bool operator==(const IccBuf& *buffer*) const

buffer

IccBuf オブジェクトへの参照。

operator!=

Icc 構造で定義されている、2つの **IccBuf** オブジェクトのバッファに含まれているデータが異なっているかどうかを示すブール値を返します。現行の2つのデータ域の長さが異なっている場合、あるいは内容が異なっている場合は true です。

Icc::Bool operator!=(const IccBuf& *buffer*) const

buffer

IccBuf オブジェクトへの参照。

operator« (1)

別のバッファを追加します。

operator«(const IccBuf& *buffer*)**operator« (2)**

ストリングを追加します。

operator«(const char* *text*)**operator« (3)**

文字を追加します。

operator«(char *ch*)**operator« (4)**

文字を追加します。

operator«(signed char *ch*)**operator« (5)**

文字を追加します。

operator«(unsigned char *ch*)**operator« (6)**

ストリングを追加します。

operator«(const signed char* *text*)**operator« (7)**

ストリングを追加します。

operator«(const unsigned char* *text*)

operator« (8)

short 型を追加します。

operator«(short *num*)**operator« (9)**

符号なし short 型を追加します。

operator«(unsigned short *num*)**operator« (10)**

long 型を追加します。

operator«(long *num*)**operator« (11)**

符号なし long 型を追加します。

operator«(unsigned long *num*)**operator« (12)**

整数を追加します。

operator«(int *num*)**operator« (13)**

float 型を追加します。

operator«(float *num*)**operator« (14)**

double 型を追加します。

operator«(double *num*)**operator« (15)**

long double 型を追加します。

operator«(long double *num*)

さまざまなタイプのデータを **IccBuf** オブジェクトに追加します。各タイプは、「読み取り可能」なフォーマットに変換されます (long 型からストリング表記など)。

overlay

データ域を外部の固定されたデータ域にします。既存の内部データ域はすべて破棄されます。[バッファの内部/外部所有権](#)に関する警告を参照してください。

IccBuf& overlay (unsigned long *length*, void* *dataArea*)

length

既存のデータ域の長さ。

dataArea

既存のデータ域のアドレス。

replace

データ域の指定されたオフセットの位置にある現在の内容を、提供されたデータに置き換え、**IccBuf** オブジェクトへの参照を返します。

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

length

ソース・データ域の長さ (バイト単位)。

dataArea

ソース・データ域の開始アドレス。

offset

IccBuf データ域の開始位置から見た、新しいデータを書き込む相対的な位置。デフォルトはオフセットなしです。

setDataLength

現行のデータ域の長さを変更し、新しい長さを返します。**IccBuf** オブジェクトが拡張可能でない場合、データ域の長さは、そのデータ域の元の長さと *length* のどちらか小さい方の長さに設定されます。

unsigned long setDataLength(unsigned long *length*)

length

データ域の新しい長さ (バイト数)。

setFMHContained

アプリケーション・プログラムが、データ域に機能管理ヘッダーが含まれていることを示すことが可能になります。

void setFMHContained(Icc::Bool yesNo = Icc::yes)

yesNo

Icc 構造で定義されている、データ域に FMH が含まれているかどうかを示すブール値。デフォルト値は **yes** です。

継承された public メソッド

これらは、このクラスによって継承された public メソッドです。

メソッド	クラス
className	IccBase
classType	IccBase
customClassNum	IccBase
operator delete	IccBase
operator new	IccBase

継承された protected メソッド

これらは、このクラスによって継承された protected メソッドです。

メソッド	クラス
setClassName	IccBase
setCustomClassNum	IccBase

列挙

DataAreaOwner

IccBuf 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たかどうかを示します。

可能な値は次のとおりです。

内部

データ域は **IccBuf** コンストラクターによって割り振られています。

外部

データ域は外部で割り振られています。

DataAreaType

IccBuf オブジェクトのデータ域を、元の長さより大きくできるかどうかを示します。

可能な値は次のとおりです。

extensible

データ域を自動的に拡張して、さらに多くのデータを格納できます。

fixed

データ域のサイズを大きくすることはできません。代入しようとするデータが多すぎる場合はデータが切り捨てられ、例外がスローされます。

IccClock クラス

IccClock クラスは、CICS 日時サービスへのアクセスを制御します。

IccBase
IccResource
IccClock

ヘッダー・ファイル: ICCCLKEH

サンプル: ICC\$CLK

IccClock コンストラクター

コンストラクター

IccClock(UpdateMode *update* = manual)

update

このクラス内で定義されている、時刻または日付サービスが使用されたときにクロックの時刻を自動的に更新するか、それとも明示的に **update** メソッドが呼び出されるまで待つかを示す列挙型。時刻を手動で更新する場合、初期クロック時刻は、**IccClock object** オブジェクトが作成された時刻です。

public メソッド

これらは、このクラス内の public メソッドです。

absTime

CICS によって提供された絶対時刻が含まれている **IccAbsTime** オブジェクトへの参照を返します。

IccAbsTime& absTime()

cancelAlarm

アラーム時刻にまだ達していない (要求が期限切れになっていない) 場合に以前の **setAlarm** 要求を取り消します。

void cancelAlarm(const **IccRequestId*** *reqId* = 0)

reqId

アラーム要求に関する情報を保持している **IccRequestId** オブジェクトへのオプション・ポインター。

条件

ISCINVREQ、NOTAUTH、NOTFND、SYSIDERR

date

日付をstringとして返します。

```
const char* date (DateFormat format = defaultFormat,  
char dateSeparator = '¥0')
```

format

このクラス内で定義されている、日付を返すフォーマットを示す列挙型。

dateSeparator

日付内のさまざまなフィールドを分離するために使用する文字。デフォルトでは、分離文字はありません。

条件

INVREQ

dayOfMonth

日付の「日」コンポーネントを 1 から 31 の範囲で返します。

```
unsigned long dayOfMonth()
```

条件

INVREQ

dayOfWeek

このクラス内で定義されている、曜日を返す列挙型を返します。

```
DayOfWeek dayOfWeek()
```

条件

INVREQ

daysSince1900

1900 年 1 月 1 日からの経過日数を返します。

```
unsigned long daysSince1900()
```

条件

INVREQ

milliseconds

1900 年 1 月 1 日 00:00 からの経過時間をミリ秒数で返します。

long double milliSeconds()

monthOfYear

MonthOfYear monthOfYear()

このクラス内で定義されている、暦月を示す列挙型を返します。

条件

INVREQ

setAlarm

time で指定された時刻にアラームを設定します。これは、アラームの取り消しに使用できる **IccAlarmRequestId** オブジェクトへの参照を返します。**cancelAlarm** メソッドを参照してください。

クラス **IccTask** の [201 ページ](#)の『**waitOnAlarm**』メソッドも参照してください。

**const IccAlarmRequestId& setAlarm (const IccTime& *time*,
const IccRequestId* *reqId* = 0)**

time

時間情報が入っている **IccTime** オブジェクトへの参照。**IccTime** は抽象クラスであるため、*time* は、実際にはクラス **IccAbsTime**、**IccTimeOfDay**、または **IccTimeInterval** のオブジェクトです。

reqId

この特定のアラーム要求を識別するために使用される、**IccRequestId** を指すオプションのポインター。

条件

EXPIRED、INVREQ

time

時刻をテキスト・ストリングとして返します。

const char* time(char *timeSeparator* = '¥0')

timeSeparator

時刻の各フィールドを分離する文字。デフォルトでは、分離文字はありません。

条件

INVREQ

更新

CICS からのクロック日時を更新します。**IccClock** コンストラクターを参照してください。

void update()

year

unsigned long year()

4桁の年番号(1996など)を返します。

条件

INVREQ

継承された public メソッド

これらは、このクラスによって継承された public メソッドです。

メソッド	クラス
actionOnCondition	IccResource
actionOnConditionAsChar	IccResource
actionsOnConditionsText	IccResource
classType	IccBase
className	IccBase
condition	IccResource
conditionText	IccResource
customClassNum	IccBase
handleEvent	IccResource
id	IccResource
isEDFOn	IccResource
name	IccResource
operator delete	IccBase
operator new	IccBase
setActionOnAnyCondition	IccResource
setActionOnCondition	IccResource
setActionsOnConditions	IccResource
setEDF	IccResource

継承された protected メソッド

これらは、このクラスによって継承された protected メソッドです。

メソッド	クラス
setClassName	IccBase
setCustomClassNum	IccBase

列挙

DateFormat

- defaultFormat

- DDMMYY
- MMDDYY
- YYDDD
- YYDDMM
- YYMMDD
- DDMMYYYY
- MMDDYYYY
- YYYYDDD
- YYYYDDMM
- YYYYMMDD

DayOfWeek

曜日を示します。

- Sunday
- Monday
- Tuesday
- Wednesday
- Thursday
- Friday
- Saturday

MonthOfYear

暦月を示します。

- January
- February
- March
- April
- May
- June
- July
- August
- September
- October
- November
- December

UpdateMode

クロックが自動的に更新されるかどうかを示します。

manual

クロックは、初期にクロック作成時の時刻を保持します。その後、**update** メソッド呼び出しが行われたときにのみ更新されます。

自動

クロックは、いずれかの時刻メソッドか日付メソッド (例えば **daysSince1900**) が呼び出されたときに、CICS の現在の時刻と日付に更新されます。

IccCondition 構造体

この構造体には、すべての CICS 条件コードの列挙型が含まれます。

ヘッダー・ファイル: ICCCNDEH

列挙

コード

可能な値は次のとおりです。

値	値	値
0 NORMAL	35 TSIOERR	70 NOTAUTH
1 ERROR	36 MAPFAIL	—
2 RDATT	37 INVERRTERM	72 SUPPRESSED
3 WRBRK	38 INVMPsz	—
4 ICCEOF	39 IGRREQID	—
5 EODS	40 OVERFLOW	75 RESIDERR
6 EOC	41 INVLDC	—
7 INBFMH	42 NOSTG	—
8 ENDINPT	43 JIDERR	—
9 NONVAL	44 QIDERR	—
10 NOSTART	45 NOJBUFSP	80 NOSPOOL
11 TERMIDERR	46 DSSTAT	81 TERMERR
12 FILENOTFOUND	47 SELNERR	82 ROLLEDBACK
13 NOTFND	48 FUNCERR	83 END
14 DUPREC	49 UNEXPIN	84 DISABLED
15 DUPKEY	50 NOPASSBKRD	85 ALLOCERR
16 INVREQ	51 NOPASSBKWR	86 STRELERR
17 IOERR	—	87 OPENERR
18 NOSPACE	53 SYSIDERR	88 SPOLBUSY
19 NOTOPEN	54 ISCINVREQ	89 SPOLERR
20 ENDFILE	55 ENQBUSY	90 NODEIDERR
21 ILLOGIC	56 ENVDEFERR	91 TASKIDERR
22 LENGERR	57 IGRREQCD	92 TCIDERR
23 QZERO	58 SESSIONERR	93 DSNNOTFOUND
24 SIGNAL	59 SYSBUSY	94 LOADING
25 QBUSY	60 SESSBUSY	95 MODELIDERR
26 ITEMERR	61 NOTALLOC	96 OUTDESCERR
27 PGMIDERR	62 CBIDERR	97 PARTNERIDERR
28 TRANSIDERR	63 INVEXITREQ	98 PROFILEIDERR

値	値	値
29 ENDDATA	64 INVPARTNSET	99 NETNAMEIDERR
30 INVTSREQ	65 INVPARTN	100 LOCKED
31 EXPIRED	66 PARTNFAIL	101 RECORDBUSY
32 RETPAGE	---	102 UOWNOTFOUND
33 RTEFAIL	---	103 UOWLNOTFOUND
34 RTESOME	69 USERIDERR	

範囲

maxValue

CICS 条件の最大値 (現行は 103)。

IccConsole クラス

これは、CICS コンソールを表す singleton クラスです。

IccBase

IccResource

IccConsole

ヘッダー・ファイル: ICCONEH

サンプル: ICC\$CON

IccConsole コンストラクター (protected)

コンストラクター

これらのタスクは、1 つのタスクで 1 つしか許可されません。追加のオブジェクトを作成しようとする、例外がスローされます。

IccConsole()

public メソッド

これらは、このクラス内の public メソッドです。

opt パラメーター

多くのメソッドに、*opt* という同じパラメーターがあります。このパラメーターについては、[62 ページの『abendCode』](#)の **abendCode** メソッドの下で説明しています。

instance

CICS コンソールを表す単一の **IccConsole** オブジェクトを指すポインターを返します。オブジェクトが存在しない場合は、このメソッドによって作成されます。

static IccConsole* instance()

put

send 内のデータを CICS コンソールに書き込みます。**put** は、**write** と同義です。[ポリモアフィック動作を参照](#)してください。

virtual void put(const IccBuf& *send*)

send

コンソールに書き込むデータが含まれている **IccBuf** オブジェクトへの参照。

replyTimeout

unsigned long replyTimeout() const

応答タイムアウトの長さ (ミリ秒数) を返します。

resetRouteCodes

void resetRouteCodes()

IccConsole オブジェクト内に保持されているすべての宛先コードを削除します。

setAllRouteCodes

void setAllRouteCodes()

IccConsole オブジェクト内にすべての可能な宛先コード (つまり、1 から 28) を設定します。

setReplyTimeout (1)

void setReplyTimeout(IccTimeInterval& *interval*)

interval

必要な時間間隔の長さを記述する **IccTimeInterval** オブジェクトへの参照。

setReplyTimeout (2)

このメソッドは、2つの異なる形式で、応答タイムアウトの長さを設定するために使用されます。

void setReplyTimeout(unsigned long *seconds*)

seconds

必要な時間間隔の長さ (秒数)。

setRouteCodes

後続の **write** 呼び出しおよび **writeAndGetReply** 呼び出しで使用するために、オブジェクト内に宛先コードを保管します。この方法で最大 28 個のコードを保持できます。

```
void setRouteCodes (unsigned short numRoutes,  
...)
```

numRoutes

この呼び出しで指定された宛先コードの数 (この呼び出しの後に続く引数の数)。

...

numRoutes で指定された数 (1 つ以上) の引数。各引数は、1 から 28 の範囲のタイプ **unsigned short** の宛先コードです。

write

send 内のデータを CICS コンソールに書き込みます。

```
void write (const IccBuf& send,  
SeverityOpt opt = none)
```

send

コンソールに書き込むデータが含まれている **IccBuf** オブジェクトへの参照。

opt

コンソール・メッセージの重大度を示す列挙型。

条件

INVREQ、LENGERR、EXPIRED

writeAndGetReply

send 内のデータを CICS コンソールに書き込み、CICS オペレーターからの返信が含まれている **IccBuf** オブジェクトへの参照を返します。

```
const IccBuf& writeAndGetReply (const IccBuf& send,  
SeverityOpt opt= none)
```

send

コンソールに書き込むデータが含まれている **IccBuf** オブジェクトへの参照。

opt

コンソール・メッセージの重大度を示す列挙型。

条件

INVREQ、LENGERR、EXPIRED

継承された public メソッド

これらは、このクラスによって継承された public メソッドです。

メソッド	クラス
actionOnCondition	IccResource
actionOnConditionAsChar	IccResource
actionsOnConditionsText	IccResource
classType	IccBase
className	IccBase
condition	IccResource
conditionText	IccResource
customClassNum	IccBase
handleEvent	IccResource
id	IccResource
isEDFOn	IccResource
name	IccResource
operator delete	IccBase
operator new	IccBase
setActionOnAnyCondition	IccResource
setActionOnCondition	IccResource
setActionsOnConditions	IccResource
setEDF	IccResource

継承された protected メソッド

これらは、このクラスによって継承された protected メソッドです。

メソッド	クラス
setClassName	IccBase
setCustomClassNum	IccBase

列挙

SeverityOpt

可能な値は次のとおりです。

- なし
- warning
- error
- severe

IccControl クラス

IccControl クラスは、提供されたファウンデーション・クラスを使用するアプリケーション・プログラムを制御します。

IccBase
IccResource
IccControl

このクラスはアプリケーション・プログラム内の singleton クラスです。CICS タスクの下で実行される各プログラムには、単一の **IccControl** オブジェクトがあります。

IccControl は、アプリケーション・コードが書き込まれている純粹仮想 **run** メソッドを持つため、抽象基本クラスです。アプリケーション・プログラマーは、**IccControl** のサブクラスを作成し、**run** メソッドをインプリメントする必要があります。

ヘッダー・ファイル: ICCCTLEH

IccControl コンストラクター (protected)

コンストラクター

IccControl()

public メソッド

これらは、このクラス内の public メソッドです。

callingProgramId

このプログラムを呼び出したプログラムを表す **IccProgramId** オブジェクトへの参照を返します。実行プログラムが別のプログラムによって呼び出されたものではない場合、返される **IccProgramId** 参照にはヌル名が含まれます。

const IccProgramId& callingProgramId()

条件

INVREQ

cancelAbendHandler

以前にこの論理プログラム・レベルで設定された出口を取り消します。

void cancelAbendHandler()

条件

NOTAUTH、PGMIDERR

commArea

COMMAREA をカプセル化する **IccBuf** オブジェクトへの参照を返します。COMMAREA は、CICS プログラムとトランザクションの間でデータを受け渡すために使用される CICS メモリーの通信域です。

IccBuf& commArea()

条件

INVREQ

コンソール

単一の **IccConsole** オブジェクトを指すポインターを返します。このオブジェクトがまだ作成されていない場合、このメソッドはそれを作成してから、そのオブジェクトを指すポインターを返します。

IccConsole* console()

initData

const IccBuf& initData()

INITPARM システム 初期設定パラメーターでプログラムに対して指定された初期設定パラメーターが含まれている **IccBuf** オブジェクトへの参照を返します。

条件

INVREQ

instance

単一の **IccControl** オブジェクトを指すポインターを返します。このオブジェクトは、まだ存在していない場合には作成されます。

static IccControl* instance()

isCreated

static Icc::Bool isCreated()

IccControl オブジェクトが既に存在するかどうかを示すブール値を返します。可能な値は、true または false です。

programId

const IccProgramId& programId()

この実行プログラムを参照する **IccProgramId** オブジェクトへの参照を返します。

条件

INVREQ

resetAbendHandler

この論理プログラム・レベルで以前に取り消された異常終了ハンドラーを再アクティブ化します。(ページ 97 ページの『[cancelAbendHandler](#)』の **cancelAbendHandler** を参照してください)。

void resetAbendHandler()

条件

NOTAUTH、PGMIDERR

returnProgramId

この論理プログラム・レベルが戻り値を発行したときに制御を再開するプログラムを参照する **IccProgramId** オブジェクトへの参照を返します。

const IccProgramId& returnProgramId()

run

virtual void run() = 0

このメソッドは、アプリケーション・プログラマーが **IccControl** のサブクラス内にインプリメントする必要があります。

session

IccSession* session()

このプログラムの基本機能を表す **IccSession** オブジェクトを指すポインターを返します。このプログラムが基本装置としてのセッションを持っていない場合は、例外がスローされます。

setAbendHandler (1)

void setAbendHandler(const IccProgramId& *programId*)

programId

どのプログラムが影響を受けるかを示す **IccProgramId** オブジェクトへの参照。

setAbendHandler (2)

これらのメソッドは、指定したプログラムをこの論理プログラム・レベルの異常終了ハンドラーとして設定します。

void setAbendHandler(const char* *programName*)

programName

影響を受けるプログラムの名前。

条件

NOTAUTH、PGMIDERR

startRequestQ

IccStartRequestQ オブジェクトを指すポインターを返します。このオブジェクトがまだ作成されていない場合、このメソッドはそれを作成してから、そのオブジェクトを指すポインターを返します。

IccStartRequestQ* startRequestQ()

システム

IccSystem* system()

IccSystem オブジェクトを指すポインターを返します。このオブジェクトがまだ作成されていない場合、このメソッドはそれを作成してから、そのオブジェクトを指すポインターを返します。

タスク

IccTask* task()

IccTask オブジェクトを指すポインターを返します。このオブジェクトがまだ作成されていない場合、このメソッドはそれを作成してから、そのオブジェクトを指すポインターを返します。

terminal

IccTerminal* terminal()

IccTerminal オブジェクトを指すポインターを返します。このオブジェクトがまだ作成されていない場合、このメソッドはそれを作成してから、そのオブジェクトを指すポインターを返します。

このメソッドには、トランザクションがその基本機能として端末を持つ必要があるという条件があります。つまり、関連する物理端末がなければなりません。

継承された public メソッド

これらは、このクラスによって継承された public メソッドです。

メソッド	クラス
actionOnCondition	IccResource
actionOnConditionAsChar	IccResource
actionsOnConditionsText	IccResource
classType	IccBase

メソッド	クラス
className	IccBase
condition	IccResource
conditionText	IccResource
customClassNum	IccBase
handleEvent	IccResource
id	IccResource
isEDFOn	IccResource
name	IccResource
operator delete	IccBase
operator new	IccBase
setActionOnAnyCondition	IccResource
setActionOnCondition	IccResource
setActionsOnConditions	IccResource
setEDF	IccResource

継承された protected メソッド

これらは、このクラスによって継承された protected メソッドです。

メソッド	クラス
setClassName	IccBase
setCustomClassNum	IccBase

IccConvId クラス

IccConvId クラスは、APPC 会話を識別するために使用されます。

IccBase
IccResourceId
IccConvId

IccConvId クラスは、APPC 会話を識別するために使用されます。

ヘッダー・ファイル: ICCRIDEH

IccConvId コンストラクター

コンストラクター (1)

IccConvId(const char* convName)

convName
 4 文字の会話名。

コンストラクター (2)

コピー・コンストラクター。

IccConvId(const IccConvId& convId)

convId

IccConvId オブジェクトへの参照。

public メソッド

これらは、このクラス内の public メソッドです。

operator= (1)

IccConvId& operator=(const char* convName)

operator= (2)

新規値を代入します。

IccConvId& operator=(const IccConvId id)

継承された public メソッド

これらは、このクラスによって継承された public メソッドです。

メソッド	クラス
classType	IccBase
className	IccBase
customClassNum	IccBase
name	IccResourceId
nameLength	IccResourceId
operator delete	IccBase
operator new	IccBase

継承された protected メソッド

これらは、このクラスによって継承された protected メソッドです。

メソッド	クラス
operator=	IccResourceId
setClassName	IccBase
setCustomClassNum	IccBase

IccDataQueue クラス

このクラスは、CICS 一時データ・キューを表します。

IccBase
IccResource
IccDataQueue

ヘッダー・ファイル: ICCDATEH

サンプル: ICC\$DAT

IccDataQueue コンストラクター

コンストラクター (1)

IccDataQueue(const IccDataQueueId& id)

id

CICS 一時データ・キューの名前が入っている **IccDataQueueId** オブジェクトへの参照。

コンストラクター (2)

IccDataQueue(const char* queueName)

queueName

作成されるキューの 4 バイトの名前。queueName が無効な場合は例外がスローされます。

public メソッド

これらは、このクラス内の public メソッドです。

clear

empty と同義。[ポリモフィック動作](#)を参照してください。

virtual void clear()

empty

void empty()

キューを空にします。つまり、キュー上のすべての項目を削除します。

条件

ISCINVREQ、NOTAUTH、QIDERR、SYSIDERR、DISABLED、INVREQ

get

readItem と同義。 [ポリモフィック動作](#)を参照してください。

virtual const IccBuf& get()

put

writeItem と同義。 [ポリモフィック動作](#)を参照してください。

virtual void put(const IccBuf& *buffer*)

buffer

キューに入れるデータが含まれている **IccBuf** オブジェクトへの参照。

readItem

const IccBuf& readItem()

データ・キューから読み取られた 1 つの項目が含まれている **IccBuf** オブジェクトへの参照を返します。

条件

IOERR、ISCINVREQ、LENGERR、NOTAUTH、NOTOPEN、QBUSY、QIDERR、QZERO、SYSIDERR、DISABLED、INVREQ

writeItem (1)

void writeItem(const IccBuf& *item*)

item

キューに書き込むデータが含まれている **IccBuf** オブジェクトへの参照。

writeItem (2)

データの項目をキューに書き込みます。

void writeItem(const char* *text*)

text

キューに書き込まれるテキスト。

条件

IOERR、ISCINVREQ、LENGERR、NOSPACE、NOTAUTH、NOTOPEN、QIDERR、SYSIDERR、DISABLED、INVREQ

継承された public メソッド

これらは、このクラスによって継承された public メソッドです。

メソッド	クラス
actionOnCondition	IccResource
actionOnConditionAsChar	IccResource
actionsOnConditionsText	IccResource
className	IccBase
classType	IccBase
condition	IccResource
conditionText	IccResource
customClassNum	IccBase
handleEvent	IccResource
id	IccResource
isEDFOn	IccResource
isRouteOptionOn	IccResource
name	IccResource
operator delete	IccBase
operator new	IccBase
routeOption	IccResource
setActionOnAnyCondition	IccResource
setActionOnCondition	IccResource
setActionsOnConditions	IccResource
setEDF	IccResource
setRouteOption	IccResource

継承された protected メソッド

これらは、このクラスによって継承された protected メソッドです。

メソッド	クラス
setClassName	IccBase
setCustomClassNum	IccBase

IccDataQueueId クラス

IccDataQueueId は、CICS 一時データ・キュー名を識別するために使用されます。

IccBase

IccResourceId

IccDataQueueId

IccDataQueueId は、CICS 一時データ・キュー名を識別するために使用されます。

ヘッダー・ファイル: ICCRIDEH

IccDataQueueId コンストラクター

コンストラクター (1)

IccDataQueueId(const char* *queueName*)

queueName

4 文字のキュー名

コンストラクター (2)

IccDataQueueId(const IccDataQueueId& *id*)

id

IccDataQueueId オブジェクトへの参照。

public メソッド

これらは、このクラス内の public メソッドです。

operator= (1)

IccDataQueueId& operator=(const char* *queueName*)

queueName

4 文字のキュー名

operator= (2)

新規値を代入します。

IccDataQueueId& operator=(const IccDataQueueId& *id*)

id

IccDataQueueId オブジェクトへの参照。

継承された public メソッド

これらは、このクラスによって継承された public メソッドです。

メソッド	クラス
classType	IccBase
className	IccBase
customClassNum	IccBase
name	IccResourceId
nameLength	IccResourceId
operator delete	IccBase
operator new	IccBase

継承された protected メソッド

これらは、このクラスによって継承された protected メソッドです。

メソッド	クラス
operator=	IccResourceId
setClassName	IccBase
setCustomClassNum	IccBase

IccEvent クラス

IccEvent クラスには、CICS イベントと呼ばれる特定の CICS 呼び出しに関する情報が含まれます。

IccBase

IccEvent

ヘッダー・ファイル: ICCEVTEH

サンプル: ICC\$RES1

IccEvent コンストラクター

コンストラクター

IccEvent (const **IccResource*** *object*,
const char* *methodName*)

object

このイベントの原因となった **IccResource** オブジェクトを指すポインター。

methodName

イベントが作成される原因となったメソッドの名前。

public メソッド

これらは、このクラス内の public メソッドです。

className

このイベントの原因となったクラスの名前を返します。

const char* className() const

classType

IccBase::ClassType classType() const

IccBase クラスのページ 74 ページの『classType』の **classType** で説明されている、このイベントの原因となったクラスのタイプを示す列挙型を返します。

condition

この CICS イベントから返される条件を示す列挙型を返します。可能な値については、**IccCondition** 構造の **Codes** タイプの下で説明されています。

**IccCondition::Codes condition(IccResource::ConditionType type =
IccResource::majorCode) const**

type

メジャー・コードとマイナー・コードのどちらが要求されるかを示す列挙型。可能な値は「majorCode」または「minorCode」です。「majorCode」がデフォルト値です。

conditionText

const char* conditionText() const

CICS 条件コードのテキスト (「NORMAL」や「LENGERR」など) を返します。

methodName

const char* methodName() const

このイベントの原因となったメソッドの名前を返します。

summary

const char* summary()

以下の形式で CICS イベントの要約を返します。

```
CICS event summary: IccDataQueue::readItem condition=23 (QZERO) minor=0
```

継承された public メソッド

これらは、このクラスによって継承された public メソッドです。

メソッド	クラス
className	IccBase
classType	IccBase
customClassNum	IccBase
operator delete	IccBase
operator new	IccBase

継承された protected メソッド

これらは、このクラスによって継承された protected メソッドです。

メソッド	クラス
setClassName	IccBase
setCustomClassNum	IccBase

IccException クラス

IccException クラスは、CICS ファウンデーション・クラスの例外に関する情報を格納しています。

IccBase
IccException

このクラスは、アプリケーション・プログラムに「スロー」されるオブジェクトを作成するために使用されます。それらのオブジェクトは、一般的には無効なメソッド呼び出しなどのエラー条件に使用されますが、アプリケーション・プログラマーは、CICS で特定の条件が発生した場合に例外をスローするように要求することもできます。

ヘッダー・ファイル: ICCEXCEH

サンプル: ICC\$EXC1、ICC\$EXC2、ICC\$EXC3

IccException コンストラクター

コンストラクター

```
IccException (Type exceptionType,  
              IccBase::ClassType classType,  
              const char* className,  
              const char* methodName,  
              IccMessage* message,  
              IccBase* object = 0,  
              unsigned short exceptionNum = 0)
```

exceptionType

このクラス内で定義されている、例外のタイプを示す列挙型。

classType

このクラス内で定義されている、例外がスローされた元のクラスのタイプを示す列挙型。

className

例外がスローされた元のクラスの名前。

methodName

例外がスローされた元のメソッドの名前。

message

例外が作成された理由に関する情報が含まれている **IccMessage** オブジェクトを指すポインター。

object

例外をスローしたオブジェクトを指すポインター。

exceptionNum

固有の例外番号。

注 : **IccException** オブジェクトは、作成された時点で、コンストラクター上で指定された **IccMessage** の所有権を取得します。**IccException** が削除されると、**IccMessage** オブジェクトは **IccException** デストラクターによって自動的に削除されます。したがって、**IccException** オブジェクトを削除する前に **IccMessage** オブジェクトを削除しないでください。

public メソッド

これらは、このクラス内の public メソッドです。

className

この例外がスローされる原因となったクラスの名前を返します。

```
const char* className() const
```

classType

```
IccBase::ClassType classType() const
```

IccBase クラス内の **ClassType** で定義されている、この例外をスローしたクラスのタイプを示す列挙型を返します。

message

```
IccMessage* message() const
```

この例外に関連付けられたすべてのメッセージに関する情報が含まれている **IccMessage** オブジェクトを指すポインターを返します。

methodName

```
const char* methodName() const
```

この例外がスローされる原因となったメソッドの名前を返します。

number

unsigned short number() const

固有の例外番号を返します。

これは、IBM サービスに有用な診断です。番号は、ソース・コードのどこから例外がスローされたかを一意的に識別します。

summary

const char* summary()

例外の要約が含まれるストリングを返します。これは、**className**、**methodName**、**number**、**Type**、および **IccMessage::text** メソッドを以下の形式で結合したものです。

```
CICS exception summary: 094 IccTempStore::readNextItem type=CICSCondition
```

type

Type type() const

このクラス内で定義されている、例外のタイプを示す列挙型を返します。

typeText

const char* typeText() const

例外タイプのストリング表記 (「objectCreationError」や「invalidArgument」など) を返します。

継承された public メソッド

これらは、このクラスによって継承された public メソッドです。

メソッド	クラス
className	IccBase
classType	IccBase
customClassNum	IccBase
operator delete	IccBase
operator new	IccBase

継承された protected メソッド

これらは、このクラスによって継承された protected メソッドです。

メソッド	クラス
setClassName	IccBase

メソッド
setCustomClassNum

クラス
IccBase

列挙

タイプ

objectCreationError

オブジェクトの作成試行は無効でした。これは例えば、**IccTask** などの singleton クラスの 2 番目のインスタンスを作成しようとした場合に発生します。

invalidArgument

無効な引数でメソッドが呼び出されました。これは例えば、含まれているデータが多すぎる **IccBuf** オブジェクトがアプリケーション・プログラムによって **IccTempStore** クラスの **writeItem** メソッドに渡される場合に発生します。9 文字のファイル名を使用して **IccFileId** オブジェクトを作成すると、このタイプの例外も生成されます。

invalidMethodCall

メソッドの呼び出しを続行できません。一般的な原因は、オブジェクトが現在の状態では呼び出しに対応できないことです。例えば、読み取る **レコードを指定** するための **IccRecordIndex** オブジェクトが既にファイルに関連付けられている場合は、**IccFile** オブジェクト上の **readRecord** 呼び出しのみが受け入れられます。

CICSCondition

IccCondition 構造体にリストされている CICS 条件がオブジェクトで発生しました。このオブジェクトは例外をスローするように構成されていました。

platformError

この特定のプラットフォームの制限により、操作は無効です。

platformError 例外は、以下の 3 つのレベルで発生する可能性があります。

1. オブジェクトがそのプラットフォーム上でサポートされていない。
2. オブジェクトがこのプラットフォームでサポートされているものの、特定のメソッドがサポートされていません。
3. メソッドがこのプラットフォームでサポートされているものの、特定の定位置パラメーターがサポートされていません。

詳しくは、[プラットフォームの差異](#)を参照してください。

familyConformanceError

このプログラムではファミリー・サブセットの制約が有効になっており、サポートされているすべてのプラットフォームで無効な操作が試行されました。

internalError

CICS ファウンデーション・クラスが内部エラーを検出しました。サポート組織に連絡してください。

IccFile クラス

IccFile クラスは、アプリケーション・プログラムから CICS ファイルにアクセスできるようにします。

IccBase
IccResource
IccFile

ヘッダー・ファイル: ICCFILEH

サンプル: ICC\$FIL

IccFile コンストラクター

コンストラクター (1)

IccFile (const IccFileId& *id*,
IccRecordIndex* *index* = 0)

id

操作対象のファイルを識別する **IccFileId** オブジェクトへの参照。

index

ファイル内の操作対象のレコードを識別する **IccRecordIndex** オブジェクトへのオプション・ポインター。

コンストラクター (2)

IccFile オブジェクトを使用してファイルにアクセスするには、そのオブジェクトに **IccRecordIndex** オブジェクトが関連付けられている必要があります。オブジェクトの作成時にこの関連付けを行っていない場合は、**registerRecordIndex** メソッドを使用します。

IccFile (const char* *fileName*,
IccRecordIndex* *index* = 0)

fileName

8 文字のファイル名

index

ファイル内の操作対象のレコードを識別する **IccRecordIndex** オブジェクトへのオプション・ポインター。

public メソッド

これらは、このクラス内の public メソッドです。

opt パラメーター

多くのメソッドに、*opt* という同じパラメーターがあります。このパラメーターについては、[62 ページの『abendCode』](#) の **abendCode** メソッドの下で説明しています。

access

ファイルのアクセス・プロパティーを示す複合数値を返します。**isReadable**、**isBrowsable**、**isAddable**、**isDeletable**、および **isUpdatable** メソッドも参照してください。

unsigned long access(Icc::GetOpt *opt* =Icc::object)

opt

Icc 構造で定義されている、以前に CICS (オブジェクト) から取得された値を使用することができるか、オブジェクトが CICS から新規の値を取得する必要があるかを示す列挙型。

accessMethod

IccValue で定義されている、このファイルのアクセス方式を表す列挙型を返します。

可能な値は次のとおりです。

- VSAM
- BDAM
- SFS

IccValue::CVDA accessMethod(Icc::GetOpt *opt* = Icc::object)

opt

access メソッドを参照してください。

条件

END、FILENOTFOUND、ILLOGIC、NOTAUTH

beginInsert (VSAM のみ)

ファイルへのデータの大量挿入の開始をシグナル通知します。

void beginInsert()

deleteLockedRecord

以前に更新モードで **readRecord** メソッドによってロックされたレコードを削除します。(readRecord メソッドも参照してください。)

void deleteLockedRecord(unsigned long *updateToken* = 0)

updateToken

以前に読み取られたどのレコードを削除するかを示すトークン。これは、更新モードの **readRecord** メソッドから返されるトークンです。

条件

DISABLED、DUPKEY、FILENOTFOUND、ILLOGIC、INVREQ、IOERR、ISCINVREQ、NOTAUTH、NOTFIND、NOTOPEN、SYSIDERR、LOADING

deleteRecord

関連する **IccRecordIndex** オブジェクトによって指定された 1 つ以上のレコードを削除し、削除したレコードの数を返します。

unsigned short deleteRecord()

条件

DISABLED、DUPKEY、FILENOTFOUND、ILLOGIC、INVREQ、IOERR、ISCINVREQ、NOTAUTH、NOTFOUND、NOTOPEN、SYSIDERR、LOADING

enableStatus

IccValue で定義されている、プログラムがファイルを使用可能であるかどうかを示す列挙型を返します。

可能な値は次のとおりです。

- DISABLED
- DISABLING
- ENABLED
- UNENABLED

IccValue::CVDA enableStatus(Icc::GetOpt *opt* = Icc::object)

opt

access メソッドを参照してください。

条件

END、FILENOTFOUND、ILLOGIC、NOTAUTH

endInsert (VSAM のみ)

大量挿入操作の最後にマークを付けます。 **beginInsert** を参照してください。

void endInsert()

isAddable

ファイルにさらにレコードを追加できるかどうかを示します。

Icc::Bool isAddable(Icc::GetOpt *opt* = Icc::object)

opt

access メソッドを参照してください。

条件

END、FILENOTFOUND、ILLOGIC、NOTAUTH

isBrowsable

ファイルをブラウズできるかどうかを示します。

Icc::Bool isBrowsable(Icc::GetOpt *opt* = Icc::object)

opt

access メソッドを参照してください。

条件

END、FILENOTFOUND、ILLOGIC、NOTAUTH

isDeletable

ファイル内のレコードを削除できるかどうかを示します。

Icc::Bool isDeletable(Icc::GetOpt *opt* = Icc::object)

opt

access メソッドを参照してください。

条件

END、FILENOTFOUND、ILLOGIC、NOTAUTH

isEmptyOnOpen

EMPTYREQ オプションが指定されているかどうかを示すブール値を返します。EMPTYREQ が指定されていると、このファイルに関連付けられているオブジェクトは、再使用可能として定義された VSAM データ・セットである場合、オープン時に空に設定されます。

Icc::Bool isEmptyOnOpen(Icc::GetOpt *opt* = Icc::object)

opt

access メソッドを参照してください。

条件

END、FILENOTFOUND、ILLOGIC、NOTAUTH

isReadable

ファイル・レコードを読み取ることができるかどうかを示します。

Icc::Bool isReadable(Icc::GetOpt *opt* = Icc::object)

opt

access メソッドを参照してください。

条件

END、FILENOTFOUND、ILLOGIC、NOTAUTH

isRecoverable

Icc::Bool isRecoverable(Icc::GetOpt *opt* = Icc::object)

opt

access メソッドを参照してください。

条件: END、FILENOTFOUND、ILLOGIC、NOTAUTH

isUpdatable

ファイルを更新できるかどうかを示します。

Icc::Bool isUpdatable(Icc::GetOpt *opt* = Icc::object)

opt

access メソッドを参照してください。

条件

END、FILENOTFOUND、ILLOGIC、NOTAUTH

keyLength

検索キーの長さを返します。

unsigned long keyLength(Icc::GetOpt *opt* = Icc::object)

opt

access メソッドを参照してください。

条件

END、FILENOTFOUND、ILLOGIC、NOTAUTH

keyPosition

各レコード内のキー・フィールドの位置をレコードの先頭からの相対位置として返します。キーがない場合は、ゼロが返されます。

long keyPosition(Icc::GetOpt *opt* = Icc::object)

opt

access メソッドを参照してください。

条件

END、FILENOTFOUND、ILLOGIC、NOTAUTH

openStatus

ファイルのオープン状況を示す CVDA を返します。可能な値は以下のとおりです。

IccValue::CVDA openStatus(Icc::GetOpt opt = Icc::object)

opt

access メソッドを参照してください。

CLOSED

ファイルは閉じています。

CLOSING

ファイルは閉じられているところです。 ファイルを閉じるには、データ・セットの動的割り振り解除および共用リソースの削除が必要になる場合があるため、プロセスが非常に長時間続く可能性があります。

CLOSEREQUEST

ファイルは開いており、1つ以上のアプリケーションがそのファイルを使用しています。それをクローズする要求を受信しました。

OPEN

ファイルは開かれています。

OPENING

ファイルは開かれているところです。

条件: END、FILENOTFOUND、ILLOGIC、NOTAUTH

readRecord

レコードを読み取り、レコードからのデータが含まれている **IccBuf** オブジェクトへの参照を返します。

**const IccBuf& readRecord (ReadMode mode = normal,
unsigned long* updateToken = 0)**

mode

このクラス内で定義されている、レコードを読み取るモードを示す列挙型。

updateToken

mode が更新され、複数の読み取り更新を行いたい場合に、メソッドによって更新される **unsigned long** トークンへのポインター。 このトークンは、更新要求を一意的に識別し、**deleteLockedRecord**、**rewriteRecord**、あるいは **unlockRecord** メソッドに渡されます。

条件

DISABLED、DUPKEY、FILENOTFOUND、ILLOGIC、INVREQ、IOERR、ISCINVREQ、LENGERR、NOTAUTH、NOTFND、NOTOPEN、SYSIDERR、LOADING

recordFormat

データの形式を示す CVDA を返します。可能な値は以下のとおりです。

IccValue::CVDA recordFormat(Icc::GetOpt *opt* = Icc::object)

opt

access メソッドを参照してください。

FIXED

レコードは固定長です。

UNDEFINED (BDAM データ・セットのみ)

ファイル上のレコード・フォーマットは未定義です。

VARIABLE

レコードは可変長です。ファイルがデータ・テーブルに関連付けられている場合、ソース・データ・セットに固定長レコードが含まれている場合でも、レコード形式は常に可変長になります。

条件: END、FILENOTFOUND、ILLOGIC、NOTAUTH

recordIndex

readRecord、**writeRecord**、および **deleteRecord** などのメソッドの使用時にアクセスするレコードを示す **IccRecordIndex** オブジェクトを指すポインターを返します。

IccRecordIndex* recordIndex() const

recordLength

現行レコードの長さを返します。

unsigned long recordLength(Icc::GetOpt *opt* = Icc::object)

opt

access メソッドを参照してください。

条件

END、FILENOTFOUND、ILLOGIC、NOTAUTH

registerRecordIndex

void registerRecordIndex(IccRecordIndex* *index*)

index

readRecord、**writeRecord** などのメソッドで使用する **IccKey**、**IccRBA**、または **IccRRN** オブジェクトを指すポインター。

rewriteRecord

buffer の内容でレコードを更新します。

```
void rewriteRecord (const IccBuf& buffer,  
                    unsigned long updateToken = 0)
```

buffer

ファイルに書き込む新規のレコード・データを保持している **IccBuf** オブジェクトへの参照。

updateToken

以前に読み取られたどのレコードを再書き込みするかを識別するトークン。 **readRecord** を参照してください。

条件

DISABLED、FILENOTFOUND、ILLOGIC、INVREQ、IOERR、ISCINVREQ、NOTAUTH、NOTFND、NOTOPEN、SYSIDERR、LOADING

setAccess

ファイルに対して許可するアクセスを設定します。

例:

```
file.setAccess(IccFile::readable + IccFile::notUpdatable);
```

```
void setAccess(unsigned long access)
```

access

このクラス内で定義されている **Access** 列挙型の 1 つ以上の値を OR (または加算) することで作成される正整数。

条件

FILENOTFOUND、INVREQ、IOERR、NOTAUTH

setEmptyOnOpen

```
void setEmptyOnOpen(Icc::Bool trueFalse)
```

ファイルを次に開くときに空にするかどうかを指定します。

条件

FILENOTFOUND、INVREQ、IOERR、NOTAUTH

setStatus

ファイルの状況を設定します。

void setStatus(Status status)

status

このクラス内で定義されている、このメソッドの呼び出し後に必要なファイルの状況を示す列挙型。

条件

FILENOTFOUND、INVREQ、IOERR、NOTAUTH

type

このファイルに対応するデータ・セットのタイプを識別する CVDA を返します。可能な値は以下のとおりです。

IccValue::CVDA type(Icc::GetOpt opt = Icc::object)

opt

access メソッドを参照してください。

ESDS

データ・セットは入力順データ・セットです。

KEYED

データ・セットは物理キーによりアドレス指定されます。

KSDS

データ・セットはキー順データ・セットです。

NOTKEYED

データ・セットは物理キーによりアドレス指定されません。

RRDS

データ・セットは相対レコード・データ・セットです。

VRRDS

データ・セットは可変相対レコード・データ・セットです。

条件: END、FILENOTFOUND、ILLOGIC、NOTAUTH

unlockRecord

以前に更新モードで読み取ることでロックされたレコードをアンロックします。 **readRecord** を参照してください。

void unlockRecord(unsigned long updateToken = 0)

updateToken

以前の **readRecord** 更新要求のどれをアンロックするかを示すトークン。

条件

DISABLED、FILENOTFOUND、ILLOGIC、IOERR、ISCINVREQ、NOTAUTH、NOTOPEN、SYSIDERR、INVREQ

writeRecord

beginInsert メソッドまたは **endInsert** メソッドと一緒に使用された場合、単一のレコードまたは一連のレコードを書き込みます。

void writeRecord(const IccBuf& *buffer*)

buffer

レコードに書き込むデータを保持している **IccBuf** オブジェクトへの参照。

条件

DISABLED、DUPREC、FILENOTFOUND、ILLOGIC、INVREQ、IOERR、ISCINVREQ、LENGERR、NOSPACE、NOTAUTH、NOTOPEN、SYSIDERR、LOADING、SUPPRESSED

継承された public メソッド

これらは、このクラスによって継承された public メソッドです。

メソッド	クラス
actionOnCondition	IccResource
actionOnConditionAsChar	IccResource
actionsOnConditionsText	IccResource
className	IccBase
classType	IccBase
condition	IccResource
conditionText	IccResource
customClassNum	IccBase
handleEvent	IccResource
id	IccResource
isEDFOn	IccResource
isRouteOptionOn	IccResource
name	IccResource
operator delete	IccBase
operator new	IccBase
routeOption	IccResource

メソッド	クラス
setActionOnAnyCondition	IccResource
setActionOnCondition	IccResource
setActionsOnConditions	IccResource
setEDF	IccResource
setRouteOption	IccResource

継承された protected メソッド

これらは、このクラスによって継承された protected メソッドです。

メソッド	クラス
setClassName	IccBase
setCustomClassNum	IccBase

列挙

Access

readable

ファイル・レコードは CICS タスクによって読み取ることができます。

notReadable

CICS タスクはファイル・レコードを読み取ることができません。

browsable

CICS タスクはファイル・レコードをブラウズすることができます。

notBrowsable

CICS タスクはファイル・レコードをブラウズすることができません。

addable

CICS タスクはレコードをファイルに追加することができます。

notAddable

CICS タスクはレコードをファイルに追加することができません。

updatable

CICS タスクはファイル内のレコードを更新することができます。

notUpdatable

CICS タスクはファイル内のレコードを更新することができません。

deletable

CICS タスクはファイル内のレコードを削除することができます。

notDeletable

CICS タスクはファイル内のレコードを削除することができません。

fullAccess

readable AND browsable AND addable AND updatable AND deletable と等価。

noAccess

notReadable AND notBrowsable AND notAddable AND notUpdatable AND notDeletable と等価。

ReadMode

ReadMode は、ファイルを読み取るモードです。

normal

更新は実行されません (読み取り専用モード)

update

レコードは更新されます。以下が行われるまで、レコードは CICS によってロックされます。

- **rewriteRecord** メソッドを使用して書き込まれる。または
- **deleteLockedRecord** メソッドを使用して削除される。または
- **unlockRecord** メソッドを使用してアンロックされる。または
- タスクがそのリソース更新をコミットまたはロールバックする。または
- タスクが異常終了する。

SearchCriterion

equalToKey

検索では、完全一致のみを検出します。

gteqToKey

検索では、完全一致または検索順序内の次のレコードを検出します。

状況

open

ファイルは開かれており、CICS タスクにより読み取り/書き込み要求に対する準備ができています。

closed

ファイルは閉じられており、現在は CICS タスクによって使用されていません。

enabled

CICS タスクはファイルにアクセスすることができます。

disabled

CICS タスクはファイルにアクセスすることができません。

IccFileId クラス

IccFileId は、CICS システム内のファイル名を識別するために使用されます。

IccBase

IccResourceId

IccFileId

ヘッダー・ファイル: ICCRIDEH

IccFileId コンストラクター

コンストラクター (1)

IccFileId(const char* *fileName*)

fileName

ファイルの名前。

コンストラクター (2)

IccFileId(const IccFileId& *id*)

id

IccFileId オブジェクトへの参照。

public メソッド

これらは、このクラス内の public メソッドです。

operator= (1)

IccFileId& operator=(const char* *fileName*)

fileName

8 バイトのファイル名。

operator= (2)

新規値を代入します。

IccFileId& operator=(const IccFileId& *id*)

id

IccFileId オブジェクトへの参照。

継承された public メソッド

これらは、このクラスによって継承された public メソッドです。

メソッド	クラス
classType	IccBase
className	IccBase
customClassNum	IccBase
name	IccResourceId
nameLength	IccResourceId
operator delete	IccBase
operator new	IccBase

継承された protected メソッド

これらは、このクラスによって継承された protected メソッドです。

メソッド	クラス
operator=	IccResourceId
setClassName	IccBase
setCustomClassNum	IccBase

IccFileIterator クラス

このクラスは、**IccFile** オブジェクトで表される CICS ファイルのレコードをブラウズするために使用できる **IccFileIterator** オブジェクトを作成するために使用されます。

IccBase
IccResource
IccFileIterator

ヘッダー・ファイル : ICCFLIEH

サンプル : ICC\$FIL

IccFileIterator コンストラクター

コンストラクター

IccFileIterator を作成する前に、**IccFile** および **IccRecordIndex** オブジェクトが存在している必要があります。

```
IccFileIterator (IccFile* file,  
                 IccRecordIndex* index,  
                 IccFile::SearchCriterion search = IccFile::gteqToKey)
```

ファイル

ブラウズされる **IccFile** オブジェクトを指すポインター

index

ファイル内のレコードを選択するために使用される **IccRecordIndex** オブジェクトを指すポインター。

search

IccFile で定義されている、検索の一致を検出するために使用する基準を示す列挙型。デフォルトは **gteqToKey** です。

条件

DISABLED、FILENOTFOUND、ILLOGIC、INVREQ、IOERR、ISCINVREQ、NOTAUTH、NOTFND、NOTOPEN、SYSIDERR、LOADING

public メソッド

これらは、このクラス内の public メソッドです。

readNextRecord

現行レコードの後にあるレコードを読み取ります。

```
const IccBuf& readNextRecord (IccFile::ReadMode mode = IccFile::normal,  
                             unsigned long* updateToken = 0)
```

mode

IccFile クラス内で定義されている、読み取り要求のタイプを示す列挙型。

updateToken

ファイル・オブジェクトに対する後続の **rewriteRecord**、**deleteLockedRecord**、または **unlockRecord** メソッドでの固有の更新要求の識別に使用するために返されるトークン。

条件

DUPKEY、ENDFILE、FILENOTFOUND、ILLOGIC、INVREQ、IOERR、ISCINVREQ、LENGERR、NOTAUTH、NOTFIND、SYSIDERR

readPreviousRecord

現行レコードの前にあるレコードを読み取ります。

```
const IccBuf& readPreviousRecord (IccFile::ReadMode mode = IccFile::normal,
                                unsigned long* updateToken = 0)
```

mode

IccFile クラス内で定義されている、読み取り要求のタイプを示す列挙型。

updateToken

readNextRecord を参照してください。

条件

DUPKEY、ENDFILE、FILENOTFOUND、ILLOGIC、INVREQ、IOERR、ISCINVREQ、LENGERR、NOTAUTH、NOTFIND、SYSIDERR

reset

IccRecordIndex オブジェクトおよび指定された検索基準によって識別されたレコードを指すように **IccFileIterator** オブジェクトをリセットします。

```
void reset (IccRecordIndex* index,
            IccFile::SearchCriterion search = IccFile::gteqToKey)
```

index

ファイル内のレコードを選択するために使用される **IccRecordIndex** オブジェクトを指すポインター。

search

IccFile で定義されている、検索の一致を検出するために使用する基準を示す列挙型。デフォルトは **gteqToKey** です。

条件

FILENOTFOUND、ILLOGIC、INVREQ、IOERR、ISCINVREQ、NOTAUTH、NOTFND、SYSIDERR

継承された public メソッド

これらは、このクラスによって継承された public メソッドです。

メソッド

actionOnCondition

クラス

IccResource

メソッド	クラス
actionOnConditionAsChar	IccResource
actionsOnConditionsText	IccResource
className	IccBase
classType	IccBase
condition	IccResource
conditionText	IccResource
customClassNum	IccBase
handleEvent	IccResource
id	IccResource
isEDFOn	IccResource
isRouteOptionOn	IccResource
name	IccResource
operator delete	IccBase
operator new	IccBase
routeOption	IccResource
setActionOnAnyCondition	IccResource
setActionOnCondition	IccResource
setActionsOnConditions	IccResource
setEDF	IccResource
setRouteOption	IccResource

継承された protected メソッド

これらは、このクラスによって継承された protected メソッドです。

メソッド	クラス
setClassName	IccBase
setCustomClassNum	IccBase

IccGroupId クラス

IccGroupId クラスは、CICS グループを識別するために使用されます。

IccBase
IccResourceId
IccGroupId

IccGroupId クラスは、CICS グループを識別するために使用されます。

ヘッダー・ファイル: ICCRIDEH

IccGroupId コンストラクター

コンストラクター (1)

IccGroupId(const char* groupName)

groupName

8 文字のグループ名。

コンストラクター (2)

コピー・コンストラクター。

IccGroupId(const IccGroupId& id)

id

IccGroupId オブジェクトへの参照。

public メソッド

これらは、このクラス内の public メソッドです。

operator= (1)

IccGroupId& operator=(const char* groupName)

groupName

8 文字のグループ名。

operator= (2)

新規値を代入します。

IccGroupId& operator=(const IccGroupId& id)

id

IccGroupId オブジェクトへの参照。

継承された public メソッド

これらは、このクラスによって継承された public メソッドです。

メソッド

クラス

classType

IccBase

className

IccBase

customClassNum

IccBase

メソッド	クラス
name	IccResourceId
nameLength	IccResourceId
operator delete	IccBase
operator new	IccBase

継承された protected メソッド

これらは、このクラスによって継承された protected メソッドです。

メソッド	クラス
operator=	IccResourceId
setClassName	IccBase
setCustomClassNum	IccBase

IccJournal クラス

IccJournal クラスは、ユーザーまたはシステム CICS ジャーナルを表します。

```

IccBase
  IccResource
    IccJournal
  
```

ヘッダー・ファイル: ICCJRNEH

サンプル: ICC\$JRN

IccJournal コンストラクター

コンストラクター (1)

```

IccJournal (const IccJournalId& id,
             unsigned long options = 0)
  
```

id

使用されるジャーナルを識別する **IccJournalId** オブジェクトへの参照。

options

このクラス内で定義されている **Options** 列挙型から構成される整数。**IccJournal** オブジェクトでの **writeRecord** 呼び出しの動作に影響します。値は、加算またはビット単位 OR することで、結合することができます。例えば、以下のとおりです。

```

IccJournal::startIO | IccJournal::synchronous
  
```

デフォルトでは、システム・デフォルトを使用します。

コンストラクター (2)

**IccJournal (unsigned short *journalNum*,
unsigned long *options* = 0)**

journalNum

ジャーナル番号 (1 から 99 の範囲)

options

上記を参照してください。

public メソッド

これらは、このクラス内の public メソッドです。

clearPrefix

registerPrefix または **setPrefix** によって設定された現行の接頭部をクリアします。現行の接頭部が **registerPrefix** を使用して設定された場合、**IccJournal** クラスは、自身が所有するその接頭部への参照のみを削除します。バッファ自体は変更されずに残ります。変更の接頭部が **setPrefix** によって設定された場合は、**IccJournal** のバッファのコピーが削除されます。

void clearPrefix()

journalTypeId

ジャーナル・レコードの発信元を識別するために使用される 2 バイトのフィールドが含まれる **IccJournalTypeId** オブジェクトへの参照を返します。

const IccJournalTypeId& journalTypeId() const

put

writeRecord と同義です。データをジャーナルに入れます。ポリモρφイズムについては、[ポリモρφフィック動作](#) を参照してください。

virtual void put(const IccBuf& *buffer*)

buffer

ジャーナルに入れるデータを保持している **IccBuf** オブジェクトへの参照。

registerPrefix

void registerPrefix(const IccBuf* *prefix*)

この **IccJournal** オブジェクトに対して **writeRecord** メソッドが呼び出された場合に使用する **prefix** オブジェクトを指すポインターを保管します。

setJournalTypeId (1)

void setJournalTypeId(const IccJournalTypeId& id)

setJournalTypeId (2)

ジャーナル・タイプ (2 バイトの ID) を設定します。これは、**writeRecord** メソッドを使用して作成したジャーナル・レコードに組み込まれます。

void setJournalTypeId(const char* jtypeid)

setPrefix (1)

void setPrefix(const IccBuf& prefix)

setPrefix (2)

void setPrefix(const char* prefix)

writeRecord メソッドが呼び出されたときに作成されるジャーナル・レコードに組み込めるように、*prefix* の *current* の内容を保管します。

wait

以前のジャーナル書き込みが完了するまで待機します。

**void wait (unsigned long requestNum=0,
 unsigned long option = 0)**

requestNum

書き込み要求。ゼロは、このジャーナルに対する最後の書き込みであることを示します。

option

IccJournal オブジェクトの **writeRecord** 呼び出しの動作に影響する整数。このクラス内で定義されている **Options** 列挙型から 0 以外の値を作成する必要があります。値は、加算またはビット単位 OR する (例えば、`IccJournal::startIO + IccJournal::synchronous`) ことで、結合することができます。デフォルトでは、システム・デフォルトを使用します。

writeRecord (1)

**unsigned long writeRecord (const IccBuf& record,
unsigned long option = 0)**

record

レコードを保持する **IccBuf** オブジェクトへの参照

option

上記を参照してください。

writeRecord (2)

レコード内のデータをジャーナルに書き込みます。返される数値は、特定の書き込み要求を表し、このクラス内の **wait** メソッドに渡すことができます。

**unsigned long writeRecord (const char* record,
unsigned long option = 0)**

record

レコードの名前

option

上記を参照してください。

条件

IOERR、JIDERR、LENGERR、NOJBUFSP、NOTAUTH、NOTOPEN

継承された public メソッド

これらは、このクラスによって継承された public メソッドです。

メソッド	クラス
actionOnCondition	IccResource
actionOnConditionAsChar	IccResource
actionsOnConditionsText	IccResource
classType	IccBase
className	IccBase
condition	IccResource
conditionText	IccResource
customClassNum	IccBase
handleEvent	IccResource
id	IccResource
isEDFOn	IccResource

メソッド	クラス
name	IccResource
operator delete	IccBase
operator new	IccBase
setActionOnAnyCondition	IccResource
setActionOnCondition	IccResource
setActionsOnConditions	IccResource
setEDF	IccResource

継承された protected メソッド

これらは、このクラスによって継承された protected メソッドです。

メソッド	クラス
setClassName	IccBase
setCustomClassNum	IccBase

列挙

オブション

IccJournal オブジェクトに対する **writeRecord** 呼び出しの動作。

これらの値は、加算あるいはビット単位 OR によって結合した整数にすることができます。

startIO

ジャーナル・レコードの出力を即時に開始することを指定します。使用頻度の低いジャーナルに「synchronous」を指定する場合は、ジャーナル・バッファが満杯になるまで要求側タスクが待機することを回避するために、「startIO」も指定してください。そのジャーナルが頻繁に使用される場合は、startIO は不要です。

noSuspend

NOJBUFSP 条件ではアプリケーション・プログラムを中断しないことを指定します。

synchronous

同期ジャーナル出力が必要であることを指定します。要求側タスクは、レコードが書き込まれるまで待機します。

IccJournalId クラス

IccJournalId は、CICS システム内のジャーナル番号を識別するために使用されます。

IccBase

IccResourceId

IccJournalId

ヘッダー・ファイル: ICCRIDEH

IccJournalId コンストラクター

コンストラクター (1)

IccJournalId(unsigned short *journalNum*)

journalNum

ジャーナルの番号 (1 から 99 の範囲)

コンストラクター (2)

コピー・コンストラクター。

IccJournalId(const IccJournalId& *id*)

id

IccJournalId オブジェクトへの参照。

public メソッド

これらは、このクラス内の public メソッドです。

number

ジャーナル番号を 1 から 99 の範囲で返します。

unsigned short number() const

operator= (1)

IccJournalId& operator=(unsigned short *journalNum*)

journalNum

ジャーナルの番号 (1 から 99 の範囲)

operator= (2)

新規値を代入します。

IccJournalId& operator=(const IccJournalId& *id*)

id

IccJournalId オブジェクトへの参照。

継承された public メソッド

これらは、このクラスによって継承された public メソッドです。

メソッド	クラス
classType	IccBase
className	IccBase
customClassNum	IccBase
name	IccResourceId
nameLength	IccResourceId
operator delete	IccBase
operator new	IccBase

継承された protected メソッド

これらは、このクラスによって継承された protected メソッドです。

メソッド	クラス
operator=	IccResourceId
setClassName	IccBase
setCustomClassNum	IccBase

IccJournalTypeId クラス

IccJournalTypeId クラス・オブジェクトは、ジャーナル・レコードの発信元を識別するために使用されます。これには、ジャーナル・レコードに含まれる 2 バイトのフィールドが含まれています。

IccBase
IccResourceId
IccJournalTypeId

IccJournalTypeId クラス・オブジェクトは、ジャーナル・レコードの発信元を識別するために使用されます。これには、ジャーナル・レコードに含まれる 2 バイトのフィールドが含まれています。

ヘッダー・ファイル: ICCRIDEH

IccJournalTypeId コンストラクター

コンストラクター (1)

IccJournalTypeId(const char* *journalTypeName*)

journalTypeName

ジャーナル・レコードで使用する 2 バイトの ID。

コンストラクター (2)

IccJournalTypeId(const IccJournalId& *id*)

id

IccJournalTypeId オブジェクトへの参照。

public メソッド

これらは、このクラス内の public メソッドです。

operator= (1)

void operator=(const IccJournalTypeId& *id*)

id

IccJournalTypeId オブジェクトへの参照。

operator= (2)

ジャーナル・レコードに組み込まれる 2 バイトのフィールドを設定します。

void operator=(const char* *journalTypeName*)

journalTypeName

ジャーナル・レコードで使用する 2 バイトの ID。

継承された public メソッド

これらは、このクラスによって継承された public メソッドです。

メソッド	クラス
classType	IccBase
className	IccBase
customClassNum	IccBase
name	IccResourceId
nameLength	IccResourceId
operator delete	IccBase
operator new	IccBase

継承された protected メソッド

これらは、このクラスによって継承された protected メソッドです。

メソッド	クラス
operator=	IccResourceId
setClassName	IccBase
setCustomClassNum	IccBase

IccKey クラス

IccKey クラスは、索引付き (KSDS) ファイルの検索キーを保持するために使用されます。

IccBase
IccRecordIndex
IccKey

ヘッダー・ファイル: ICCRECEH

サンプル: ICC\$FIL

IccKey コンストラクター

コンストラクター (1)

IccKey (**const char*** *initValue*,
Kind *kind* = complete)

コンストラクター (2)

IccKey (**unsigned short** *completeLength*,
Kind *kind* = complete)

コンストラクター (3)

IccKey(**const IccKey&** *key*)

public メソッド

これらは、このクラス内の public メソッドです。

assign
検索キーを **IccKey** オブジェクトにコピーします。

void assign (**unsigned short** *length*,
const void* *dataArea*)

length
データ域の長さ。

dataArea

検索キーを保持するデータ域の開始位置を指すポインター。

completeLength

キーが完全である場合の長さを返します。

unsigned short completeLength() const

kind

Kind kind() const

このクラス内で定義されている、キーが総称であるか完全であるかを示す列挙型を返します。

operator= (1)

IccKey& operator=(const IccKey& key)

operator= (2)

IccKey& operator=(const IccBuf& buffer)

operator= (3)

新規値をキーに代入します。

IccKey& operator=(const char* value)

operator== (1)

Icc::Bool operator==(const IccKey& key) const

operator== (2)

Icc::Bool operator==(const IccBuf& text) const

operator==(3)

等式をテストします。

Icc::Bool operator==(const char* *text*) const

operator!=(1)

Icc::Bool operator!=(const IccKey& *key*) const

operator!=(2)

Icc::Bool operator!=(const IccBuf& *text*) const

operator!=(3)

不等式をテストします。

Icc::Bool operator!=(const char* *text*) const

setKind

キーのタイプを総称から完全 (あるいはその逆) に変更します。

void setKind(Kind *kind*)

kind

このクラス内で定義されている、キーが総称であるか完全であることを示す列挙型。

value

const char* value()

検索キーが含まれるデータ域の開始位置を返します。

継承された public メソッド

これらは、このクラスによって継承された public メソッドです。

メソッド	クラス
className	IccBase
classType	IccBase
customClassNum	IccBase
length	IccRecordIndex
operator delete	IccBase
operator new	IccBase
type	IccRecordIndex
value	IccRecordIndex

継承された protected メソッド

これらは、このクラスによって継承された protected メソッドです。

メソッド	クラス
setClassName	IccBase
setCustomClassNum	IccBase

列挙

Kind

完全

指定されたキーが総称でないことを指定します。

総称

検索キーが総称であることを指定します。指定されたキーに接頭部が一致するキーを持つレコードが検出された場合、検索基準が満たされます。

IccLockId クラス

IccLockId クラスは、ロック要求を識別するために使用します。

IccBase

IccResourceId

IccLockId

IccLockId クラスは、ロック要求を識別するために使用します。

ヘッダー・ファイル: ICCRIDEH

IccLockId コンストラクター

コンストラクター (1)

IccLockId(const char* name)

name

ロック要求の 8 文字の名前。

コンストラクター (2)

コピー・コンストラクター。

IccLockId(const IccLockId& *id*)

id

IccLockId オブジェクトへの参照。

public メソッド

これらは、このクラス内の public メソッドです。

operator= (1)

IccLockId& operator=(const char* *name*)

name

ロック要求の 8 文字の名前。

operator= (2)

新規値を代入します。

IccLockId& operator=(const IccLockId& *id*)

id

IccLockId オブジェクトへの参照。

継承された public メソッド

これらは、このクラスによって継承された public メソッドです。

メソッド	クラス
classType	IccBase
className	IccBase
customClassNum	IccBase
name	IccResourceId
nameLength	IccResourceId
operator delete	IccBase

メソッド
operator new

クラス
IccBase

継承された protected メソッド

これらは、このクラスによって継承された protected メソッドです。

メソッド
operator=
setClassName
setCustomClassNum

クラス
IccResourceId
IccBase
IccBase

IccMessage クラス

IccMessage を使用して、メッセージ記述を保持することができます。

IccBase
IccMessage

これは主に **IccException** クラスによって、**IccException** オブジェクトの作成理由を説明するために使用されます。

ヘッダー・ファイル: ICCMSGEH

IccMessage コンストラクター

コンストラクター

**IccMessage (unsigned short *number*,
const char* *text*,
const char* *className* = 0,
const char* *methodName* = 0)**

number
メッセージに関連付けられた番号

text
メッセージに関連付けられたテキスト

className
メッセージに関連付けられたクラスのオプション名。

methodName
メッセージに関連付けられたメソッドのオプション名。

public メソッド

これらは、このクラス内の public メソッドです。

className
メッセージに関連付けられているクラスの名前を返します (存在する場合)。返す名前がない場合は、NULL ポインターが返されます。

const char* className() const

methodName

const char* methodName() const

メッセージが関連付けられているメソッドの名前を返します (存在する場合)。返す名前がない場合は、NULL ポインターが返されます。

number

unsigned short number() const

メッセージの番号を返します。

summary

const char* summary()

メッセージのテキストを返します。

text

const char* text() const

summary と同様にメッセージのテキストを返します。

継承された public メソッド

これらは、このクラスによって継承された public メソッドです。

メソッド	クラス
className	IccBase
classType	IccBase
customClassNum	IccBase
operator delete	IccBase
operator new	IccBase

継承された protected メソッド

これらは、このクラスによって継承された protected メソッドです。

メソッド	クラス
setClassName	IccBase
setCustomClassNum	IccBase

IccPartnerId クラス

IccPartnerId クラスは、CICS リモート (APPC) パートナー・トランザクション定義を表します。

IccBase

IccResourceId

IccPartnerId

IccPartnerId クラスは、CICS リモート (APPC) パートナー・トランザクション定義を表します。

ヘッダー・ファイル: ICCRIDEH

IccPartnerId コンストラクター

コンストラクター (1)

IccPartnerId(const char* *partnerName*)

partnerName

APPC パートナーの 8 文字の名前。

コンストラクター (2)

コピー・コンストラクター。

IccPartnerId(const IccPartnerId& *id*)

id

IccPartnerId オブジェクトへの参照。

public メソッド

operator= (1)

IccPartnerId& operator=(const char* *partnerName*)

partnerName

APPC パートナーの 8 文字の名前。

operator= (2)

新規値を代入します。

IccPartnerId& operator=(const IccPartnerId& *id*)

id

IccPartnerId オブジェクトへの参照。

継承された public メソッド

これらは、このクラスによって継承された public メソッドです。

メソッド	クラス
classType	IccBase
className	IccBase
customClassNum	IccBase
name	IccResourceId
nameLength	IccResourceId
operator delete	IccBase
operator new	IccBase

継承された protected メソッド

これらは、このクラスによって継承された protected メソッドです。

メソッド	クラス
operator=	IccResourceId
setClassName	IccBase
setCustomClassNum	IccBase

IccProgram クラス

IccProgram クラスは、現在実行されているプログラム (**IccControl** オブジェクトが表している) の外部の CICS プログラムを表します。

IccBase
IccResource
IccProgram

ヘッダー・ファイル: ICCPRGEH

サンプル: ICC\$PRG1、ICC\$PRG2、ICC\$PRG3

IccProgram コンストラクター

コンストラクター (1)

IccProgram(const IccProgramId& id)

id

IccProgramId オブジェクトへの参照。

コンストラクター (2)

IccProgram(const char* progName)

progName

8 文字のプログラム名。

public メソッド

opt パラメーター

多くのメソッドに、*opt* という同じパラメーターがあります。このパラメーターについては、[62 ページの『abendCode』](#)の **abendCode** メソッドの下で説明しています。

address

メモリー内のプログラム・モジュールのアドレスを返します。これは、**load** 呼び出しが正常に完了した後にのみ有効です。

const void* address() const

clearInputMessage

setInputMessage または **registerInputMessage** によって設定された現行の入力メッセージをクリアします。現行の入力メッセージが **registerInputMessage** を使用して設定された場合は、ポインターのみが削除されます。バッファは変更されずに残ります。現行の入力メッセージが **setInputMessage** を使用して設定された場合は、**clearInputMessage** はバッファが使用するメモリーを解放します。

void clearInputMessage()

entryPoint

const void* entryPoint() const

ロードされたプログラム・モジュールのエントリー・ポイントへのポインターを返します。これは、**load** 呼び出しが正常に完了した後にのみ有効です。

length

unsigned long length() const

プログラム・モジュールの長さを返します。これは、**load** 呼び出しが正常に完了した後にのみ有効です。

link

```
void link (const IccBuf* commArea = 0,  
          const IccTransId* transId = 0,  
          CommitOpt opt = noCommitOnReturn)
```

commArea

呼び出し側プログラムと呼び出されるプログラムの間で情報を受け渡すために使用されるバッファである COMMAREA が含まれる **IccBuf** オブジェクトを指すオプションのポインター。

transId

プログラムがリモート (DPL) プログラム・リンクである場合に、そのプログラムを実行するミラー・トランザクションの名前を示す **IccTransId** オブジェクトを指すオプションのポインター。

opt

このクラス内で定義されている列挙型で、プログラムがリモート (DPL) である場合にリンクの動作に影響を及ぼします。デフォルト (noCommitOnReturn) では、現行タスクがリソースをコミットするまで、リモート CICS 領域上のリソース変更はコミットされません。代替値 (commitOnReturn) は、このタスクがこの後に異常終了または問題を検出するかどうかにかかわらず、リモート・プログラムのリソースがコミットされることを意味します。

条件: INVREQ、NOTAUTH、PGMIDERR、SYSIDERR、LENGERR、ROLLEDBACK、TERMERR

制約事項

リンクはネストすることができます。つまり、リンク先のプログラムを別のプログラムにリンクすることができます。ただし、実装上の制限により、そのようなプログラムのネストは 15 回しかできません。これを超えると、例外がスローされます。

load

```
void load(LoadOpt opt = releaseAtTaskEnd)
```

opt

このクラス内で定義されている、CICS がタスクの終了時にプログラムがアンロードされるのを自動的に許可するか (releaseAtTaskEnd)、許可しないか (hold) を示す列挙型。

条件: NOTAUTH、PGMIDERR、INVREQ、LENGERR

registerInputMessage

link メソッドが呼び出されたときのために InputMessage へのポインターを保管します。

```
void registerInputMessage(const IccBuf& msg)
```

setInputMessage

このクラスで **link** メソッドを使用する場合に、呼び出されたプログラムから **IccSession::receive()** によって使用できるようにするデータを指定します。

void setInputMessage(const IccBuf& msg)

unload

プログラムのアンロードを許可します。 **load** を呼び出すことで再ロードすることができます。

void unload()

条件

NOTAUTH、PGMIDERR、INVREQ

継承された public メソッド

これらは、このクラスによって継承された public メソッドです。

メソッド	クラス
actionOnCondition	IccResource
actionOnConditionAsChar	IccResource
actionsOnConditionsText	IccResource
className	IccBase
classType	IccBase
condition	IccResource
conditionText	IccResource
customClassNum	IccBase
handleEvent	IccResource
id	IccResource
isEDFOn	IccResource
isRouteOptionOn	IccResource
name	IccResource
operator delete	IccBase
operator new	IccBase
routeOption	IccResource
setActionOnAnyCondition	IccResource
setActionOnCondition	IccResource
setActionsOnConditions	IccResource
setEDF	IccResource
setRouteOption	IccResource

継承された protected メソッド

これらは、このクラスによって継承された protected メソッドです。

メソッド	クラス
setClassName	IccBase
setCustomClassNum	IccBase

列挙

CommitOpt

noCommitOnReturn

リモート CICS 領域上のリソースに対する変更は、現行タスクがリソースをコミットするまでコミットされません。これはデフォルト設定値です。

commitOnReturn

リモート CICS 領域上のリソースに対する変更は、現行タスクが後で異常終了したり問題を検出したりするかどうかに関わらずコミットされます。

LoadOpt

releaseAtTaskEnd

CICS がタスクの終了時にプログラムのアンロードを自動的に許可することを示します。

hold

CICS がタスクの終了時にプログラムのアンロードを自動的に許可しないことを示します。(この場合、このタスクまたは別のタスクが明示的に **unload** メソッドを使用する必要があります)。

IccProgramId クラス

IccProgramId オブジェクトは、CICS システム内のプログラム名を表します。

IccBase

IccResourceId

IccProgramId

ヘッダー・ファイル: ICCRIDEH

IccProgramId コンストラクター

コンストラクター (1)

IccProgramId(const char* *progName*)

progName

8 文字のプログラム名。

コンストラクター (2)

コピー・コンストラクター。

IccProgramId(const IccProgramId& *id*)

id

IccProgramId オブジェクトへの参照。

public メソッド

operator= (1)

IccProgramId& operator=(const char* *progName*)

progName

8 文字のプログラム名。

operator= (2)

新規値を代入します。

IccProgramId& operator=(const IccProgramId& *id*)

id

IccProgramId オブジェクトへの参照。

継承された public メソッド

これらは、このクラスによって継承された public メソッドです。

メソッド	クラス
classType	IccBase
className	IccBase
customClassNum	IccBase
name	IccResourceId
nameLength	IccResourceId
operator delete	IccBase
operator new	IccBase

継承された protected メソッド

これらは、このクラスによって継承された protected メソッドです。

メソッド	クラス
operator=	IccResourceId
setClassName	IccBase
setCustomClassNum	IccBase

IccRBA クラス

IccRBA オブジェクトは、VSAM ESDS ファイルへのアクセスに使用される相対バイト・アドレスを保持します。

IccBase
IccRecordIndex
IccRBA

IccRBA オブジェクトは、VSAM ESDS ファイルへのアクセスに使用される相対バイト・アドレスを保持します。

ヘッダー・ファイル: ICCRECEH

IccRBA コンストラクター

コンストラクター

IccRBA(unsigned long *initRBA* = 0)

initRBA
相対バイト・アドレスの初期値。

public メソッド

operator= (1)

IccRBA& operator=(const IccRBA& *rba*)

operator= (2)
相対バイト・アドレスの新規値を代入します。

IccRBA& operator=(unsigned long *num*)

num
有効な相対バイト・アドレス。

operator== (1)

Icc::Bool operator== (const IccRBA& *rba*) const

operator== (2)
等式をテストします

Icc::Bool operator== (unsigned long *num*) const

operator!= (1)

Icc::Bool operator== (const IccRBA& *rba*) const

operator!= (2)
不等式をテストします

Icc::Bool operator!=(unsigned long *num*) const

number

unsigned long number() const
相対バイト・アドレスを返します。

継承された public メソッド

これらは、このクラスによって継承された public メソッドです。

メソッド	クラス
className	IccBase
classType	IccBase
customClassNum	IccBase
length	IccRecordIndex
operator delete	IccBase
operator new	IccBase
type	IccRecordIndex
value	IccRecordIndex

継承された protected メソッド

これらは、このクラスによって継承された protected メソッドです。

メソッド	クラス
setClassName	IccBase

メソッド
setCustomClassNum

クラス
IccBase

IccRecordIndex クラス

CICS ファイル制御レコード ID。

IccBase
IccRecordIndex
IccKey
IccRBA
IccRRN

CICS ファイル制御レコード ID。プログラムで取得、削除、または更新する特定のレコードを CICS に通知するために使用されます。**IccRecordIndex** は、**IccKey**、**IccRBA**、および **IccRRN** の派生元となる基本クラスです。

ヘッダー・ファイル: ICCRECEH

IccRecordIndex コンストラクター (protected)

コンストラクター

IccRecordIndex(Type type)

type

このクラス内で定義されている、索引タイプが key、RBA、または RRN のいずれであるかを示す列挙型。

注: **IccRecordIndex** オブジェクトを作成する必要があるため、これは保護されています。サブクラス **IccKey**、**IccRBA**、および **IccRRN** を参照してください。

public メソッド

length

レコード ID の長さを返します。

unsigned short length() const

type

Type type() const

このクラス内で定義されている、索引タイプが key、RBA、または RRN のいずれであるかを示す列挙型を返します。

継承された public メソッド

これらは、このクラスによって継承された public メソッドです。

メソッド	クラス
className	IccBase
classType	IccBase
customClassNum	IccBase
operator delete	IccBase
operator new	IccBase

継承された protected メソッド

これらは、このクラスによって継承された protected メソッドです。

メソッド	クラス
setClassName	IccBase
setCustomClassNum	IccBase

列挙

タイプ

Type は、アクセス方式を示します。

可能な値は次のとおりです。

- キー
- RBA
- RRN

IccRequestId クラス

IccRequestId は、要求の名前を保持するために使用されます。

IccBase
IccResourceId
IccRequestId

IccRequestId は、要求の名前を保持するために使用されます。この要求 ID は、後で要求を取り消すために使用することができます。例については、**IccStartRequestQ** クラスの **start** メソッドおよび **cancel** メソッドを参照してください。

ヘッダー・ファイル: ICCRIDEH

IccRequestId コンストラクター

コンストラクター (1)

空の **IccRequestId** オブジェクト。

IccRequestId()

コンストラクター (2)

IccRequestId(const char* *requestName*)

requestName

8 文字の要求名。

コンストラクター (3)

コピー・コンストラクター。

IccRequestId(const IccRequestId& *id*)

id

IccRequestId への参照。

public メソッド

operator= (1)

IccRequestId& operator=(const IccRequestId& *id*)

id

このオブジェクトへプロパティがコピーされる **IccRequestId** オブジェクトへの参照。

operator= (2)

新規値を代入します。

IccRequestId& operator=(const char* *requestName*)

requestName

このオブジェクトにコピーされる 8 文字のストリング。

継承された public メソッド

これらは、このクラスによって継承された public メソッドです。

メソッド	クラス
classType	IccBase
className	IccBase
customClassNum	IccBase
name	IccResourceId
nameLength	IccResourceId

メソッド	クラス
operator delete	IccBase
operator new	IccBase

継承された protected メソッド

これらは、このクラスによって継承された protected メソッドです。

メソッド	クラス
operator=	IccResourceId
setClassName	IccBase
setCustomClassNum	IccBase

IccResource クラス

IccResource クラスは、他のクラスを派生させるために使用する基本クラスです。

```
IccBase
  IccResource
```

IccResource に関連付けられたメソッドについてはここで説明されていますが、実際には、それらのメソッドは派生されたクラスのオブジェクトに対してのみ呼び出されます。

IccResource は、すべての CICS リソース、タスク、ファイル、プログラムなどの親クラスです。すべてのクラスが **IccBase** から継承されますが、CICS サービスを使用するクラスだけは **IccResource** から継承されます。

ヘッダー・ファイル: ICCRESEH

サンプル: ICC\$RES1、ICC\$RES2

IccResource コンストラクター (protected)

コンストラクター

IccResource(IccBase::ClassType classType)

classType

どのサブクラス・タイプであるかを示す列挙型。例えば、**IccTempStore** オブジェクトの場合、クラス・タイプは cTempStore です。可能な値は、**IccBase** クラスに関する説明の **ClassType** の項にリストされています。

public メソッド

actionOnCondition

CICS によって発生している指定された条件に対する応答で、クラスが実行するアクションを示す列挙型を返します。使用できる値については、このクラスで説明しています。

ActionOnCondition actionOnCondition(IccCondition::Codes condition)

condition

列挙型の条件名。可能な値のリストについては、**IccCondition** 構造体の項を参照してください。

actionOnConditionAsChar

char actionOnConditionAsChar(IccCondition::Codes condition)

このメソッドは **actionOnCondition** と同じですが、以下のように列挙型ではなく文字を返します。

0 (ゼロ)

この CICS 条件では、アクションは実行されません。

H

この CICS 条件では、仮想メソッド **handleEvent** が呼び出されます。

X

この CICS 条件では、例外が生成されます。

A

この CICS 条件では、プログラムが異常終了します。

actionsOnConditionsText

1 文字が 1 つの可能な条件を表している文字からなる文字列を返します。各文字は、その対応する条件に対して実行するアクションを示します。。

文字列で使用される文字については、[158 ページの『actionOnConditionAsChar』](#)で説明されています。例えば、文字列 0X00H0A ... は、最初の 7 個の条件に対するアクションを以下のように示しています。

条件 0 (NORMAL)

action=0 (noAction)

条件 1 (ERROR)

action=X (throwException)

条件 2 (RDATT)

action=0 (noAction)

条件 3 (WRBRK)

action=0 (noAction)

条件 4 (ICCEOF)

action=H (callHandleEvent)

条件 5 (EODS)

action=0 (noAction)

条件 6 (EOC)

action=A (abendTask)

const char* actionsOnConditionsText()

clear

オブジェクトの内容をクリアします。このメソッドは仮想メソッドであり、派生クラス内の適切な場所に実装されます。ポリモフィズムについての説明は、[ポリモフィック動作を参照してください](#)。このクラスのデフォルトの実装では、サブクラスでオーバーライドされていないことを示す例外をスローします。

virtual void clear()

condition

このオブジェクトによって作成された最新の CICS 呼び出しの条件コードを示す番号を返します。

unsigned long condition(ConditionType type = majorCode) const

type

要求された条件のタイプを示す、このクラスに定義された列挙。指定可能な値は majorCode (デフォルト) および minorCode です。

conditionText

const char* conditionText() const

このオブジェクトの最後の CICS 条件のシンボル名を返します。

get

virtual const IccBuf& get()

IccResource オブジェクトからデータを取得し、それを **IccBuf** 参照として返します。このメソッドは仮想メソッドであり、派生クラス内の適切な場所に実装されます。ポリモアフィズムについての説明は、[ポリモアフィック動作を参照してください](#)。このクラスでのデフォルトのインプリメンテーションは、サブクラス内でオーバーライドされていないことを示すために例外をスローします。

handleEvent

この仮想関数は、CICS イベント (ページ [107](#) ページの『[IccEvent クラス](#)』の **IccEvent** クラスを参照) を処理するために (アプリケーション・プログラマーによって) サブクラスに再実装することができます。

virtual HandleEventReturnOpt handleEvent(IccEvent& event)

event

このメソッドが呼び出されている理由を説明する **IccEvent** オブジェクトへの参照。

id

const IccResourceId* id() const

この **IccResource** オブジェクトに関連付けられている **IccResourceId** オブジェクトへのポインターを返します。

isEDFOn

Icc::Bool isEDFOn() const

EDF トレースがアクティブであるかどうかを示すブール値を返します。この値は yes または no のいずれかです。

isRouteOptionOn

Icc::Bool isRouteOptionOn() const

ルート・オプションがアクティブであるかどうかを示すブール値を返します。この値は yes または no のいずれかです。

name

const char* name() const

使用されるリソースの名前を示す文字列を返します。**IccTempStore** オブジェクトの場合、8 文字の一時記憶域キュー名が返されます。**IccTerminal** オブジェクトの場合、4 文字の端末名が返されます。これは、**id()**→**name** を呼び出すのと同様です。

put

バッファからの情報を **IccResource** オブジェクトに挿入します。このメソッドは仮想メソッドであり、派生クラス内の適切な場所に実装されます。ポリモフィズムについては、[ポリモフィック動作](#)を参照してください。このクラスのデフォルトの実装では、サブクラスでオーバーライドされていないことを示す例外をスローします。

virtual void put(const IccBuf& buffer)

buffer

オブジェクトに挿入するデータが入っている **IccBuf** オブジェクトへの参照。

routeOption

const IccSysId& routeOption() const

すべての CICS 要求が送信されるシステムを表す **IccSysId** オブジェクトへの参照を返します (明示的な機能シップ)。

setActionOnAnyCondition

CICS 条件が発生した場合に、CICS ファウンデーション・クラスによって実行されるデフォルト・アクションを指定します。

void setActionOnAnyCondition(ActionOnCondition action)

action

列挙型のアクション名。可能な値は、このクラスの説明の項にリストされています。

setActionOnCondition

指定された CICS 条件が発生した場合に、CICS ファウンデーション・クラスによって自動的に実行されるアクションを指定します。

```
void setActionOnCondition (ActionOnCondition action,  
                          IccCondition::Codes condition)
```

action

列挙型のアクション名。可能な値は、このクラスの説明の項にリストされています。

condition

IccCondition 構造体を参照してください。

setActionsOnConditions

```
void setActionsOnConditions(const char* actions = 0)
```

actions

それぞれの条件に対して実行するアクションを示す文字列。デフォルトでは、いずれのアクションも示されません。この場合、各条件に対して **ActionOnCondition** のデフォルトである **noAction** が指定されます。文字列は、**actionsOnConditionsText** メソッドによって返される文字列と同じフォーマットにする必要があります。

setEDF

このリソース・オブジェクトの EDF のオン/オフを切り替えます。これらのメソッドは、指定されたリモート・システムに CICS 要求を送信するようにオブジェクトに強制します。これは、明示的な機能シップと呼ばれます。

```
void setEDF(Icc::Bool onOff)
```

onOff

EDF トレースのオン/オフを切り替えるかどうかを選択するブール値。

setRouteOption (1)

このパラメーターは、以下のものです。

```
void setRouteOption(const IccSysId& sysId)
```

sysId

コマンドのルーティング先となるリモート・システムを表す **IccSysId** オブジェクト。

setRouteOption (2)

このオプションは、特定のクラスでのみ有効です。**IccResource** の他のサブクラスでこのメソッドを使用しようとすると、例外がスローされます。

有効なクラスは以下のとおりです。

- **IccDataQueue**
- **IccFile**
- **IccFileIterator**
- **IccProgram**
- **IccStartRequestQ**
- **IccTempStore**

ルート・オプションをオフにするには、例えば次のように、パラメーターを指定しません。

```
obj.setRouteOption()
```

void setRouteOption(const char* sysName = 0)

sysName

コマンドがルーティングされるシステムの 4 文字の名前。

継承された public メソッド

これらは、このクラスによって継承された public メソッドです。

メソッド	クラス
className	IccBase
classType	IccBase
customClassNum	IccBase
operator delete	IccBase
operator new	IccBase

継承された protected メソッド

これらは、このクラスによって継承された protected メソッドです。

メソッド	クラス
setClassName	IccBase
setCustomClassNum	IccBase

列挙

ActionOnCondition

可能な値は次のとおりです。

noAction

通常どおり続行。CICS サービスを呼び出すメソッドを実行した後に、**condition** メソッドを使用して CICS 条件をテストすることは、アプリケーション・プログラムが行う必要があります。

callHandleEvent

仮想 **handleEvent** メソッドを呼び出します。

throwException

IccException オブジェクトが作成され、スローされます。これは一般的に、より深刻な条件またはエラーの場合に使用されます。

abendTask

CICS タスクを異常終了します。

HandleEventReturnOpt

可能な値は次のとおりです。

rContinue

CICS イベントが十分に進行したため、通常の処理が再開されます。

rThrowException

アプリケーション・プログラムが CICS イベントを処理できなかったため、例外がスローされます。

rAbendTask

アプリケーション・プログラムが CICS イベントを処理できなかったため、CICS タスクが異常終了します。

ConditionType

可能な値は次のとおりです。

majorCode

戻り値は CICS RESP 値です。これは、IccCondition::codes 内の値の 1 つです。

minorCode

戻り値は CICS RESP2 値です。

IccResourceId クラス

これは、**IccTransId** およびその他の名前の末尾が「Id」であるすべてのクラスの派生元となる基本クラスです。

IccBase**IccResourceId**

これらの派生クラスの多くは、CICS リソース名を表します。

ヘッダー・ファイル: ICCRIDEH

IccResourceId コンストラクター (protected)

コンストラクター (1)

**IccResourceId (IccBase::ClassType typ,
const IccResourceId& id)**

type

IccBase クラス内で定義されている列挙型の 1 つで、クラスのタイプを示します。

id

このオブジェクトを作成するために使用する **IccResourceId** オブジェクトへの参照。

コンストラクター (2)

**IccResourceId (IccBase::ClassType type,
const char* resName)**

type

IccBase クラス内で定義されている列挙型の 1 つで、クラスのタイプを示します。

resName

このオブジェクトを作成するために使用されるリソースの名前。

public メソッド

これらは、このクラス内の public メソッドです。

name

リソース ID の名前をストリングとして返します。ほとんどの **...Id** オブジェクトの名前は、4 文字または 8 文字です。

const char* name() const

nameLength

unsigned short nameLength() const

name メソッドによって返される名前の長さを返します。

protected メソッド

operator=

IccResourceId オブジェクトを *id* と同じ値に設定します。

IccResourceId& operator=(const IccResourceId& id)

id

IccResourceId オブジェクトへの参照。

継承された public メソッド

これらは、このクラスによって継承された public メソッドです。

メソッド

クラス

className

IccBase

classType

IccBase

メソッド	クラス
customClassNum	IccBase
operator delete	IccBase
operator new	IccBase

継承された protected メソッド

これらは、このクラスによって継承された protected メソッドです。

メソッド	クラス
setClassName	IccBase
setCustomClassNum	IccBase

IccRRN クラス

IccRRN オブジェクトは、相対レコード番号を保持し、VSAM RRDS ファイル内のレコードを識別するために使用されます。

IccBase
IccRecordIndex
IccRRN

IccRRN オブジェクトは、相対レコード番号を保持し、VSAM RRDS ファイル内のレコードを識別するために使用されます。

ヘッダー・ファイル: ICCRECEH

IccRRN コンストラクター

コンストラクター

IccRRN(unsigned long *initRRN* = 1)

initRRN

初期相対レコード番号 (0 より大きい整数)。デフォルトは、1 です。

public メソッド

これらは、このクラス内の public メソッドです。

operator= (1)

IccRRN& operator=(const IccRRN& *rrn*)

operator= (2)

相対レコード番号の新規値を代入します。

IccRRN& operator=(unsigned long *num*)

num

相対レコード番号 (0 より大きい整数)。

operator== (1)

Icc::Bool operator== (const IccRRN& *rrn*) const

operator== (2)

等式をテストします

Icc::Bool operator== (unsigned long *num*) const

operator!= (1)

Icc::Bool operator!= (const IccRRN& *rrn*) const

operator!= (2)

不等式をテストします

Icc::Bool operator!=(unsigned long *num*) const

number

unsigned long number() const

相対レコード番号を返します。

継承された public メソッド

これらは、このクラスによって継承された public メソッドです。

メソッド

クラス

className

IccBase

classType

IccBase

customClassNum

IccBase

length

IccRecordIndex

メソッド	クラス
operator delete	IccBase
operator new	IccBase
type	IccRecordIndex
value	IccRecordIndex

継承された protected メソッド

これらは、このクラスによって継承された protected メソッドです。

メソッド	クラス
setClassName	IccBase
setCustomClassNum	IccBase

IccSemaphore クラス

このクラスは、リソース更新の同期を使用可能にします。

IccBase
IccResource
IccSemaphore

ヘッダー・ファイル: ICCSEMEH

サンプル: ICC\$SEM

IccSemaphore コンストラクター

コンストラクター (1)

**IccSemaphore (const char* *resource*,
LockType *type* = byValue,
LifeTime *life* = UOW)**

resource

type が byValue である場合はテキスト・ストリング。それ以外の場合はストレージ内のアドレス。

type

このクラス内で定義されている、ロックが値によるものであるかアドレスによるものであるかを示す列挙型。デフォルトは値によるものです。

life

このクラス内で定義されている、セマフォが持続する時間の長さを示す列挙型。デフォルトでは、UOW の長さだけ持続します。

コンストラクター (2)

**IccSemaphore (const IccLockId& *id*,
LifeTime *life* = UOW)**

id

IccLockId オブジェクトへの参照

life

このクラス内で定義されている、セマフォが持続する時間の長さを示す列挙型。デフォルトでは、UOW の長さだけ持続します。

public メソッド

これらは、このクラス内の public メソッドです。

lifeTime

このクラス内で定義されている、ロックが現行の作業単位の長さだけ持続するか (UOW)、タスクが終了するまで持続するか (task) を示す列挙型を返します。

LifeTime lifeTime() const

lock

void lock()

ロックの取得を試行します。このメソッドは、別のタスクが既にロックを所有している場合はブロックします。

条件

ENQBUSY、LENGERR、INVREQ

tryLock

ロックの取得を試行します。このメソッドは、別のタスクが既にロックを所有している場合はブロックしません。成功したかどうかを示すブール値を返します。

Icc::Bool tryLock()

条件

ENQBUSY、LENGERR、INVREQ

type

このクラス内で定義されている、セマフォのタイプを示す列挙型を返します。

LockType type() const

unlock

void unlock()

ロックを解放します。

条件

LENGERR、INVREQ

継承された public メソッド

これらは、このクラスによって継承された public メソッドです。

メソッド	クラス
actionOnCondition	IccResource
actionOnConditionAsChar	IccResource
actionsOnConditionsText	IccResource
classType	IccBase
className	IccBase
condition	IccResource
conditionText	IccResource
customClassNum	IccBase
handleEvent	IccResource
id	IccResource
isEDFOn	IccResource
name	IccResource
operator delete	IccBase
operator new	IccBase
setActionOnAnyCondition	IccResource
setActionOnCondition	IccResource
setActionsOnConditions	IccResource
setEDF	IccResource

継承された protected メソッド

これらは、このクラスによって継承された protected メソッドです。

メソッド	クラス
setClassName	IccBase
setCustomClassNum	IccBase

列挙

LockType

byValue

ロックは内容 (例えば、名前) に掛かっています。

byAddress

ロックはメモリー・アドレスに掛かっています。

LifeTime

UOW

セマフォは現行作業単位の長さだけ持続します。

タスク

セマフォはタスクの長さだけ持続します。

IccSession クラス

このクラスは、APPC および DTP プログラミングを使用可能にします。

IccBase

IccResource

IccSession

ヘッダー・ファイル: ICCSESEH

サンプル: ICC\$SES1、ICC\$SES2

IccSession コンストラクター (public)

コンストラクター (1)

IccSession(const IccPartnerId& id)

id

IccPartnerId オブジェクトへの参照

コンストラクター (2)

**IccSession (const IccSysId& sysId,
const char* profile = 0)**

sysId

リモート CICS システムを表す **IccSysId** オブジェクトへの参照

profile

8 文字のプロファイル名。

コンストラクター (3)

**IccSession (const char* sysName,
const char* profile = 0)**

sysName

このセッションが関連付けられるリモート CICS システムの 4 文字の名前。

profile

8 文字のプロファイル名。

IccSession コンストラクター (protected)

コンストラクター

このコンストラクターは、基本機能としてセッションを持っているバックエンドの DTP CICS タスク用です。この場合、アプリケーション・プログラムは、**IccControl** オブジェクトに対する **session** メソッドを使用して、**IccSession** オブジェクトにアクセスします。

IccSession()

public メソッド

これらは、このクラス内の public メソッドです。

allocate

リモート・システムへのセッション (通信チャネル) を確立します。

void allocate(AllocateOpt option = queue)

option

このクラス内で定義されている、このメソッドの呼び出し時に通信チャネルが使用できない場合に CICS が実行するアクションを示す列挙型。

条件

INVREQ、SYSIDERR、CBIDERR、NETNAMEIDERR、PARTNERIDERR、SYSBUSY

connectProcess (1)

このメソッドは、このセッション・オブジェクトの構成に **IccPartnerId** オブジェクトが使用された場合にのみ使用することができます。

**void connectProcess (SyncLevel level,
const IccBuf* PIP = 0)**

level

このクラス内で定義されている、この会話に使用される同期レベルを示す列挙型。

PIP

リモート・システムに送信される PIP データが含まれている **IccBuf** オブジェクトへのオプション・ポインター。

connectProcess (2)

```
void connectProcess (SyncLevel level,  
                    const IccTransId& transId,  
                    const IccBuf* PIP = 0)
```

level

このクラス内で定義されている、この会話に使用される同期レベルを示す列挙型。

transId

リモート・システムで開始されるトランザクションの名前を保持している **IccTransId** オブジェクトへの参照。

PIP

リモート・システムに送信される PIP データが含まれている **IccBuf** オブジェクトへのオプション・ポインター。

connectProcess (3)

情報を送受信するための準備として、リモート・システム上でパートナー・プロセスを開始します。

```
void connectProcess (SyncLevel level,  
                    const IccTPNameId& TPName,  
                    const IccBuf* PIP = 0)
```

level

このクラス内で定義されている、この会話に使用される同期レベルを示す列挙型。

TPName

1 から 64 文字の TP 名が含まれている **IccTPNameId** オブジェクトへの参照。

PIP

リモート・システムに送信される PIP データが含まれている **IccBuf** オブジェクトへのオプション・ポインター。

条件

INVREQ、LENGERR、NOTALLOC、PARTNERIDERR、NOTAUTH、
TERMERR、SYSBUSY

converse

converse は、*send* の内容を送信し、リモート APPC パートナーからの返信を保持している **IccBuf** オブジェクトへの参照を返します。

```
const IccBuf& converse(const IccBuf& send)
```

send

送信するデータを格納している **IccBuf** オブジェクトへの参照。

条件

EOC、INVREQ、LENGERR、NOTALLOC、SIGNAL、TERMERR

convId

4 バイトの会話 ID が含まれている **IccConvId** オブジェクトへの参照を返します。

const IccConvId& convId()

errorCode

const char* errorCode() const

isErrorSet が true を返した場合に受信する 4 バイトのエラー・コードを返します。詳細については関連する DTP のガイドを参照してください。

extractProcess

void extractProcess()

APPC 会話接続ヘッダーから情報を取得し、その情報をオブジェクト内部に保持します。オブジェクトから情報を取り出すには、**PIPList**、**process**、および **syncLevel** の各メソッドを参照してください。このメソッドは、バックエンド・タスクで PIP データ、プロセス名、またはエンド・タスクが実行されている同期レベルにアクセスしたい場合に使用してください。

条件

INVREQ、NOTALLOC、LENGERR

flush

累積データおよび制御情報が APPC マップ式会話で伝送されるようにします。

void flush()

条件

INVREQ、NOTALLOC

free

CICS に APPC セッションを返し、他のタスクで使用できるようにします。

void free()

条件

INVREQ、NOTALLOC

get

receive の同義語です。ポリモフィズムについては、[ポリモフィック動作](#) を参照してください。

virtual const IccBuf& get()

isErrorSet

Icc::Bool isErrorSet() const

Icc 構造で定義されている、エラーが設定されているかどうかを示すブール変数を返します。

isNoDataSet

Icc::Bool isNoDataSet() const

Icc 構造で定義されている、**send** に対して制御情報のみでデータが返されなかったかどうかを示すブール変数を返します。

isSignalSet

Icc::Bool isSignalSet() const

Icc 構造で定義されている、リモート・プロセスからシグナル通知を受信したかどうかを示すブール変数を返します。

issueAbend

void issueAbend()

会話を異常終了します。パートナー・トランザクションには TERMERR 状態が発生します。

条件

INVREQ、NOTALLOC、TERMERR

issueConfirmation

確認オプションを指定したパートナーの **send** 要求に肯定応答を送信します。

void issueConfirmation()

条件

INVREQ、NOTALLOC、TERMERR、SIGNAL

issueError

パートナー・プロセスにエラーをシグナル通知します。

void issueError()

条件

INVREQ、NOTALLOC、TERMERR、SIGNAL

issuePrepare

これは、APPC リンク上の DTP にのみ適用されます。これを使用すると、同期点開始プログラムは、同期点交換の最初のフロー (準備からコミットへ) だけを送信することにより、同期点処理のための同期点着信側を準備することができます。

void issuePrepare()

条件

INVREQ、NOTALLOC、TERMERR

issueSignal

モード変更が必要であることをシグナル通知します。

void issueSignal()

条件

INVREQ、NOTALLOC、TERMERR

PIPList

フロントエンド・プロセスから送信された PIP データが含まれている **IccBuf** オブジェクトへの参照を返します。このメソッドは、バックエンド DTP プロセスに対する **extractProcess** を呼び出した後に呼び出してください。

IccBuf& PIPList()

process

const IccBuf& process() const

フロントエンド・プロセスから送信されたプロセス・データが含まれている **IccBuf** オブジェクトへの参照を返します。このメソッドは、バックエンド DTP プロセスに対する **extractProcess** を呼び出した後に呼び出してください。

put

send と同義です。ポリモアフィズムについては、[ポリモアフィック動作](#) を参照してください。

virtual void put(const IccBuf& data)

data

リモート・プロセスに送信するデータを保持している **IccBuf** オブジェクトへの参照。

受信

const IccBuf& receive()

リモート・システムから受信したデータが含まれている **IccBuf** オブジェクトへの参照を返します。

条件

EOC、INVREQ、LENGERR、NOTALLOC、SIGNAL、TERMERR

send (1)

**void send (const IccBuf& send,
SendOpt option = normal)**

send

送信するデータを格納している **IccBuf** オブジェクトへの参照。

option

このクラス内で定義されている、**send** メソッドの動作に影響する列挙型。デフォルトは normal です。

send (2)

リモート・パートナーにデータを送信します。

void send(SendOpt option = normal)

option

このクラス内で定義されている、**send** メソッドの動作に影響する列挙型。デフォルトは normal です。

条件

INVREQ、LENGERR、NOTALLOC、SIGNAL、TERMERR

sendInvite (1)

**void sendInvite (const IccBuf& send,
SendOpt option = normal)**

send

送信するデータを格納している **IccBuf** オブジェクトへの参照。

option

このクラス内で定義されている、**sendInvite** メソッドの動作に影響する列挙型。デフォルトは **normal** です。

sendInvite (2)

リモート・パートナーにデータを送信し、方向転換を指示します。つまり、このオブジェクトに対する次のメソッドは **receive** になります。

void sendInvite(SendOpt option = normal)

option

このクラス内で定義されている、**sendInvite** メソッドの動作に影響する列挙型。デフォルトは **normal** です。

条件

INVREQ、LENGERR、NOTALLOC、SIGNAL、TERMERR

sendLast (1)

**void sendLast (const IccBuf& send,
SendOpt option = normal)**

send

送信するデータを格納している **IccBuf** オブジェクトへの参照。

option

このクラス内で定義されている、**sendLast** メソッドの動作に影響する列挙型。デフォルトは **normal** です。

sendLast (2)

リモート・パートナーにデータを送信し、これが最後の伝送であることを示します。**free** メソッドを次に呼び出す必要がありますが、同期レベルが 2 の場合は除きます。その場合は、**free** の前にリソースの更新をコミットする必要があります。(IccTaskClass のページ [194 ページ](#)の『commitUOW』の **commitUOW** を参照してください)。

void sendLast(SendOpt option = normal)

option

このクラス内で定義されている、**sendLast** メソッドの動作に影響する列挙型。デフォルトは normal です。

条件

INVREQ、LENGERR、NOTALLOC、SIGNAL、TERMERR

状態

IccValue 構造で定義されている、APPC 会話の現行状態を示す CVDA を返します。

可能な値は次のとおりです。

- ALLOCATED
- CONFFREE
- CONFSEND
- FREE
- PENDFREE
- PENDRECEIVE
- RECEIVE
- ROLLBACK
- SEND
- SYNCFREE
- SYNCRECEIVE
- SYNCSEND
- NOTAPPLIC

APPC 会話の状態が存在しない場合は、IccValue::NOTAPPLIC が返されます。

IccValue::CVDA state(StateOpt option = lastCommand)

option

このクラス内で定義されている、会話の状態を報告する方法を示す列挙型

条件

INVREQ、NOTALLOC

stateText

state メソッドが返す状態のシンボル名を返します。例えば、**state** が IccValue::ALLOCATED を返す場合、**stateText** は「ALLOCATED」を返します。

const char* stateText(StateOpt option = lastCommand)

option

このクラス内で定義されている、会話の状態を報告する方法を示す列挙型

syncLevel

SyncLevel syncLevel() const

このクラス内で定義されている、このセッションで使用されている同期レベルを示す列挙型を返します。
このメソッドは、バックエンド DTP プロセスに対する **extractProcess** を呼び出した後に呼び出してください。

継承された public メソッド

これらは、このクラスによって継承された public メソッドです。

メソッド	クラス
actionOnCondition	IccResource
actionOnConditionAsChar	IccResource
actionsOnConditionsText	IccResource
classType	IccBase
className	IccBase
condition	IccResource
conditionText	IccResource
customClassNum	IccBase
handleEvent	IccResource
id	IccResource
isEDFOn	IccResource
name	IccResource
operator delete	IccBase
operator new	IccBase
setActionOnAnyCondition	IccResource
setActionOnCondition	IccResource
setActionsOnConditions	IccResource
setEDF	IccResource

継承された protected メソッド

これらは、このクラスによって継承された protected メソッドです。

メソッド	クラス
setClassName	IccBase
setCustomClassNum	IccBase

列挙

AllocateOpt

queue

使用可能なすべてのセッションが使用中である場合、CICS はセッションを割り振ることができるまで、この要求をキューに入れます (そしてメソッドをブロックします)。

noQueue

セッションを割り振ることができない場合は、制御がアプリケーションへ戻されます。CICS で SYSBUSY 条件が発生します。

allocate メソッドでキューイングが必要かどうかを示します。

SendOpt

normal

デフォルト。

confirmation

SyncLevel level1 または level2 を使用するプログラムにリモート・パートナー・プログラムからの応答が必要であることを示します。リモート・パートナーは、**issueConfirmation** メソッドを使用して肯定的に応答することも、**issueError** メソッドを使用して否定的に応答することもできます。送信側プログラムは、応答を受信するまでは CICS から制御を返されません。

wait

データの送信と、それを内部のバッファーに入れないことを要求します。このオプションを指定しない場合、CICS は、パフォーマンスを向上させるために、要求を自由にバッファーに入れることができます。

StateOpt

StateOpt は、会話の状態をどのように報告するかを示すために使用されます。

lastCommand

セッションで最後の操作が完了した時点の状態を返します。

extractState

明示的に抽出された現在の状態を返します。

SyncLevel

level0

同期レベル 0

level1

同期レベル 1

level2

同期レベル 2

IccStartRequestQ クラス

これは、アプリケーション・プログラマーが別の CICS トランザクションの非同期の開始を要求することを可能にする singleton クラスです。

IccBase

IccResource

IccStartRequestQ

(ページ [184](#) ページの『[start](#)』の **start** メソッドを参照してください)。

非同期に開始されるトランザクションは、**IccStartRequestQ** クラスのメソッド **retrieveData** を使用して、**start** 要求を発行したトランザクションから渡される情報を取得します。

cancel メソッドを使用することで、期限切れになっていない開始要求を取り消すことができます。

ヘッダー・ファイル : ICCSRQEH

サンプル : ICC\$SRQ1、ICC\$SRQ2

IccStartRequestQ コンストラクター (protected)

コンストラクター

IccStartRequestQ()

public メソッド

これらは、このクラス内の public メソッドです。

cancel

以前に発行され、まだ期限切れになっていない **start** 要求を取り消します。

```
void cancel (const IccRequestId& reqId,  
             const IccTransId* transId = 0)
```

reqId

取り消される要求を表す **IccRequestId** オブジェクトへの参照。

transId

取り消されるトランザクションを表す **IccTransId** オブジェクトへのオプション・ポインター。

条件

ISCINVREQ、NOTAUTH、NOTFND、SYSIDERR

clearData

clearData は、開始されるトランザクションに渡される現行データをクリアします。

void clearData()

データは、**setData** または **registerData** を使用して設定されました。

registerData を使用してデータが設定された場合は、データへのポインターのみが削除され、バッファ内のデータは変更されずに残ります。

setData を使用してデータが設定された場合、**clearData** はバッファが使用するメモリーを解放します。

データ

開始要求時に渡されたデータが含まれている **IccBuf** オブジェクトへの参照を返します。このメソッドは **retrieveData** メソッドを呼び出した後に呼び出してください。

```
const IccBuf& data() const
```

instance

static IccStartRequestQ* instance()

単一の **IccStartRequestQ** オブジェクトを指すポインターを返します。オブジェクトが存在しない場合は作成されます。 **IccControl** のページ [100 ページ](#) の『**startRequestQ**』の **startRequestQ** メソッドも参照してください。

queueName

const char* queueName() const

開始要求元によって渡されたキューの名前を返します。このメソッドは、**retrieveData** メソッドを呼び出した後に呼び出してください。

registerData

後続の **start** メソッドの呼び出しごとに開始データの問い合わせ先となる **IccBuf** オブジェクトを登録します。これは、**start** メソッドの使用時に **IccBuf** オブジェクトを見つけることができるように、**IccStartRequestQ** 内に **IccBuf** オブジェクトのアドレスを保管するだけです。これは、呼び出されたときに **IccBuf** オブジェクト内に保持されているデータのコピーを取得する **setData** メソッドとは異なります。

void registerData(const IccBuf* buffer)

buffer

start 要求時に渡されるデータを保持する **IccBuf** オブジェクトへのポインター。

reset

void reset()

このクラス内で以前に **set...** メソッドによって行われた関連付けをすべてクリアします。

retrieveData

非同期開始要求によって開始されたタスクが、開始要求元から渡された情報にアクセスするために使用します。情報は、**data**、**queueName**、**returnTermId**、および **returnTransId** の各メソッドによって返されます。

void retrieveData(RetrieveOpt option = noWait)

option

このクラス内で定義されている、使用可能な開始データがない場合に何が起きるかを示す列挙型。

条件

ENDDATA、ENVDEFERR、IOERR、LENGERR、NOTFND、INVREQ

注 : **start** メソッドを発行する前に可能なすべてのオプション (**setData**、**setQueueName**、**setReturnTermId**、および **setReturnTransId**) が使用されていない場合に、ENVDEFERR 条件が発生します。したがって、この条件は必ずしもエラー条件ではなく、プログラムではそれに応じた処理が必要です。

returnTermId

セッションに關与する端末を識別する **IccTermId** オブジェクトへの参照を返します。このメソッドは **retrieveData** メソッドを呼び出した後に呼び出してください。

const IccTermId& returnTermId() const

returnTransId

const IccTransId& returnTransId() const

開始要求時に渡される **IccTransId** オブジェクトへの参照を返します。このメソッドは、**retrieveData** メソッドを呼び出した後に呼び出してください。

setData

void setData(const IccBuf& buf)

buf 内のデータを **IccStartRequestQ** にコピーします。そのデータは、**start** メソッドが呼び出されたときに、開始されるトランザクションに渡されます。開始されるトランザクションにデータを渡すための代替方法については、[182 ページの『registerData』](#)のページの **registerData** も参照してください。

setQueueName

start メソッドが呼び出されたときに、このキュー名を開始されるトランザクションに渡すように要求します。

void setQueueName(const char* queueName)

queueName

8 文字のキュー名。

setReturnTermId (1)

void setReturnTermId(const IccTermId& termId)

termId

セッションに關与する端末を識別する **IccTermId** オブジェクトへの参照。

setReturnTermId (2)

start メソッドが呼び出されたときに、開始されるトランザクションにこの戻り端末 ID を渡すように要求します。

void setReturnTermId(const char* termName)

termName

セッションに関連している端末の 4 文字の名前。

setReturnTransId (1)

```
void setReturnTransId(const IccTransId& transId)
```

transId

IccTransId オブジェクトへの参照。

setReturnTransId (2)

start メソッドが呼び出されたときに、開始されるトランザクションにこの戻りトランザクション ID を渡すように要求します。

```
void setReturnTransId(const char* transName)
```

transName

戻りトランザクションの 4 文字の名前。

setStartOpts

開始されるトランザクションを保護するかどうか、および検査するかどうかを設定します。

```
void setStartOpts (ProtectOpt popt = none,  
                  CheckOpt copt = check)
```

popt

このクラス内で定義されている、開始要求を保護するかどうかを示す列挙型。

copt

このクラス内で定義されている、開始要求を検査するかどうかを示す列挙型。

start

指定された CICS トランザクションを非同期に開始します。返される **IccRequestId** オブジェクトへの参照は、**start** 要求を識別し、後でその参照を使用して、**start** 要求を **cancel** することができます。

```
const IccRequestId& start (const IccTransId& transId,  
                          const IccTermId* termId,  
                          const IccTime* time = 0,  
                          const IccRequestId* reqId = 0)
```

または

```
const IccRequestId& start (const IccTransId& transId,
                           const IccUserId* userId,
                           const IccTime* time = 0,
                           const IccRequestId* reqId = 0)
```

または

```
const IccRequestId& start (const IccTransId& transId,
                           const IccTime* time = 0,
                           const IccRequestId* reqId = 0)
```

transId

開始されるトランザクションを表す **IccTransId** オブジェクトへの参照。

termId

セッションに関与する端末を識別する **IccTermId** オブジェクトへの参照。

userId

ユーザー ID を表す **IccUserId** オブジェクトへの参照。

time

タスクの開始時期を指定する **IccTime** オブジェクトを指す (オプションの) ポインター。デフォルトでは、タスクは即時に開始されます。

reqId

cancel で要求を取り消すことができるように、この開始要求の識別に使用される **IccRequestId** オブジェクトへの (オプション) ポインター。

条件

INVREQ、IOERR、ISCINVREQ、LENGERR、NOTAUTH、SYSIDERR、TERMIDERR、TRANSIDERR、USERIDERR

継承された public メソッド

これらは、このクラスによって継承された public メソッドです。

メソッド	クラス
actionOnCondition	IccResource
actionOnConditionAsChar	IccResource
actionsOnConditionsText	IccResource
className	IccBase
classType	IccBase
condition	IccResource
conditionText	IccResource
customClassNum	IccBase
handleEvent	IccResource
id	IccResource
isEDFOn	IccResource
isRouteOptionOn	IccResource
name	IccResource
operator delete	IccBase

メソッド	クラス
operator new	IccBase
routeOption	IccResource
setActionOnAnyCondition	IccResource
setActionOnCondition	IccResource
setActionsOnConditions	IccResource
setEDF	IccResource
setRouteOption	IccResource

継承された protected メソッド

これらは、このクラスによって継承された protected メソッドです。

メソッド	クラス
setClassName	IccBase
setCustomClassNum	IccBase

列挙

RetrieveOpt

- noWait
- wait

ProtectOpt

- なし
- protect

CheckOpt

- check
- noCheck

IccSysId クラス

IccSysId クラスは、リモート CICS システムを識別するために使用されます。

IccBase

IccResourceId

IccSysId

IccSysId クラスは、リモート CICS システムを識別するために使用されます。

ヘッダー・ファイル: ICCRIDEH

IccSysId コンストラクター

コンストラクター (1)

IccSysId(const char* name)

name

CICS システムの 4 文字の名前。

コンストラクター (2)

コピー・コンストラクター。

IccSysId(const IccSysId& *id*)

id

IccSysId オブジェクトへの参照。

public メソッド

これらは、このクラス内の public メソッドです。

operator= (1)

IccSysId& operator=(const IccSysId& *id*)

id

既存の **IccSysId** オブジェクトへの参照。

operator= (2)

オブジェクト内に保持される CICS システムの名前を設定します。

IccSysId& operator=(const char* *name*)

name

CICS システムの 4 文字の名前。

継承された public メソッド

メソッド	クラス
classType	IccBase
className	IccBase
customClassNum	IccBase
name	IccResourceId
nameLength	IccResourceId
operator delete	IccBase
operator new	IccBase

継承された protected メソッド

これらは、このクラスによって継承された protected メソッドです。

メソッド	クラス
operator=	IccResourceId
setClassName	IccBase
setCustomClassNum	IccBase

IccSystem クラス

これは、CICS システムを表す singleton クラスです。これは、アプリケーション・プログラムが、自身が実行されている CICS システムに関する情報を検出するために使用します。

IccBase
IccResource
IccSystem

ヘッダー・ファイル: ICCSYSEH

サンプル: ICC\$SYS

IccSystem コンストラクター (protected)

コンストラクター

IccSystem()

public メソッド

これらは、このクラス内の public メソッドです。

applName
CICS 領域の 8 文字の名前を返します。

const char* applName()

条件

INVREQ

beginBrowse (1)

**void beginBrowse (ResourceType *resource*,
const IccResourceId* *resId* = 0)**

resource

このクラス内で定義されている、CICS システム内でブラウズされるリソースのタイプを示す列挙型。

resId

リソースに対するブラウズの開始点を示す **IccResourceId** オブジェクトへのオプション・ポインター。

beginBrowse (2)

一連の CICS リソースに対するブラウズの開始をシグナル通知します。

```
void beginBrowse (ResourceType resource,  
                  const char* resName)
```

resource

このクラス内で定義されている、CICS システム内でブラウズされるリソースのタイプを示す列挙型。

resName

リソースのブラウズの開始点となるリソースの名前。

条件

END、FILENOTFOUND、ILLOGIC、NOTAUTH

dateFormat

CICS 領域のデフォルトの日付形式を返します。

```
const char* dateFormat()
```

条件

INVREQ

endBrowse

一連の CICS リソースに対するブラウズの終了をシグナル通知します。

```
void endBrowse(ResourceType resource)
```

条件

END、FILENOTFOUND、ILLOGIC、NOTAUTH

freeStorage

IccSystem **getStorage** メソッドによって取得されたストレージを解放します。

```
void freeStorage(void* pStorage)
```

条件

INVREQ

getFile (1)

IccFile* getFile(const IccFileId& id)

id

CICS ファイルを識別する **IccFileId** オブジェクトへの参照。

getFile (2)

引数によって識別される **IccFile** オブジェクトを指すポインターを返します。

IccFile* getFile(const char* fileName)

fileName

CICS ファイルの名前。

条件

END、FILENOTFOUND、ILLOGIC、NOTAUTH

getNextFile

このメソッドは、**beginBrowse(IccSystem::file)** 呼び出しが正常に完了した後にのみ有効です。このメソッドは、CICS システム内のブラウズ順序における次のファイル・オブジェクトを返します。

IccFile* getNextFile()

条件

END、FILENOTFOUND、ILLOGIC、NOTAUTH

getStorage

要求されたサイズのストレージ・ブロックを取得し、それに対するポインターを返します。タスクの終了時にストレージは自動的に解放されません。ストレージが解放されるのは、**freeStorage** 操作が実行されたときのみです。

**void* getStorage (unsigned long size,
char initByte = -1,
unsigned long storageOpts = 0)**

size

要求されているストレージの量 (バイト数)

initByte

割り振られたストレージのすべてのバイトの初期設定

storageOpts

IccTask クラスで定義されている列挙型。これは、CICS がストレージを割り振る方法に影響します。

条件

LENGERR、NOSTG

instance

singleton **IccSystem** オブジェクトを指すポインターを返します。このオブジェクトは、まだ存在していない場合には作成されます。

static IccSystem* instance()

operatingSystem

char operatingSystem()

以下のように、CICS が稼働しているオペレーティング・システムを識別する 1 文字の値を返します。

A

AIX

N

Windows

X

z/OS

条件

NOTAUTH

operatingSystemLevel

CICS が稼働しているオペレーティング・システムのリリース番号を示すハーフワード・バイナリー・フィールドを返します。返される値は、正式なリリース番号を 10 倍にしたものです (バージョン番号は表されません)。例えば、MVS/ESA バージョン 3 リリース 2.1 は、値 21 を生成します。

unsigned short operatingSystemLevel()

条件

NOTAUTH

IccSystem public メソッド: release

CICS システムのレベルを返します。この値は、**EXE CICS INQUIRE SYSTEM** コマンドの **RELEASE** パラメーターに返される数値から得られます。

例えば、CICS Transaction Server for z/OS バージョン 4 リリース 2 に対して返されるリリース・レベルは 670 です。

unsigned long release()

条件

NOTAUTH

releaseText

release と同じものを返しますが、4 文字のストリングである点が異なります。例えば、CICS Transaction Server for z/OS [バージョン 1] リリース 3 は「0130」を返します。

const char* releaseText()

条件

NOTAUTH

sysId

この CICS システムを識別する **IccSysId** オブジェクトへの参照を返します。

IccSysId& sysId()

条件

INVREQ

workArea

CICS システムの作業域を保持している **IccBuf** オブジェクトへの参照を返します。

const IccBuf& workArea()

条件

INVREQ

継承された public メソッド

これらは、このクラスによって継承された public メソッドです。

メソッド	クラス
actionOnCondition	IccResource
actionOnConditionAsChar	IccResource
actionsOnConditionsText	IccResource
classType	IccBase
className	IccBase
condition	IccResource
conditionText	IccResource
customClassNum	IccBase
handleEvent	IccResource
id	IccResource
isEDFOn	IccResource
name	IccResource
operator delete	IccBase
operator new	IccBase

メソッド	クラス
setActionOnAnyCondition	IccResource
setActionOnCondition	IccResource
setActionsOnConditions	IccResource
setEDF	IccResource

継承された protected メソッド

これらは、このクラスによって継承された protected メソッドです。

メソッド	クラス
setClassName	IccBase
setCustomClassNum	IccBase

列挙

ResourceType

- autoInstallModel
- connection
- dataQueue
- exitProgram
- externalDataSet
- file
- journal
- modename
- partner
- profile
- program
- requestId
- systemDumpCode
- tempStore
- terminal
- transactionDumpCode
- transaction
- transactionClass

IccTask クラス

IccTask は、CICS サービスに関連するタスクを呼び出すために使用する singleton クラスです。

IccBase
IccResource
IccTask

ヘッダー・ファイル : ICCTSKEH

サンプル : ICC\$TSK

IccTask コンストラクター (protected)

コンストラクター

IccTask()

public メソッド

これらは、このクラス内の public メソッドです。

opt パラメーター

多くのメソッドに、*opt* という同じパラメーターがあります。このパラメーターについては、[62 ページの『abendCode』](#)の **abendCode** メソッドの下で説明しています。

異常終了

このタスクを異常終了するように CICS に要求します。

```
void abend (const char* abendCode = 0,  
            AbendHandlerOpt opt1 = respectAbendHandler,  
            AbendDumpOpt opt2 = createDump)
```

abendCode

4 文字の異常終了コード

opt1

このクラス内で定義されている、**IccControl** クラスの **setAbendHandler** メソッドによって指定された異常終了処理プログラムを実行するか無視するかを示す列挙型。

opt2

このクラス内で定義されている、ダンプを作成するかどうかを示す列挙型。

abendData

IccAbendData* abendData()

このタスクに関連するプログラム異常終了 (存在する場合) に関する情報が含まれている **IccAbendData** オブジェクトを指すポインターを返します。

commitUOW

void commitUOW()

このタスクの現行の UOW 内のリソース更新をコミットします。また、これは後続のリソース更新活動用に、新規 UOW を開始します。

条件

INVREQ、ROLLEDBACK

delay

このタスクを一定期間、あるいは特定の時刻まで遅延するように要求します。

```
void delay (const IccTime& time,  
            const IccRequestId* reqId = 0)
```

time

遅延時間に関する情報が入っているオブジェクトへの参照。そのオブジェクトは、以下のいずれかのタイプです。

IccAbsTime

1900 年の始まり以来のミリ秒数として時間を表します。

IccTimeInterval

時間間隔 (3 時間 2 分 1 秒など) を表します。

IccTimeOfDay

時刻 (13 時 30 分 (午後 1 時 30 分) など) を表します。

reqId

IccRequestId オブジェクトを指すオプションのポインター。これを使用して、まだ満了していない遅延要求をキャンセルできます。

条件

EXPIRED、INVREQ

dump

このタスクのメモリー・ダンプを取得するように CICS に要求します。(setDumpOpts も参照。) ダンプの文字 ID を返します。

```
const char* dump (const char* dumpCode,  
                  const IccBuf* buf = 0)
```

dumpCode

このダンプを識別する 4 文字のラベル

buf

ダンプに含める追加データが含まれている **IccBuf** オブジェクトを指すポインター。

条件

INVREQ、IOERR、NOSPACE、NOSTG、NOTOPEN、OPENERR、SUPPRESSED

enterTrace

CICS トレース・テーブルにユーザー・トレース項目を書き込みます。

```
void enterTrace (unsigned short traceNum,
                const char* resource = 0,
                IccBuf* data = 0,
                TraceOpt opt = normal)
```

traceNum

ユーザー・トレース・テーブル項目のトレース ID。0 から 199 の範囲の値。

resource

トレース・テーブル項目のリソース・フィールドに入力される 8 文字の名前。

data

トレース・レコードに含めるデータが含まれている **IccBuf** オブジェクトを指すポインター。

opt

このクラス内で定義されている、通常のトレースを行うか、例外のみをトレースするかを示す列挙型。

条件

INVREQ、LENGERR

facilityType

このクラス内で定義されている、このタスクが備えている基本機能のタイプを示す列挙型を返します。通常、これは端末です (誰かが CICS 端末上でトランザクション名をキー入力することでタスクが開始された場合など)。タスクがマップされた APPC 会話のバックエンドである場合は、セッションです。

FacilityType facilityType()

条件

INVREQ

freeStorage

IccTask **getStorage** メソッドによって取得されたストレージを解放します。

void freeStorage(void* pStorage)

条件

INVREQ

getStorage

要求されたサイズのストレージ・ブロックを取得します。ストレージは、タスクの終了時に自動的に、あるいは **freeStorage** 操作が実行されると解放されます。**IccSystem** クラスのページ [190 ページの『getStorage』](#) の **getStorage** も参照してください。

```
void* getStorage (unsigned long size,
                 char initByte = -1,
                 unsigned short storageOpts = 0)
```

size

要求されているストレージの量 (バイト数)

initByte

割り振られたストレージのすべてのバイトの初期設定

storageOpts

このクラスで定義されている列挙型。これは、CICS がストレージを割り振る方法に影響します。

条件

LENGERR、NOSTG

instance

singleton **IccTask** オブジェクトを指すポインターを返します。このオブジェクトは、まだ存在していない場合には作成されます。

static IccTask* instance();

isCommandSecurityOn

Icc::Bool isCommandSecurityOn()

Icc 構造で定義されている、このタスクがコマンド・セキュリティ検査の対象であるかどうかを示すブール値を返します。

条件

INVREQ

isCommitSupported

Icc 構造で定義されている、このタスクが **commit** メソッドをサポートできるかどうかを示すブール値を返します。このメソッドは、ほとんどの環境で true を返しますが、DPL 環境は例外です (**IccProgram** のページ 148 ページの『[link](#)』の **link** を参照)。

Icc::Bool isCommitSupported()

条件

INVREQ

isResourceSecurityOn

Icc 構造で定義されている、このタスクがリソース・セキュリティ検査の対象であるかどうかを示すブール値を返します。

Icc::Bool isResourceSecurityOn()

条件

INVREQ

isRestarted

Icc 構造で定義されている、このタスクが CICS によって自動的に再始動されたかどうかを示すブール値を返します。

Icc::Bool isRestarted()

条件

INVREQ

isStartDataAvailable

Icc 構造で定義されている、このタスクで開始データが使用可能であるかどうかを示すブール値を返します。開始データが使用可能な場合は、**IccStartRequestQ** クラスの **retrieveData** メソッドを参照してください。

Icc::Bool isStartDataAvailable()

条件

INVREQ

number

CICS システム内で固有の、このタスクの番号を返します。

unsigned long number() const

principalSysId

IccSysId& principalSysId(Icc::GetOpt opt = Icc::object)

このタスクの基本システム ID を識別する **IccSysId** オブジェクトへの参照を返します。

条件

INVREQ

priority

このタスクの優先順位を返します。

unsigned short priority(Icc::GetOpt opt = Icc::object)

条件

INVREQ

rollBackUOW

このタスク内の現行の UOW に関連付けられたリソース更新をロールバック (バックアウト) します。

void rollBackUOW()

条件

INVREQ、ROLLEDBACK

setDumpOpts

このタスクのダンプ・オプションを設定します。このメソッドは、このクラス内で定義されている **dump** メソッドの動作に影響します。

void setDumpOpts(unsigned long *opts* = dDefault)

opts

このクラス内で定義されている **DumpOpts** 列挙型からの値を加算または論理 OR することで作成される整数。

setPriority

このタスクのディスパッチング優先順位を変更します。

void setPriority(unsigned short *pri*)

pri

新しい優先順位。

条件

INVREQ

setWaitText

このタスクが **waitExternal** または **waitOnAlarm** メソッド呼び出しの結果として中断されている間に、誰かがこのタスクに対して照会を行った場合に表示されるテキストを設定します。

void setWaitText(const char* *name*)

name

このタスクが待機している理由を示す 8 文字のストリング・ラベル。

startType

StartType startType()

このクラスで定義されている、このタスクがどのように開始されたかを示す列挙型を返します。

条件

INVREQ

suspend

このタスクを中断し、他のタスクをディスパッチできるようにします。

void suspend()

transId

const IccTransId& transId()

この CICS タスクのトランザクション名を表す **IccTransId** オブジェクトを返します。

triggerDataQueueId

const IccDataQueueId& triggerDataQueueId()

このタスクが **IccDataQueue** に到着したデータの結果として開始された場合に、トリガー・キューを表す **IccDataQueueId** への参照を返します。 **startType** メソッドを参照してください。

条件

INVREQ

userId

このタスクに関連付けられているユーザーの ID を返します。

const IccUserId& userId(Icc::GetOpt opt = Icc::object)

opt

Icc 構造で定義されている、オブジェクト内に既に存在している情報を使用するか、CICS からその情報を更新するかを示す列挙型。

条件

INVREQ

waitExternal

イベント制御ブロック (ECB) を通知するイベントを待ちます。

この呼び出しを行うと、いずれかの ECB が通知されるまで、つまり、いずれかのイベントが発生するまで、タスクの発行が中断されます。タスクは、複数の ECB を待つことができ、それらのいずれかが通知されるとすぐにディスパッチすることができます。ECB について詳しくは、[WAIT EXTERNAL](#) を参照してください。

void waitExternal (long ECBList,
 unsigned long numEvents,
 WaitPurgeability opt = purgeable,
 WaitPostType type = MVSPost)**

ECBList

イベントを表す ECB のアドレス・リストへのポインター。

numEvents

ECBList 内のイベントの数。

opt

このクラス内で定義されている、待機がパージ可能であるかどうかを示す列挙型。

type

このクラス内で定義されている、ポスト・タイプが標準の MVS POST であるかどうかを示す列挙型。

条件

INVREQ

waitOnAlarm

アラームがオフ (期限切れ) になるまでタスクを中断します。

IccClock の [89 ページ](#) の『setAlarm』も参照してください。

void waitOnAlarm(const IccAlarmRequestId& id)

id

特定のアラーム要求を識別する **IccAlarmRequestId** オブジェクトへの参照。

条件

INVREQ

workArea

このタスクの作業域を保持している **IccBuf** オブジェクトへの参照を返します。

IccBuf& workArea()

条件

INVREQ

継承された public メソッド

これらは、このクラスによって継承された public メソッドです。

メソッド	クラス
actionOnCondition	IccResource
actionOnConditionAsChar	IccResource
actionsOnConditionsText	IccResource
classType	IccBase
className	IccBase
condition	IccResource
conditionText	IccResource
customClassNum	IccBase
handleEvent	IccResource
id	IccResource
isEDFOn	IccResource

メソッド	クラス
name	IccResource
operator delete	IccBase
operator new	IccBase
setActionOnAnyCondition	IccResource
setActionOnCondition	IccResource
setActionsOnConditions	IccResource
setEDF	IccResource

継承された protected メソッド

これらは、このクラスによって継承された protected メソッドです。

メソッド	クラス
setClassName	IccBase
setCustomClassNum	IccBase

列挙

AbendHandlerOpt

respectAbendHandler

異常終了処理プログラムが有効である場合に、それに制御を渡すことを許可します。

ignoreAbendHandler

異常終了処理プログラムが有効であっても、それに制御を渡すことを許可しません。

AbendDumpOpt

createDump

異常終了要求に対してサービスを行う場合に、トランザクション・ダンプを作成します。

suppressDump

異常終了要求に対してサービスを行う場合に、トランザクション・ダンプを取得しません。

DumpOpts

値を加算またはビット単位 OR によってまとめて、必要な組み合わせを取得することができます。

値を加算またはビット単位 OR によってまとめて、必要な組み合わせを取得することができます。例えば、IccTask::dProgram + IccTask::dDCT + IccTask::dSIT のようにします。

dDefault

dComplete

dTask

dStorage

dProgram

dTerminal

dTables

dDCT

dFCT

dPCT

dPPT

dSIT

dTCT

dTRT

FacilityType

none

タスクには基本機能がありません。つまり、これはバックグラウンド・タスクです。

terminal

このタスクは端末を基本装置としています。

session

このタスクは、セッションを基本機能としています。つまり、これはバックエンド DTP プログラムとして開始されたと考えられます。

dataqueue

このタスクは一時データ・キューを基本装置としています。

StartType

DPL

分散プログラム・リンク要求

dataQueueTrigger

データ・キューへのデータの到着によるトリガー

startRequest

非同期開始要求の結果として開始されたもの。**IccStartRequestQ** クラスを参照。

FEPIRequest

フロントエンド・プログラミング・インターフェース。

terminalInput

端末入力によって開始されたもの

CICSInternalTask

CICS によって開始されたもの。

StorageOpts

ifSOSReturnCondition

使用可能なスペースが不足している場合は、タスクをブロックする代わりに NOSTG 条件を返します。

below

16 MB 境界より下のストレージを割り振ります。

userDataKey

USER データ・キー内のストレージを割り振ります。

CICSDataKey

CICS データ・キー内のストレージを割り振ります。

TraceOpt

normal

トレース項目は標準エントリーです。

例外 (exception)

トレース項目は例外エントリーです。

WaitPostType

MVSPost

ECB は、MVS POST サービスを使用して通知されます。

handPost

ECB は、手動で通知されます (つまり、MVS POST サービス以外のメソッドを使用)。

WaitPurgeability

purgeable

タスクは、システム呼び出しを介してページすることができます。

notPurgeable

タスクは、システム呼び出しを介してページできません。

IccTempStore クラス

IccTempStore オブジェクトは、データの一時記憶域を管理するために使用されます。

IccBase

IccResource

IccTempStore

IccTempStore データは、複数のトランザクション呼び出しにまたがって存在できます。

ヘッダー・ファイル : ICCTMPEH

サンプル : ICC\$TMP

IccTempStore コンストラクター

コンストラクター (1)

**IccTempStore (const IccTempStoreId& *id*,
Location *loc* = auxStorage)**

id

IccTempStoreId オブジェクトへの参照

loc

このクラス内で定義されている、ストレージを最初に作成したときに配置する場所を示す列挙型。デフォルトでは、補助記憶域 (ディスク) を使用します。

コンストラクター (2)

**IccTempStore (const char* *storeName*,
Location *loc* = auxStorage)**

storeName

使用するキューの 8 文字の名前を指定します。この名前は、CICS システム内で固有でなければなりません。

loc

このクラス内で定義されている、ストレージを最初に作成したときに配置する場所を示す列挙型。デフォルトでは、補助記憶域 (ディスク) を使用します。

public メソッド

これらは、このクラス内の public メソッドです。

opt パラメーター

多くのメソッドに、*opt* という同じパラメーターがあります。このパラメーターについては、[62 ページの『abendCode』](#)の **abendCode** メソッドの下で説明しています。

clear

empty と同義。ポリモアフィズムについては、[ポリモアフィック動作](#) を参照してください。

virtual void clear()**empty****void empty()**

IccTempStore オブジェクトに関連付けられたすべての一時データを削除し、関連する TD キューを削除します。

条件

INVREQ、ISCINVREQ、NOTAUTH、QIDERR、SYSIDERR

get

readNextItem と同義。ポリモアフィズムについては、[ポリモアフィック動作](#) を参照してください。

virtual const IccBuf& get()**numberOfItems****unsigned short numberOfItems() const**

一時記憶域内の項目数を返します。これは、**writeItem** 呼び出しが正常に完了した後にのみ有効です。

put

writeItem と同義。ポリモアフィズムについては、[ポリモアフィック動作](#) を参照してください。

virtual void put(const IccBuf& buffer)

buffer

一時記憶域キューの末尾に追加するデータが入っている **IccBuf** オブジェクトへの参照。

readItem

一時記憶域キューから指定された項目を読み取り、情報が含まれる **IccBuf** オブジェクトへの参照を返します。

const IccBuf& readItem(unsigned short itemNum)

itemNum

キューから取得する論理レコードの項目番号を指定します。

条件

INVREQ、IOERR、ISCINVREQ、ITEMERR、LENGERR、NOTAUTH、QIDERR、SYSIDERR

readNextItem

一時記憶域キューから次の項目を読み取り、情報が含まれる **IccBuf** オブジェクトへの参照を返します。

const IccBuf& readNextItem()

条件

INVREQ、IOERR、ISCINVREQ、ITEMERR、LENGERR、NOTAUTH、QIDERR、SYSIDERR

rewriteItem

パラメーターは以下のとおりです。このメソッドは、一時記憶域キュー内の指定された項目を更新します。

**void rewriteItem (unsigned short itemNum,
 const IccBuf& item,
 NoSpaceOpt opt = suspend)**

itemNum

変更する論理レコードの項目番号を指定します。

item

更新データが入っている **IccBuf** オブジェクトの名前。

opt

このクラス内で定義されている、キュー内のスペース不足によってレコードを追加できない場合にアプリケーション・プログラムを中断するかどうかを示す列挙型。suspend がデフォルトです。

条件

INVREQ、IOERR、ISCINVREQ、ITEMERR、LENGERR、NOSPACE、NOTAUTH、QIDERR、SYSIDERR

writeItem (1)

**unsigned short writeItem (const IccBuf& item,
NoSpaceOpt opt = suspend)**

item

一時記憶域キューの末尾に追加するデータが含まれる **IccBuf** オブジェクトの名前。

opt

このクラス内で定義されている、キュー内のスペース不足によってレコードを追加できない場合にアプリケーション・プログラムを中断するかどうかを示す列挙型。suspend がデフォルトです。

writeItem (2)

このメソッドは、新規レコードを一時記憶域キューの末尾に追加します。戻り値は、作成された項目番号です (正常に完了した場合)。

**unsigned short writeItem (const char* text,
NoSpaceOpt opt = suspend)**

text

一時記憶域キューの末尾に追加されるテキスト・ストリング。

opt

このクラス内で定義されている、キュー内のスペース不足によってレコードを追加できない場合にアプリケーション・プログラムを中断するかどうかを示す列挙型。suspend がデフォルトです。

条件

INVREQ、IOERR、ISCINVREQ、ITEMERR、LENGERR、NOSPACE、NOTAUTH、QIDERR、SYSIDERR

継承された public メソッド

これらは、このクラスによって継承された public メソッドです。

メソッド	クラス
actionOnCondition	IccResource
actionOnConditionAsChar	IccResource
actionsOnConditionsText	IccResource
className	IccBase
classType	IccBase
condition	IccResource
conditionText	IccResource
customClassNum	IccBase
handleEvent	IccResource
id	IccResource
isEDFOn	IccResource
isRouteOptionOn	IccResource
name	IccResource

メソッド	クラス
operator delete	IccBase
operator new	IccBase
routeOption	IccResource
setActionOnAnyCondition	IccResource
setActionOnCondition	IccResource
setActionsOnConditions	IccResource
setEDF	IccResource
setRouteOption	IccResource

継承された protected メソッド

これらは、このクラスによって継承された protected メソッドです。

メソッド	クラス
setClassName	IccBase
setCustomClassNum	IccBase

列挙

場所

auxStorage

一時保管データは、補助記憶域 (ディスク) 上にあります。

ストレージ

一時保管データは、メモリー内にあります。

NoSpaceOpt

キューのスペース不足によってレコードを即時に追加できない場合は、このアクションを実行します。

suspend

アプリケーション・プログラムを一時的に中断します。

returnCondition

アプリケーション・プログラムは中断しませんが、代わりに NOSPAC 条件が発生します。

IccTempStoreId クラス

IccTempStoreId クラスは、CICS システム内の一時記憶域名を識別するために使用されます。

IccBase

IccResourceId

IccTempStoreId

ヘッダー・ファイル: ICCRIDEH

IccTempStoreId コンストラクター

コンストラクター (1)

IccTempStoreId(const char* name)

name

一時記憶域エントリーの 8 文字の名前。

コンストラクター (2)

コピー・コンストラクター。

IccTempStoreId(const IccTempStoreId& *id*)

id

IccTempStoreId オブジェクトへの参照。

public メソッド

これらは、このクラス内の public メソッドです。

operator= (1)

IccTempStoreId& operator=(const char* *name*)

name

一時記憶域エントリーの 8 文字の名前。

operator= (2)

新規値を代入します。

IccTempStoreId& operator=(const IccTempStoreId& *id*)

id

IccTempStoreId オブジェクトへの参照。

継承された public メソッド

これらは、このクラスによって継承された public メソッドです。

メソッド	クラス
classType	IccBase
className	IccBase
customClassNum	IccBase
name	IccResourceId
nameLength	IccResourceId
operator delete	IccBase

メソッド
operator new

クラス
IccBase

継承された protected メソッド

これらは、このクラスによって継承された protected メソッドです。

メソッド
operator=
setClassName
setCustomClassNum

クラス
IccResourceId
IccBase
IccBase

IccTermId クラス

IccTermId クラスは、CICS システム内の端末名を識別するために使用されます。

IccBase
IccResourceId
IccTermId

ヘッダー・ファイル: ICCRIDEH

IccTermId コンストラクター

コンストラクター (1)

IccTermId(const char* *name*)

name
4 文字の端末名

コンストラクター (2)
コピー・コンストラクター。

IccTermId(const IccTermId& *id*)

id
IccTermId オブジェクトへの参照。

public メソッド

これらは、このクラス内の public メソッドです。

operator= (1)

IccTermId& operator=(const char* name)

name

4 文字の端末名

operator= (2)

新規値を代入します。

IccTermId& operator=(const IccTermId& id)

id

IccTermId オブジェクトへの参照。

継承された public メソッド

これらは、このクラスによって継承された public メソッドです。

メソッド	クラス
classType	IccBase
className	IccBase
customClassNum	IccBase
name	IccResourceId
nameLength	IccResourceId
operator delete	IccBase
operator new	IccBase

継承された protected メソッド

これらは、このクラスによって継承された protected メソッドです。

メソッド	クラス
operator=	IccResourceId
setClassName	IccBase
setCustomClassNum	IccBase

IccTerminal クラス

これは、CICS タスクに属する端末を表す singleton クラスです。このクラスは、トランザクションの基本装置が 3270 端末である場合にのみ作成され、それ以外の場合は例外がスローされます。

IccBase

IccResource

IccTerminal

ヘッダー・ファイル: ICCTRMEH

サンプル: ICC\$TRM

IccTerminal コンストラクター (protected)

コンストラクター

IccTerminal()

public メソッド

これらは、このクラス内の public メソッドです。

opt パラメーター

多くのメソッドに、*opt* という同じパラメーターがあります。このパラメーターについては、[62 ページの『abendCode』](#)の **abendCode** メソッドの下で説明しています。

AID

このクラス内で定義されている、この端末で最後に押された AID (アクション ID) キーを示す列挙型を返します。

AIDVal AID()

clear

virtual void clear()

erase と同義です。ポリモアフィズムについては、[ポリモアフィック動作](#) を参照してください。

cursor

unsigned short cursor()

画面の左上隅からのオフセットとして現行カーソル位置を返します。

データ

IccTerminalData* data()

端末の特性に関する情報が含まれる **IccTerminalData** オブジェクトへのポインターを返します。このオブジェクトは、まだ存在していなければ作成されます。

erase

void erase()

端末に表示されているすべてのデータを消去します。

条件

INVREQ、INVPARTN

freeKeyboard

端末が入力を受け入れることができるように、キーボードを解放します。

void freeKeyboard()

条件

INVREQ、INVPARTN

get

receive の同義語です。ポリモフィズムについては、[ポリモフィック動作](#) を参照してください。

virtual const IccBuf& get()

height

unsigned short height(Icc::getopt *opt* = Icc::object)

画面に保持される行数を返します。

条件

INVREQ

inputCursor

画面上のカーソルの位置を返します。

unsigned short inputCursor()

instance

static IccTerminal* instance()

単一の **IccTerminal** オブジェクトを指すポインターを返します。このオブジェクトは、まだ存在していなければ作成されます。

line

unsigned short line()

画面の先頭から数えた、カーソルの現在の行番号を返します。

netName

const char* netName()

基本機能のネットワーク論理装置名を表す 8 バイトのストリングを返します。

operator« (1)

端末に送信される後続のデータの前景色を設定します。

IccTerminal& operator « (Color *color*)

operator« (2)

端末に送信される後続のデータに使用する強調表示を設定します。

IccTerminal& operator « (Highlight *highlight*)

operator« (3)

別のバッファーに書き込みます。

IccTerminal& operator « (const IccBuf& *buffer*)

operator« (4)

文字を書き込みます。

IccTerminal& operator « (char *ch*)

operator« (5)

文字を書き込みます。

IccTerminal& operator « (signed char *ch*)

operator« (6)

文字を書き込みます。

IccTerminal& operator « (unsigned char *ch*)

operator« (7)

ストリングを書き込みます。

IccTerminal& operator « (const char* *text*)

operator« (8)

ストリングを書き込みます。

IccTerminal& operator « (const signed char* *text*)

operator« (9)

ストリングを書き込みます。

IccTerminal& operator « (const unsigned char* *text*)

operator« (10)

short 型を書き込みます。

IccTerminal& operator « (short *num*)

operator« (11)

符号なし short 型を書き込みます。

IccTerminal& operator « (unsigned short *num*)

operator« (12)

long 型を書き込みます。

IccTerminal& operator « (long *num*)

operator« (13)

符号なし long 型を書き込みます。

IccTerminal& operator « (unsigned long *num*)

operator« (14)

整数を書き込みます。

IccTerminal& operator « (int *num*)

operator« (15)

float 型を書き込みます。

IccTerminal& operator « (float *num*)

operator« (16)

double 型を書き込みます。

IccTerminal& operator « (double *num*)

operator« (17)

long double 型を書き込みます。

IccTerminal& operator « (long double *num*)

operator« (18)

IccTerminal& operator « (IccTerminal& (*f)(IccTerminal&))

以下の構文が使用可能になります。

```
Term « "Hello World" « endl;  
Term « "Hello again" « flush;
```

put

virtual void put(const IccBuf& *buf*)

sendLine と同義。ポリモアフィズムについては、[ポリモアフィック動作](#) を参照してください。

受信

端末からデータを受信します。

```
const IccBuf& receive(Case caseOpt = upper)
```

caseOpt

このクラス内で定義されている、テキストを大文字に変換するかどうかを示す列挙型。

条件

EOC、INVREQ、LENGERR、NOTALLOC、SIGNAL、TERMERR

receive3270Data

端末から 3270 データ・バッファーを受信します。

```
const IccBuf& receive3270Data(Case caseOpt = upper)
```

caseOpt

このクラス内で定義されている、テキストを大文字に変換するかどうかを示す列挙型。

条件

INVREQ、LENGERR、TERMERR

send (1)

```
void send(const IccBuf& buffer)
```

buffer

送信するデータを保持している **IccBuf** オブジェクトへの参照。

send (2)

```
void send (const char* format,  
...)
```

format

printf 標準ライブラリー関数のものと同様な書式制御ストリング。

...
format に付随するオプションの引数。

send (3)

**void send (unsigned short row,
 unsigned short col,
 const IccBuf& buffer)**

row
データの書き込みが開始される行。

col
データの書き込みが開始される列。

buffer
送信するデータを保持している **IccBuf** オブジェクトへの参照。

send (4)
指定されたデータを現行カーソル位置、または引数によって指定されたカーソル位置に書き込みます。

**void send (unsigned short row,
 unsigned short col,
 const char* format,
 ...)**

row
データの書き込みが開始される行。

col
データの書き込みが開始される列。

format
printf 標準ライブラリー関数のものと同様な書式制御ストリング。

...
format に付随するオプションの引数。

条件

INVREQ、LENGERR、TERMERR

send3270Data (1)

void send3270Data(const IccBuf& buffer)

buffer
送信するデータを保持している **IccBuf** オブジェクトへの参照。

send3270Data (2)

```
void send3270Data(const char* format,  
...)
```

format

printf 標準ライブラリー関数のものと同様な書式制御ストリング

...

format に付随するオプションの引数。

send3270Data (3)

```
void send3270Data (unsigned short col,  
const IccBuf& buf)
```

col

データの書き込みが開始される列

buffer

送信するデータを保持している **IccBuf** オブジェクトへの参照。

send3270Data (4)

指定されたデータを端末の次の行あるいは現在行の指定された列に書き込みます。

```
void send3270Data (unsigned short col,  
const char* format,  
...)
```

col

データの書き込みが開始される列

format

printf 標準ライブラリー関数のものと同様な書式制御ストリング

...

format に付随するオプションの引数。

条件

INVREQ、LENGERR、TERMERR

sendLine (1)

void sendLine(const IccBuf&*buffer*)

buffer

送信するデータを保持している **IccBuf** オブジェクトへの参照。

sendLine (2)

**void sendLine (const char* *format*,
...)**

format

printf 標準ライブラリー関数のものと同様な書式制御ストリング

...

format に付随するオプションの引数。

sendLine (3)

**void sendLine (unsigned short *col*,
const IccBuf& *buf*)**

col

データの書き込みが開始される列

buffer

送信するデータを保持している **IccBuf** オブジェクトへの参照。

sendLine (4)

指定されたデータを端末の次の行あるいは現在行の指定された列に書き込みます。

**void sendLine (unsigned short *col*,
const char* *format*,
...)**

col

データの書き込みが開始される列

format

printf 標準ライブラリー関数のものと同様な書式制御ストリング

...

format に付随するオプションの引数。

条件

INVREQ、LENGERR、TERMERR

setColor

端末に送信される後続のテキストの色を変更します。

void setColor(Color *color*=defaultColor)

color

このクラス内で定義されている、画面に書き込まれるテキストの色を示す列挙型。

setCursor (1)

void setCursor(unsigned short *offset*)

offset

左上隅を 0 とした場合のカーソルの位置。

setCursor (2)

画面上のカーソルの位置を設定する 2 つの方法。

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

row

最上部の行を 1 とした場合のカーソルの行番号

col

左端の列を 1 とした場合のカーソルの列番号

条件

INVREQ、INVPARTN

setHighlight

端末に送信される後続のデータの強調表示を変更します。

void setHighlight(Highlight *highlight* = normal)

highlight

このクラス内で定義されている、画面に書き込まれるテキストの強調表示を示す列挙型。

setLine

行 *lineNum* の先頭にカーソルを移動します。ここで、1 は端末の最初の行です。デフォルトでは、カーソルは行 1 の始めに移動します。

void setLine(unsigned short *lineNum* = 1)

lineNum

先頭からカウントした行番号。

条件

INVREQ、INVPARTN

setNewLine

numLines 行のブランク行を端末に送信するように要求します。

void setNewLine(unsigned short *numLines* = 1)

numLines

ブランク行の数。

条件

INVREQ、INVPARTN

setNextCommArea

この端末で開始される次のトランザクションに渡される COMMAREA を指定します。

void setNextCommArea(const IccBuf& *commArea*)

commArea

COMMAREA として使用するバッファへの参照。

setNextInputMessage

receive メソッドによって、この端末で開始される次のトランザクションで使えるようにするデータを指定します。

void setNextInputMessage(const IccBuf& *message*)

message

入力メッセージを保持するバッファへの参照。

setNextTransId

この端末で開始される次のトランザクションを指定します。

```
void setNextTransId (const IccTransId& transid,  
                    NextTransIdOpt opt = queue)
```

transid

トランザクションの名前を保持する **IccTransId** オブジェクトへの参照

opt

このクラス内で定義されている、*transId* をキューに入れるか、この端末で即時に (つまり、次のトランザクションとして) 開始するかを示す列挙型。

signoff

void signoff()

現在サインオンしているユーザーをサインオフします。権限がデフォルト・ユーザーに戻ります。

条件

INVREQ

signon (1)

```
void signon (const IccUserId& id,  
            const char* password = 0,  
            const char* newPassword = 0)
```

id

IccUserId オブジェクトへの参照

password

8 文字の既存のパスワード。

newPassword

オプションの 8 文字の新規パスワード。

signon (2)

ユーザーを端末にサインオンさせます。

```
void signon (IccUser& user,  
            const char* password = 0,  
            const char* newPassword = 0)
```

user

IccUser オブジェクトへの参照

password

8 文字の既存のパスワード。

newPassword

オプションの 8 文字の新規パスワード。このメソッドは、**IccGroupId** および言語の情報を検出するために **IccUser** オブジェクトに問い合わせを行う最初の **signon** メソッドとは異なります。オブジェクトは、言語および ESM の戻りコードおよび応答コードによっても更新されます。

条件

INVREQ、NOTAUTH、USERIDERR

waitForAID (1)

入力があるまで待機し、このクラス内で定義されている、予期される AID を示す列挙型を返します。

AIDVal waitForAID()

waitForAID (2)

指定された AID キーが押されるまで待ってから、制御を返します。このメソッドは、オペレーターが正しい AID キーを押すまで、端末からの入力を受信するループを実行します。

void waitForAID(AIDVal aid)

aid

このクラス内で定義されている、最後に押された AID キーを示す列挙型。

条件

EOC、INVREQ、LENGERR、NOTALLOC、SIGNAL、TERMERR

width

画面の幅を文字数で返します。

unsigned short width(Icc::getopt opt = Icc::object)

条件

INVREQ

workArea

端末作業域を保持している **IccBuf** オブジェクトへの参照を返します。

IccBuf& workArea()

継承された public メソッド

これらは、このクラスによって継承された public メソッドです。

メソッド	クラス
actionOnCondition	IccResource
actionOnConditionAsChar	IccResource
actionsOnConditionsText	IccResource
classType	IccBase
className	IccBase
condition	IccResource
conditionText	IccResource
customClassNum	IccBase
handleEvent	IccResource
id	IccResource
isEDFOn	IccResource
name	IccResource
operator delete	IccBase
operator new	IccBase
setActionOnAnyCondition	IccResource
setActionOnCondition	IccResource
setActionsOnConditions	IccResource
setEDF	IccResource

継承された protected メソッド

これらは、このクラスによって継承された protected メソッドです。

メソッド	クラス
setClassName	IccBase
setCustomClassNum	IccBase

列挙

AIDVal

ENTER

CLEAR

PA1 から PA3

PF1 から PF24

Case

upper

mixed

Color

defaultColor

blue

red

pink

green

cyan

yellow

neutral

Highlight

defaultHighlight

blink

reverse

underscore

NextTransIdOpt

queue

トランザクションをキューに入れます。その際、端末上でキューに入っているその他の未解決の開始があれば、それらも一緒にキューに入れます。

immediate

トランザクションを即時に (つまり、端末上でキューに入っているその他の未解決の開始の前に) 開始します。

IccTerminalData クラス

IccTerminalData は、**IccTerminal** が所有する singleton クラスです。これには、端末特性に関する情報が含まれています。

IccTerminal クラスの [212 ページの『データ』](#) を参照してください。

IccBase

IccResource

IccTerminalData

ヘッダー・ファイル : ICCTMDEH

サンプル : ICC\$TRM

IccTerminalData コンストラクター (protected)

コンストラクター

IccTerminalData()

public メソッド

これらは、このクラス内の public メソッドです。

opt パラメーター

多くのメソッドに、*opt* という同じパラメーターがあります。このパラメーターについては、[62 ページの『abendCode』](#)の **abendCode** メソッドの下で説明しています。

alternateHeight

画面の代替の高さを行数で返します。

unsigned short alternateHeight(Icc::GetOpt *opt* = Icc::object)

opt

オブジェクト内の情報を抽出前に CICS から更新するかどうかを示す列挙型。デフォルトでは更新しません。

条件

INVREQ

alternateWidth

画面の代替の幅を文字数で返します。

unsigned short alternateWidth(Icc::GetOpt *opt* = Icc::object)

条件

INVREQ

defaultHeight

画面のデフォルトの高さを行数で返します。

unsigned short defaultHeight(Icc::GetOpt *opt* = Icc::object)

条件

INVREQ

defaultWidth

画面のデフォルトの幅を文字数で返します。

unsigned short defaultWidth(Icc::GetOpt *opt* = Icc::object)

条件

INVREQ

graphicCharCodeSet

バイナリー・コード・ページのグローバル ID を 1 から 65534 の範囲の値として、あるいは非グラフィック端末の場合は 0 を返します。

unsigned short graphicCharCodeSet(Icc::GetOpt *opt* = Icc::object)

条件

INVREQ

graphicCharSetId

図形文字セットのグローバル ID を 1 から 65534 の範囲の数値として、あるいは非グラフィック端末の場合は 0 を返します。

unsigned short graphicCharSetId(Icc::GetOpt *opt* = Icc::object)

条件

INVREQ

isAPLKeyboard

端末に APL キーボード機能があるかどうかを示すブール値を返します。

Icc::Bool isAPLKeyboard(Icc::GetOpt *opt* = Icc::object)

条件

INVREQ

isAPLText

端末に APL テキスト機能があるかどうかを示すブール値を返します。

Icc::Bool isAPLText(Icc::GetOpt *opt* = Icc::object)

条件

INVREQ

isBTrans

端末に背景透過性機能があるかどうかを示すブール値を返します。

Icc::Bool isBTrans(Icc::GetOpt *opt* = Icc::object)

条件

INVREQ

isColor

端末に拡張カラー機能があるかどうかを示すブール値を返します。

Icc::Bool isColor(Icc::GetOpt *opt* = Icc::object)

条件

INVREQ

isEWA

端末が消去書き込み代替をサポートするかどうかを示すブール値を返します。

Icc::Bool isEWA(Icc::GetOpt *opt* = Icc::object)

条件

INVREQ

isExtended3270

端末が 3270 拡張データ・ストリームをサポートするかどうかを示すブール値を返します。

Icc::Bool isExtended3270(Icc::GetOpt *opt* = Icc::object)

条件

INVREQ

isFieldOutline

端末がフィールド外形線をサポートするかどうかを示すブール値を返します。

Icc::Bool isFieldOutline(Icc::GetOpt *opt* = Icc::object)

条件

INVREQ

isGoodMorning

端末に「good morning」メッセージがあるかどうかを示すブール値を返します。

Icc::Bool isGoodMorning(Icc::GetOpt *opt* = Icc::object)

条件

INVREQ

isHighlight

端末に拡張強調表示機能があるかどうかを示すブール値を返します。

Icc::Bool isHighlight(Icc::GetOpt *opt* = Icc::object)

条件

INVREQ

isKatakana

端末が英数カナ文字モードをサポートするかどうかを示すブール値を返します。

Icc::Bool isKatakana(Icc::GetOpt *opt* = Icc::object)

条件

INVREQ

isMSRControl

端末が磁気スロット読取装置制御をサポートするかどうかを示すブール値を返します。

Icc::Bool isMSRControl(Icc::GetOpt *opt* = Icc::object)

条件

INVREQ

isPS

端末がプログラム式シンボルをサポートするかどうかを示すブール値を返します。

Icc::Bool isPS(Icc::GetOpt *opt* = Icc::object)

条件

INVREQ

isSOSI

端末が EBCDIC/DBCS 混合フィールドをサポートするかどうかを示すブール値を返します。

Icc::Bool isSOSI(Icc::GetOpt *opt* = Icc::object)

条件

INVREQ

isTextKeyboard

端末が TEXTKYBD をサポートするかどうかを示すブール値を返します。

Icc::Bool isTextKeyboard(Icc::GetOpt *opt* = Icc::object)

条件

INVREQ

isTextPrint

端末が TEXTPRINT をサポートするかどうかを示すブール値を返します。

Icc::Bool isTextPrint(Icc::GetOpt *opt* = Icc::object)

条件

INVREQ

isValidation

端末が妥当性検査をサポートするかどうかを示すブール値を返します。

Icc::Bool isValidation(Icc::GetOpt *opt* = Icc::object)

条件

INVREQ

継承された public メソッド

これらは、このクラスによって継承された public メソッドです。

メソッド	クラス
actionOnCondition	IccResource
actionOnConditionAsChar	IccResource
actionsOnConditionsText	IccResource
classType	IccBase
className	IccBase
condition	IccResource
conditionText	IccResource
customClassNum	IccBase
handleEvent	IccResource
id	IccResource
isEDFOn	IccResource
name	IccResource
operator delete	IccBase
operator new	IccBase
setActionOnAnyCondition	IccResource
setActionOnCondition	IccResource
setActionsOnConditions	IccResource
setEDF	IccResource

継承された protected メソッド

これらは、このクラスによって継承された protected メソッドです。

メソッド	クラス
setClassName	IccBase
setCustomClassNum	IccBase

IccTime クラス

IccTime は、時間情報を含めるために使用され、**IccAbsTime**、**IccTimeInterval**、および **IccTimeOfDay** の各クラスの派生元となる基本クラスです。

IccBase
IccResource
IccTime

ヘッダー・ファイル: ICCTIMEH

IccTime コンストラクター (protected)

コンストラクター

IccTime (unsigned long *hours* = 0,
unsigned long *minutes* = 0,
unsigned long *seconds* = 0)

hours
時間数

minutes
分数

seconds
秒数

public メソッド

これらは、このクラス内の public メソッドです。

hours
コンストラクターで指定された値である時刻の時間コンポーネントを返します。

virtual unsigned long hours() const

minutes

virtual unsigned long minutes() const

コンストラクターで指定された時間値の「分」のコンポーネントを返します。

seconds

virtual unsigned long seconds() const

コンストラクターで指定された時間値の「秒」のコンポーネントを返します。

timeInHours

virtual unsigned long timeInHours()

時間を時間数で返します。

timeInMinutes

virtual unsigned long timeInMinutes()

時間を分数で返します。

timeInSeconds

virtual unsigned long timeInSeconds()

時間を秒数で返します。

type

Type type() const

このクラス内で定義されている、**IccTime** のサブクラスのタイプを示す列挙型を返します。

継承された public メソッド

これらは、このクラスによって継承された public メソッドです。

メソッド	クラス
actionOnCondition	IccResource
actionOnConditionAsChar	IccResource
actionsOnConditionsText	IccResource
className	IccBase
classType	IccBase
condition	IccResource
conditionText	IccResource
customClassNum	IccBase
handleEvent	IccResource
isEDFOn	IccResource
operator delete	IccBase
operator new	IccBase
setActionOnAnyCondition	IccResource
setActionOnCondition	IccResource
setActionsOnConditions	IccResource
setEDF	IccResource

継承された protected メソッド

これらは、このクラスによって継承された protected メソッドです。

メソッド	クラス
setClassName	IccBase
setCustomClassNum	IccBase

列挙

タイプ

absTime

オブジェクトは **IccAbsTime** クラスのオブジェクトです。これは、現在の日時を 1900 年の始め以来経過したミリ秒数で表すために使用します。

timeInterval

オブジェクトは **IccTimeInterval** クラスのオブジェクトです。これは、5 分など、時間の長さを表すために使用します。

timeOfDay

IccTimeOfDay クラスのオブジェクト。これは、特定の時刻、例えば深夜 0 時などを表すために使用します。

IccTimeInterval クラス

このクラスは、時間間隔に関する情報を保持します。

IccBase
IccResource
IccTime
IccTimeInterval

ヘッダー・ファイル: ICCTIMEH

IccTimeInterval コンストラクター

コンストラクター (1)

**IccTimeInterval (unsigned long *hours* = 0,
unsigned long *minutes* = 0,
unsigned long *seconds* = 0)**

hours

時分秒の「時」の初期設定値。デフォルトは 0 です。

minutes

時分秒の「分」の初期設定値。デフォルトは 0 です。

seconds

時分秒の「秒」の初期設定値。デフォルトは 0 です。

コンストラクター (2)

コピー・コンストラクター。

IccTimeInterval(const IccTimeInterval& *time*)

public メソッド

これらは、このクラス内の public メソッドです。

operator=

1 つの **IccTimeInterval** オブジェクトを別のオブジェクトに代入します。

IccTimeInterval& operator=(const IccTimeInterval& *timeInterval*)

set

IccTimeInterval オブジェクトに保持されている時刻を変更します。

**void set (unsigned long *hours*,
 unsigned long *minutes*,
 unsigned long *seconds*)**

hours

新規の「時間」設定

minutes

新規の「分」設定

seconds

新規の「秒」設定

継承された public メソッド

これらは、このクラスによって継承された public メソッドです。

メソッド	クラス
actionOnCondition	IccResource
actionOnConditionAsChar	IccResource
actionsOnConditionsText	IccResource
classType	IccBase
className	IccBase
condition	IccResource
conditionText	IccResource
customClassNum	IccBase
handleEvent	IccResource
hours	IccTime
isEDFOn	IccResource
minutes	IccTime

メソッド	クラス
operator delete	IccBase
operator new	IccBase
setActionOnAnyCondition	IccResource
setActionOnCondition	IccResource
setActionsOnConditions	IccResource
setEDF	IccResource
timeInHours	IccTime
timeInMinutes	IccTime
timeInSeconds	IccTime
type	IccTime

継承された protected メソッド

これらは、このクラスによって継承された protected メソッドです。

メソッド	クラス
setClassName	IccBase
setCustomClassNum	IccBase

IccTimeOfDay クラス

このクラスは、時刻に関する情報を保持します。

```

IccBase
  IccResource
    IccTime
      IccTimeOfDay

```

ヘッダー・ファイル: ICCTIMEH

IccTimeOfDay コンストラクター

コンストラクター (1)

```

IccTimeOfDay (unsigned long hours = 0,
              unsigned long minutes = 0,
              unsigned long seconds = 0)

```

hours

時分秒の「時」の初期設定値。デフォルトは 0 です。

minutes

時分秒の「分」の初期設定値。デフォルトは 0 です。

seconds

時分秒の「秒」の初期設定値。デフォルトは 0 です。

コンストラクター (2)

コピー・コンストラクター

IccTimeOfDay(const IccTimeOfDay& *time*)

public メソッド

これらは、このクラス内の public メソッドです。

operator=

1 つの **IccTimeOfDay** オブジェクトを別のオブジェクトに代入します。

IccTimeOfDay& operator=(const IccTimeOfDay& *timeOfDay*)

set

IccTimeOfDay オブジェクトに保持されている時刻を変更します。

**void set (unsigned long *hours*,
 unsigned long *minutes*,
 unsigned long *seconds*)**

hours

新規の「時間」設定

minutes

新規の「分」設定

seconds

新規の「秒」設定

継承された public メソッド

これらは、このクラスによって継承された public メソッドです。

メソッド	クラス
<code>actionOnCondition</code>	<code>IccResource</code>
<code>actionOnConditionAsChar</code>	<code>IccResource</code>
<code>actionsOnConditionsText</code>	<code>IccResource</code>
<code>classType</code>	<code>IccBase</code>
<code>className</code>	<code>IccBase</code>
<code>condition</code>	<code>IccResource</code>
<code>conditionText</code>	<code>IccResource</code>
<code>customClassNum</code>	<code>IccBase</code>
<code>handleEvent</code>	<code>IccResource</code>

メソッド	クラス
hours	IccTime
isEDFOn	IccResource
minutes	IccTime
operator delete	IccBase
operator new	IccBase
setActionOnAnyCondition	IccResource
setActionOnCondition	IccResource
setActionsOnConditions	IccResource
setEDF	IccResource
timeInHours	IccTime
timeInMinutes	IccTime
timeInSeconds	IccTime
type	IccTime

継承された protected メソッド

これらは、このクラスによって継承された protected メソッドです。

メソッド	クラス
setClassName	IccBase
setCustomClassNum	IccBase

IccTPNameId クラス

IccTPNameId クラスは、1 から 64 バイトの TP パートナー名を保持します。

```
IccBase
IccResourceId
IccTPNameId
```

IccTPNameId クラスは、1 から 64 バイトの TP パートナー名を保持します。

ヘッダー・ファイル: ICCRIDEH

IccTPNameId コンストラクター

コンストラクター (1)

```
IccTPNameId(const char* name)
```

name

1 から 64 文字の TP 名。

コンストラクター (2)

コピー・コンストラクター。

IccTPNameId(const IccTPNameId& *id*)

id

IccTPNameId オブジェクトへの参照。

public メソッド

これらは、このクラス内の public メソッドです。

operator= (1)

IccTPNameId& operator=(const char* *name*)

name

1 から 64 文字の TP 名。

operator= (2)

新規値を代入します。

IccTPNameId& operator=(const IccTPNameId& *id*)

id

IccTPNameId オブジェクトへの参照。

継承された public メソッド

これらは、このクラスによって継承された public メソッドです。

メソッド	クラス
classType	IccBase
className	IccBase
customClassNum	IccBase
name	IccResourceId
nameLength	IccResourceId
operator delete	IccBase
operator new	IccBase

継承された protected メソッド

これらは、このクラスによって継承された protected メソッドです。

メソッド	クラス
operator=	IccResourceId
setClassName	IccBase
setCustomClassNum	IccBase

IccTransId クラス

IccTransId クラスは、CICS システム内のトランザクション名を識別します。

IccBase
IccResourceId
IccTransId

ヘッダー・ファイル: ICCRIDEH

IccTransId コンストラクター

コンストラクター (1)

IccTransId(const char* *name*)

name

4 文字のトランザクション名。

コンストラクター (2)

コピー・コンストラクター。

IccTransId(const IccTransId& *id*)

id

IccTransId オブジェクトへの参照。

public メソッド

これらは、このクラス内の public メソッドです。

operator= (1)

IccTransId& operator=(const char* *name*)

name

4 文字のトランザクション名。

operator= (2)

新規値を代入します。

IccTransId& operator=(const IccTransId& *id*)

id

IccTransId オブジェクトへの参照。

継承された public メソッド

これらは、このクラスによって継承された public メソッドです。

メソッド	クラス
classType	IccBase
className	IccBase
customClassNum	IccBase
name	IccResourceId
nameLength	IccResourceId
operator delete	IccBase
operator new	IccBase

継承された protected メソッド

これらは、このクラスによって継承された protected メソッドです。

メソッド	クラス
operator=	IccResourceId
setClassName	IccBase
setCustomClassNum	IccBase

IccUser クラス

このクラスは、CICS ユーザーを表します。

IccBase

IccResource

IccUser

ヘッダー・ファイル : ICCUSREH

サンプル : ICC\$USR

IccUser コンストラクター

コンストラクター (1)

```
IccUser (const IccUserId& id,  
const IccGroupId* gid = 0)
```

id

ユーザー ID 名が含まれている **IccUserId** オブジェクトへの参照

gid

ユーザーのグループ ID に関する情報が含まれている **IccGroupId** オブジェクトへのオプション・ポインター。

コンストラクター (2)

```
IccUser (const char* userName,  
const char* groupName = 0)
```

userName

8 文字のユーザー ID

gid

オプションの 8 文字のグループ ID。

public メソッド

これらは、このクラス内の public メソッドです。

changePassword

ユーザーのパスワードの変更を試行します。

```
void changePassword (const char* password,  
const char* newPassword)
```

password

ユーザーの既存のパスワード (最大 8 文字のストリング)

newPassword

ユーザーの新規パスワード (最大 8 文字のストリング)

条件

INVREQ、NOTAUTH、USERIDERR

daysUntilPasswordExpires

パスワードが期限切れになるまでの日数を返します。このメソッドは、このクラスで **verifyPassword** メソッド呼び出しが正常に実行された後に有効です。

```
unsigned short daysUntilPasswordExpires() const
```

ESMReason

unsigned long ESMReason() const

changePassword または **verifyPassword** メソッド呼び出しが失敗した場合に、関連する外部セキュリティ理由コードを返します。

ESMResponse

unsigned long ESMResponse() const

changePassword または **verifyPassword** メソッド呼び出しが失敗した場合に、関連する外部セキュリティ応答コードを返します。

groupId

const IccGroupId& groupId() const

ユーザーのグループ ID に関する情報を保持する **IccGroupId** オブジェクトへの参照を返します。

invalidPasswordAttempts

unsigned long invalidPasswordAttempts() const

最後にサインオンに成功して以降に、このユーザーについて誤ったパスワードが入力された回数を返します。このメソッドは、**verifyPassword** メソッドが正常に完了した後にのみ使用してください。

language

const char* language() const

IccTerminal での **signon** を行う呼び出しが正常に完了した後に、ユーザーの言語を返します。

lastPasswordChange

const IccAbsTime& lastPasswordChange() const

パスワードが最後に変更された時刻を保持する **IccAbsTime** オブジェクトへの参照を返します。このメソッドは、**verifyPassword** メソッドが正常に完了した後にのみ使用してください。

lastUseTime

const IccAbsTime& lastUseTime() const

ユーザー ID が最後に使用された時刻を保持する **IccAbsTime** オブジェクトへの参照を返します。このメソッドは、**verifyPassword** メソッドが正常に完了した後にのみ使用してください。

passwordExpiration

const IccAbsTime& passwordExpiration() const

パスワードが期限切れになる時刻を保持する **IccAbsTime** オブジェクトへの参照を返します。このメソッドは、**verifyPassword** メソッドが正常に完了した後にのみ使用してください。

setLanguage

void setLanguage(const char* language)

このユーザーに関連付ける IBM 定義の各国語コードを設定します。これは、3 文字の値でなければなりません。

verifyPassword

void verifyPassword(const char* password)

提供されたパスワードが、この **IccUser** 用に外部セキュリティー・マネージャーが記録しているパスワードと一致しているかを検査します。

条件

INVREQ、NOTAUTH、USERIDERR

継承された public メソッド

これらは、このクラスによって継承された public メソッドです。

メソッド	クラス
actionOnCondition	IccResource
actionOnConditionAsChar	IccResource
actionsOnConditionsText	IccResource
classType	IccBase
className	IccBase
condition	IccResource
conditionText	IccResource
customClassNum	IccBase
handleEvent	IccResource
id	IccResource
isEDFOn	IccResource
name	IccResource
operator delete	IccBase
operator new	IccBase
setActionOnAnyCondition	IccResource
setActionOnCondition	IccResource
setActionsOnConditions	IccResource

メソッド

setEDF

クラス

IccResource

継承された protected メソッド

これらは、このクラスによって継承された protected メソッドです。

メソッド

setClassName

setCustomClassNum

クラス

IccBase

IccBase

IccUserId クラス

IccUserId クラスは、8 文字のユーザー名を表します。

IccBase

IccResourceId

IccUserId

IccUserId クラスは、8 文字のユーザー名を表します。

ヘッダー・ファイル: ICCRIDEH

IccUserId コンストラクター

コンストラクター (1)

IccUserId(const char* name)

name

ユーザー ID の 8 文字の名前。

コンストラクター (2)

コピー・コンストラクター。

IccUserId(const IccUserId& id)

id

IccUserId オブジェクトへの参照。

public メソッド

これらは、このクラス内の public メソッドです。

operator= (1)

IccUserId& operator=(const char* name)

name

ユーザー ID の 8 文字の名前。

operator= (2)

新規値を代入します。

IccUserId& operator=(const IccUserId& id)

id

IccUserId オブジェクトへの参照。

継承された public メソッド

これらは、このクラスによって継承された public メソッドです。

メソッド	クラス
classType	IccBase
className	IccBase
customClassNum	IccBase
name	IccResourceId
nameLength	IccResourceId
operator delete	IccBase
operator new	IccBase

継承された protected メソッド

これらは、このクラスによって継承された protected メソッドです。

メソッド	クラス
operator=	IccResourceId
setClassName	IccBase
setCustomClassNum	IccBase

IccValue 構造

この構造には、CICS 値データ域 (CVDA) が列挙型として含まれます。

ヘッダー・ファイル: ICCVALEH

列挙型

有効な CVDA のリスト

有効な CVDA は、システム・プログラミング参照情報の CVDA と数値に関するトピックにリストされています。

main 関数

このコードをアプリケーションに組み込むことをお勧めします。

CICS ファウンデーション・クラスを正しく初期化し、デフォルト例外処理を提供し、割り振られたメモリーを完了後に解放します。この **main** 関数を独自のバリエーションに置き換えることもできますが、この操作はほとんどの場合、必要ありません。

ソース・ファイル: ICCMAIN

スタブには、以下の 3 つの機能があります。

1. ファウンデーション・クラス環境を初期化します。この方法をカスタマイズするには、以下を制御する **#define** を使用します。
 - ・メモリー管理 (ストレージ管理を参照)
 - ・ファミリー・サブセットの制約 ([『61 ページの『FamilySubset』』](#) を参照)
 - ・EDF 使用可能化 (プログラムのデバッグを参照)
2. **IccControl** から派生した **IccUserControl** クラスのデフォルト定義 (デフォルト・コンストラクター、**run** メソッドなど) を行います。
3. **try catch** 構成体を使用して、ユーザーの制御オブジェクトの **run** メソッドを呼び出します。

次の情報は、**main** コードの関数部分です。

```
int main() 1
{
    Icc::initializeEnvironment(ICC_CLASS_MEMORY_MGMT, 2
                                ICC_FAMILY_SUBSET,
                                ICC_EDF_BOOL);
    try 3
    {
        ICC_USER_CONTROL control; 4
        control.run(); 5
    }
    catch(IccException& exc) 6
    {
        Icc::catchException(exc); 7
    }
    catch(...) 8
    {
        Icc::unknownException(); 9
    }
    Icc::returnToCICS(); 10
}
```

1

これは C++ の main エントリー・ポイントです。

2

この呼び出しにより環境が初期化されます。この呼び出しは必須です。3 つのパラメーターは、既にプラットフォームのデフォルトに定義されています。

3

ユーザーのアプリケーション・コードが例外をキャッチしない場合は、**try** および **catch** を使用して、そのアプリケーション・コードを実行します。

4

制御オブジェクトを作成します。

5

制御オブジェクトの **run** メソッド (**IccControl** で純粋仮想として定義されている) を呼び出します。

- 6** アプリケーションによってキャッチされていない **IceException** オブジェクトをキャッチします。
- 7** この関数を呼び出して、タスクを異常終了します。
- 8** アプリケーションによってキャッチされていないその他の例外をキャッチします。
- 9** この関数を呼び出して、タスクを異常終了します。
- 10** CICS に制御を返します。

特記事項

本書は米国 IBM が提供する製品およびサービスについて作成したものです。この資料の他の言語版を IBM から入手できる場合があります。ただし、これを入手するには、本製品または当該言語版製品を所有している必要がある場合があります。

本書に記載の製品、サービス、または機能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、および機能については、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。IBM 製品、プログラムまたはサービスに代えて、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せん。ただし、IBM 以外の製品とプログラムの操作またはサービス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。実施権についてのお問い合わせは、書面にて下記宛先にお送りください。

〒 103-8510

東京都中央区日本橋箱崎町 19 番 21 号

日本アイ・ビー・エム株式会社

法務・知的財産

知的財産権ライセンス涉外

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を提供し、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。

この情報には、技術的に不適切な記述や誤植を含む場合があります。本書は定期的に見直され、必要な変更は本書の次版に組み込まれます。IBM は予告なしに、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、お客様自身の責任でご使用ください。

IBM は、お客様が提供するいかなる情報も、お客様に対してなんら義務も負うことのない、自ら適切と信ずる方法で、使用もしくは配布することができるものとします。

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

IBM Director of Licensing

IBM Corporation

North Castle Drive, MD-NC119 Armonk,

NY 10504-1785

United States of America

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。したがって、他社製品に関す

る実行性、互換性、またはその他の要求については確認できません。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をお願いします。

本書には、日常の業務処理で用いられるデータや報告書の例が含まれています。より具体性を与えるために、それらの例には、個人、企業、ブランド、あるいは製品などの名前が含まれている場合があります。これらの名前はすべて架空のものであり、類似する個人や企業が実在しているとしても、それは偶然にすぎません。

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、サンプル・プログラムが書かれているオペレーティング・プラットフォーム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めかしたり、保証することはできません。これらのサンプル・プログラムは特定物として現存するままの状態を提供されるものであり、いかなる保証も提供されません。IBM は、お客様の当該サンプル・プログラムの使用から生ずるいかなる損害に対しても一切の責任を負いません。

プログラミング・インターフェース情報

CICS には、プログラミング・インターフェースと見なすことのできる資料と、プログラミング・インターフェースと見なすことのできない資料があります。

オンライン製品資料の以下のセクションには、CICS Transaction Server for z/OS, バージョン 5 リリース 6 のサービスを取得するプログラムをお客様が作成するためのプログラミング・インターフェースが含まれています。

- [アプリケーションの開発](#)
- [システム・プログラムの開発](#)
- [CICS TS セキュリティー](#)
- [外部インターフェースに向けた開発](#)
- [アプリケーション開発のリファレンス](#)
- [リファレンス: システム・プログラミング](#)
- [リファレンス: 接続](#)

オンライン製品資料の以下のセクションには、CICS Transaction Server for z/OS, バージョン 5 リリース 6 のプログラミング・インターフェースとして意図されていない (プログラミング・インターフェースと誤解される可能性のある) 情報が含まれています。

- [トラブルシューティングおよびサポート](#)
- [CICS TS 診断参照](#)

PDF 形式のマニュアルで CICS 資料にアクセスする場合は、CICS Transaction Server for z/OS, バージョン 5 リリース 6 のサービスを取得するプログラムをお客様が作成するためのプログラミング・インターフェースが以下のマニュアルに含まれています。

- [アプリケーション・プログラミング・ガイドおよびアプリケーション・プログラミング・リファレンス](#)
- [Business Transaction Services](#)
- [Customization Guide](#)
- [C++ OO Class Libraries](#)
- [Debugging Tools Interfaces Reference](#)
- [Distributed Transaction Programming Guide](#)
- [External Interfaces Guide](#)
- [Front End Programming Interface Guide](#)

- IMS Database Control Guide
- インストール・ガイド
- セキュリティー・ガイド
- Supplied Transactions
- CICSplex® SM Managing Workloads
- CICSplex SM Managing Resource Usage
- CICSplex SM アプリケーション・プログラミング・ガイドおよび CICSplex SM アプリケーション・プログラミング・リファレンス
- CICS における Java™ アプリケーション

PDF 形式のマニュアルで CICS 資料にアクセスする場合は、CICS Transaction Server for z/OS, バージョン 5 リリース 6 のプログラミング・インターフェースとして意図されていない (プログラミング・インターフェースと誤解される可能性のある) 情報が以下のマニュアルに含まれています。

- Data Areas
- Diagnosis Reference
- Problem Determination Guide
- CICSplex SM Problem Determination Guide

商標

IBM、IBM ロゴおよび ibm.com® は、世界の多くの国で登録された International Business Machines Corporation の商標または登録商標です。他の製品名およびサービス名等は、それぞれ IBM または各社の商標である場合があります。現時点での IBM の商標リストについては、<http://www.ibm.com/legal/copytrade.shtml> をご覧ください。

Adobe、Adobe ロゴ、PostScript、PostScript ロゴは、Adobe Systems Incorporated の米国およびその他の国における登録商標または商標です。

インテル、Intel、Intel ロゴ、Intel Inside、Intel Inside ロゴ、Intel Centrino、Intel Centrino ロゴ、Celeron、Intel Xeon、Intel SpeedStep、Itanium、および Pentium は、Intel Corporation または子会社の米国およびその他の国における商標または登録商標です。

Java およびすべての Java 関連の商標およびロゴは Oracle やその関連会社の米国およびその他の国における商標または登録商標です。

Linux® は、Linus Torvalds の米国およびその他の国における登録商標です。

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

UNIX は The Open Group の米国およびその他の国における登録商標です。

製品資料に関するご使用条件

これらの資料は、以下のご使用条件に同意していただける場合に限りご使用いただけます。

適用範囲

IBM Web サイトの「ご利用条件」に加えて、以下のご使用条件が適用されます。

個人使用

これらの資料は、すべての著作権表示その他の所有権表示をしていただくことを条件に、非商業的な個人による使用目的に限り複製することができます。ただし、IBM の明示的な承諾をえずに、これらの資料またはその一部について、二次的著作物を作成したり、配布 (頒布、送信を含む) または表示 (上映を含む) することはできません。

商用使用

これらの資料は、すべての著作権表示その他の所有権表示をしていただくことを条件に、お客様の企業内に限り、複製、配布、および表示することができます。ただし、IBM の明示的な承諾をえずにこれらの資料の二次的著作物を作成したり、お客様の企業外で資料またはその一部を複製、配布、または表示することはできません。

権利

ここで明示的に許可されているもの以外に、資料や資料内に含まれる情報、データ、ソフトウェア、またはその他の知的所有権に対するいかなる許可、ライセンス、または権利を明示的にも黙示的にも付与するものではありません。

資料の使用が IBM の利益を損なうと判断された場合や、上記の条件が適切に守られていないと判断された場合、IBM はいつでも自らの判断により、ここで与えた許可を撤回できるものとさせていただきます。

お客様がこの情報をダウンロード、輸出、または再輸出する際には、米国のすべての輸出入関連法規を含む、すべての関連法規を遵守するものとします。

IBM これらの資料の内容についていかなる保証もしません。これらの資料は、特定物として現存するままの状態を提供され、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なしで提供されます。

IBM オンラインでのプライバシー・ステートメント

サービス・ソリューションとしてのソフトウェアも含めた IBM ソフトウェア製品 (ソフトウェア・オファリング) では、製品の使用に関する情報の収集、エンド・ユーザーの使用感の向上、エンド・ユーザーとの対話またはその他の目的のために、Cookie はじめさまざまなテクノロジーを使用することがあります。多くの場合、ソフトウェア・オファリングにより個人情報が収集されることはありません。IBM の「ソフトウェア・オファリング」の一部には、個人情報を収集できる機能を持つものがあります。ご使用の「ソフトウェア・オファリング」が、これらの Cookie およびそれに類するテクノロジーを通じてお客様による個人情報の収集を可能にする場合、以下の具体的事項をご確認ください。

CICSplex SM Web ユーザー・インターフェース (メイン・インターフェース) の場合:

このソフトウェア・オファリングは、展開される構成に応じて、セッション管理、認証、お客様の利便性の向上、または利用の追跡または機能上の目的のために、それぞれのお客様のユーザー名、およびその他の個人情報を、セッションごとの Cookie および持続的な Cookie を使用して収集する場合があります。これらの Cookie を無効にすることはできません。

CICSplex SM Web ユーザー・インターフェース (データ・インターフェース) の場合:

このソフトウェア・オファリングは、展開される構成に応じて、セッション管理、認証、または利用の追跡または機能上の目的のために、それぞれのお客様のユーザー名またはその他の個人情報を、セッションごとの Cookie を使用して収集する場合があります。これらの Cookie を無効にすることはできません。

CICSplex SM Web ユーザー・インターフェース (「Hello World」ページ) の場合:

このソフトウェア・オファリングは、展開される構成に応じて、個人情報を収集しないセッションごとの Cookie を使用する場合があります。これらの Cookie を無効にすることはできません。

CICS Explorer® の場合:

このソフトウェア・オファリングは、展開される構成に応じて、セッション管理、お客様の利便性の向上、または利用の追跡または機能上の目的のために、それぞれのお客様のユーザー名、およびその他の個人情報を、セッションごとの設定および持続的な設定を使用して収集する場合があります。これらの設定を無効にすることはできませんが、ユーザー・パスワードの暗号化形式でのディスクへの保管は、サインオン中にチェック・ボックスにチェック・マークを付けることによるユーザーの明示的な操作によってのみ有効化することができます。

この「ソフトウェア・オファリング」が Cookie およびさまざまなテクノロジーを使用してエンド・ユーザーから個人を特定できる情報を収集する機能を提供する場合、お客様は、このような情報を収集するにあたって適用される法律、ガイドライン等を遵守する必要があります。これには、エンドユーザーへの通知や同意の要求も含まれますがそれらには限られません。

このような目的での Cookie を含む様々なテクノロジーの使用の詳細については、『IBM オンラインでのプライバシー・ステートメント』 (<http://www.ibm.com/privacy/details/jp/ja/>) の『クッキー、ウェブ・ビー

コン、その他のテクノロジー』および『IBM Software Products and Software-as-a-Service Privacy Statement』 (<http://www.ibm.com/software/info/product-privacy>) を参照してください。

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]

異常終了

IccTask クラス内 [194](#)

異常終了コード [33](#)

インストール済みの内容

場所 [2](#)

オープン

Status [124](#)

オブジェクト (object)

GetOpt [61](#)

オプション

列挙型 [134](#)

IccJournal クラス内 [134](#)

[カ行]

外部

DataAreaOwner 内 [86](#)

関数

boolText [58](#)

catchException [58](#)

conditionText [58](#)

Icc 構造内 [58](#)

initializeEnvironment [59](#)

isClassMemoryMgmtOn [59](#)

isEDFOn [59](#)

isFamilySubsetEnforcementOn [59](#)

returnToCICS [60](#)

setEDF [60](#)

unknownException [60](#)

完全

Kind [141](#)

完全キー [16](#)

継承された protected メソッド

IccAbendData クラス内 [66](#)

IccAbsTime クラス内 [71](#)

IccAlarmRequestId クラス内 [73](#)

IccBuf クラスにおける [86](#)

IccClock クラス内 [90](#)

IccConsole クラス内 [96](#)

IccControl クラス内 [101](#)

IccConvId クラス内 [102](#)

IccDataQueue クラス内 [105](#)

IccDataQueueId クラス内 [107](#)

IccEvent クラス内 [109](#)

IccException クラス内 [111](#)

IccFile クラス内 [123](#)

IccFileId クラス内 [125](#)

IccFileIterator クラス内 [128](#)

IccGroupId クラス内 [130](#)

IccJournal クラス内 [134](#)

IccJournalId クラス内 [136](#)

IccJournalTypeId クラス内 [137](#)

継承された protected メソッド (続き)

IccKey クラス内 [141](#)

IccLockId クラス内 [143](#)

IccMessage クラス内 [144](#)

IccPartnerId クラス内 [146](#)

IccProgram クラス内 [150](#)

IccProgramId クラス内 [151](#)

IccRBA クラス内 [153](#)

IccRecordIndex クラス内 [155](#)

IccRequestId クラス内 [157](#)

IccResource クラス内 [162](#)

IccResourceId クラス内 [165](#)

IccSemaphore クラス内 [169](#)

IccSession クラス内 [179](#)

IccStartRequestQ クラス内 [186](#)

IccSysId クラス内 [188](#)

IccSystem クラス内 [193](#)

IccTask クラス内 [202](#)

IccTempStore クラス内 [208](#)

IccTempStoreId クラス内 [210](#)

IccTermId クラス内 [211](#)

IccTerminal クラス内 [225](#)

IccTerminalData クラス内 [232](#)

IccTime クラス内 [235](#)

IccTimeInterval クラス内 [237](#)

IccTimeOfDay クラス内 [239](#)

IccTransId クラス内 [242](#)

IccUser クラス内 [246](#)

IccUserId クラス内 [247](#)

継承された public メソッド

IccAbendData クラス内 [66](#)

IccAbsTime クラス内 [70](#)

IccAlarmRequestId クラス内 [73](#)

IccBuf クラスにおける [86](#)

IccClock クラス内 [90](#)

IccConsole クラス内 [96](#)

IccControl クラス内 [100](#)

IccConvId クラス内 [102](#)

IccDataQueue クラス内 [105](#)

IccDataQueueId クラス内 [107](#)

IccEvent クラス内 [109](#)

IccException クラス内 [111](#)

IccFile クラス内 [122](#)

IccFileId クラス内 [125](#)

IccFileIterator クラス内 [127](#)

IccGroupId クラス内 [129](#)

IccJournal クラス内 [133](#)

IccJournalId クラス内 [136](#)

IccJournalTypeId クラス内 [137](#)

IccKey クラス内 [141](#)

IccLockId クラス内 [142](#)

IccMessage クラス内 [144](#)

IccPartnerId クラス内 [146](#)

IccProgram クラス内 [149](#)

IccProgramId クラス内 [151](#)

IccRBA クラス内 [153](#)

IccRecordIndex クラス内 [155](#)

継承された public メソッド (続き)

IccRequestId クラス内 [156](#)
IccResourceId クラス内 [164](#)
IccRRN クラス内 [166](#)
IccSemaphore クラス内 [169](#)
IccSession クラス内 [179](#)
IccStartRequestQ クラス内 [185](#)
IccSysId クラス内 [187](#)
IccSystem クラス内 [192](#)
IccTask クラス内 [201](#)
IccTempStore クラス内 [207](#)
IccTempStoreId クラス内 [209](#)
IccTermId クラス内 [211](#)
IccTerminal クラス内 [225](#)
IccTerminalData クラス内 [232](#)
IccTime クラス内 [234](#)
IccTimeInterval クラス内 [236](#)
IccTimeOfDay クラス内 [238](#)
IccTPNameId クラス内 [240](#)
IccTransId クラス内 [242](#)
IccUser クラス内 [245](#)
IccUserId クラス内 [247](#)

更新、レコードの [17](#)

更新

IccClock クラス内 [89](#)
ReadMode [123](#)

コード

列挙型 [92](#)
IccCondition 構造内 [92](#)

コンストラクター

IccAbendData クラス内 [62](#)
IccAbendData コンストラクター (protected) [62](#)
IccAbsTime クラス内 [67](#)
IccAbsTime コンストラクター [67](#)
IccAlarmRequestId クラス内 [71](#)
IccAlarmRequestId コンストラクター [71](#)
IccBase クラス内 [73](#)
IccBase コンストラクター (protected) [73](#)
IccBuf クラスにおける [77, 78](#)
IccBuf コンストラクターにおける [77, 78](#)
IccClock クラス内 [87](#)
IccClock コンストラクター [87](#)
IccConsole クラス内 [93](#)
IccConsole コンストラクター (protected) [93](#)
IccControl クラス内 [97](#)
IccControl コンストラクター (protected) [97](#)
IccConvId クラス内 [101, 102](#)
IccConvId コンストラクター [101, 102](#)
IccDataQueue クラス内 [103](#)
IccDataQueue コンストラクター [103](#)
IccDataQueueId クラス内 [106](#)
IccDataQueueId コンストラクター [106](#)
IccEvent クラス内 [107](#)
IccEvent コンストラクター [107](#)
IccException クラス内 [109](#)
IccException コンストラクター [109](#)
IccFile クラス内 [113](#)
IccFile コンストラクター [113](#)
IccFileId クラス内 [124](#)
IccFileId コンストラクター [124](#)
IccFileIterator クラス内 [126](#)
IccFileIterator コンストラクター [126](#)
IccGroupId クラス内 [128, 129](#)
IccGroupId コンストラクター [128, 129](#)

コンストラクター (続き)

IccJournal クラス内 [130](#)
IccJournal コンストラクター [130](#)
IccJournalId クラス内 [134, 135](#)
IccJournalId コンストラクター [134, 135](#)
IccJournalTypeId クラス内 [136](#)
IccJournalTypeId コンストラクター [136](#)
IccKey クラス内 [138](#)
IccKey コンストラクター [138](#)
IccLockId クラス内 [141, 142](#)
IccLockId コンストラクター [141, 142](#)
IccMessage クラス内 [143](#)
IccMessage コンストラクター [143](#)
IccPartnerId クラス内 [145](#)
IccPartnerId コンストラクター [145](#)
IccProgram クラス内 [146, 147](#)
IccProgram コンストラクター [146, 147](#)
IccProgramId クラス内 [150](#)
IccProgramId コンストラクター [150](#)
IccRBA クラス内 [152](#)
IccRBA コンストラクター [152](#)
IccRecordIndex クラス内 [154](#)
IccRecordIndex コンストラクター (protected) [154](#)
IccRequestId クラス内 [155, 156](#)
IccRequestId コンストラクター [155, 156](#)
IccResource クラス内 [157](#)
IccResource コンストラクター (protected) [157](#)
IccResourceId クラス内 [163, 164](#)
IccResourceId コンストラクター (protected) [163, 164](#)
IccRRN クラス内 [165](#)
IccRRN コンストラクター [165](#)
IccSemaphore クラス内 [167](#)
IccSemaphore コンストラクター [167](#)
IccSession クラス内 [170, 171](#)
IccSession コンストラクター (protected) [171](#)
IccSession コンストラクター (public) [170](#)
IccStartRequestQ クラス内 [181](#)
IccStartRequestQ コンストラクター (protected) [181](#)
IccSysId クラス内 [186, 187](#)
IccSysId コンストラクター [186, 187](#)
IccSystem クラス内 [188](#)
IccSystem コンストラクター (protected) [188](#)
IccTask クラス内 [194](#)
IccTask コンストラクター (protected) [194](#)
IccTempStore クラス内 [204](#)
IccTempStore コンストラクター [204](#)
IccTempStoreId クラス内 [208, 209](#)
IccTempStoreId コンストラクター [208, 209](#)
IccTermId クラス内 [210](#)
IccTermId コンストラクター [210](#)
IccTerminal クラス内 [212](#)
IccTerminal コンストラクター (protected) [212](#)
IccTerminalData クラス内 [227](#)
IccTerminalData コンストラクター (protected) [227](#)
IccTime クラス内 [233](#)
IccTime コンストラクター (protected) [233](#)
IccTimeInterval クラス内 [235](#)
IccTimeInterval コンストラクター [235](#)
IccTimeOfDay クラス内 [237, 238](#)
IccTimeOfDay コンストラクター [237, 238](#)
IccTPNameId クラス内 [239, 240](#)
IccTPNameId コンストラクター [239, 240](#)
IccTransId クラス内 [241](#)
IccTransId コンストラクター [241](#)

コンストラクター (続き)

IccUser クラス内 [242, 243](#)

IccUser コンストラクター [242, 243](#)

IccUserId クラス内 [246](#)

IccUserId コンストラクター [246](#)

コンソール

IccControl クラス内 [98](#)

[サ行]

削除、レコードの [17](#)

削除演算子 [5](#)

参照の有効範囲 [44](#)

サンプル・アプリケーションの実行 [2](#)

サンプル・ソース [2](#)

サンプル・ソース・コード

インストール済みの内容 [2](#)

場所 [2](#)

時刻サービス [29](#)

システム

IccControl クラス内 [100](#)

自動

UpdateMode [91](#)

自動削除 [5](#)

自動作成 [5](#)

主

main 関数における [248](#)

受信

IccSession クラス内 [176](#)

IccTerminal クラス内 [217](#)

使用可能

Status [124](#)

状況

列挙型 [124](#)

IccFile クラス内 [124](#)

条件 0 (NORMAL)

actionsOnConditionsText [158](#)

条件 1 (ERROR)

actionsOnConditionsText [158](#)

条件 2 (RDATT)

actionsOnConditionsText [158](#)

条件 3 (WRBRK)

actionsOnConditionsText [158](#)

条件 4 (ICCEOF)

actionsOnConditionsText [158](#)

条件 5 (EODS)

actionsOnConditionsText [158](#)

条件 6 (EOC)

actionsOnConditionsText [158](#)

状態

IccSession クラス内 [176](#)

使用不可

Status [124](#)

新規演算子 [5](#)

ストレージ

Location [208](#)

総称

Kind [141](#)

総称キー [16](#)

送信

IccSession クラス内 [176](#)

IccTerminal クラス内 [217, 218](#)

相対バイト・アドレス [15](#)

相対レコード番号 [15](#)

[タ行]

ダイナミック・リンク・ライブラリー
インストール済みの内容 [2](#)

場所 [2](#)

タイプ

列挙型 [112, 155, 235](#)

IccException クラス内 [112](#)

IccRecordIndex クラス内 [155](#)

IccTime クラス内 [235](#)

タスク

IccControl クラス内 [100](#)

LifeTime [170](#)

データの有効範囲 [44](#)

同期

Options [134](#)

動的削除 [5](#)

動的作成 [5](#)

[ナ行]

内部

DataAreaOwner 内 [86](#)

なし

FacilityType [203](#)

[ハ行]

場所

インストール済みの内容 [2](#)

サンプル・ソース・コード [2](#)

ダイナミック・リンク・ライブラリー [2](#)

ヘッダー・ファイル内 [2](#)

列挙型 [208](#)

IccTempStore クラス内 [208](#)

バッファー [12, 15](#)

パラメーター受け渡し [44](#)

範囲

列挙型 [93](#)

IccCondition 構造内 [93](#)

日付サービス [29](#)

ファイルの順次読み取り [17](#)

ファウンデーション・クラスの概要

リソース・オブジェクトでのメソッドの呼び出し [12](#)

リソース・オブジェクトの作成 [11](#)

プラットフォーム

列挙型 [61](#)

Icc 構造内 [61](#)

[マ行]

メイン

main 関数における [248](#)

メソッド

ファウンデーション・クラスの参照情報内 [45](#)

[ヤ行]

呼び出し規則 [44](#)

[ラ行]

例外 [33](#)

例外 (exception)

[TraceOpt 204](#)

レコードの再書き込み [17](#)

列挙

[オプション 134](#)

[コード 92](#)

[状況 124](#)

[タイプ 112, 155, 235](#)

[場所 208](#)

[範囲 93](#)

[プラットフォーム 61](#)

[AbendDumpOpt 202](#)

[AbendHandlerOpt 202](#)

[Access 123](#)

[ActionOnCondition 162](#)

[AIDVal 225](#)

[AllocateOpt 179](#)

[Bool 60](#)

[BoolSet 61](#)

[Case 225](#)

[CheckOpt 186](#)

[ClassMemoryMgmt 61](#)

[ClassType 75](#)

[Color 226](#)

[CommitOpt 150](#)

[ConditionType 163](#)

[DataAreaOwner 86](#)

[DataAreaType 86](#)

[DateFormat 90](#)

[DayOfWeek 91](#)

[DumpOpts 202](#)

[FacilityType 203](#)

[FamilySubset 61](#)

[GetOpt 61](#)

[HandleEventReturnOpt 163](#)

[Highlight 226](#)

[Icc 構造内 60](#)

[IccBase クラス内 75](#)

[IccBuf クラスにおける 86](#)

[IccClock クラス内 90](#)

[IccCondition 構造内 92](#)

[IccConsole クラス内 96](#)

[IccException クラス内 112](#)

[IccFile クラス内 123](#)

[IccJournal クラス内 134](#)

[IccKey クラス内 141](#)

[IccProgram クラス内 150](#)

[IccRecordIndex クラス内 155](#)

[IccResource クラス内 162](#)

[IccSemaphore クラス内 169](#)

[IccSession クラス内 179](#)

[IccStartRequestQ クラス内 186](#)

[IccSystem クラス内 193](#)

[IccTask クラス内 202](#)

[IccTempStore クラス内 208](#)

[IccTerminal クラス内 225](#)

[IccTime クラス内 235](#)

[Kind 141](#)

[LifeTime 170](#)

[LoadOpt 150](#)

[LockType 169](#)

[MonthOfYear 91](#)

[NameOpt 76](#)

[NextTransIdOpt 226](#)

列挙 (続き)

[NoSpaceOpt 208](#)

[ProtectOpt 186](#)

[ReadMode 123](#)

[ResourceType 193](#)

[RetrieveOpt 186](#)

[SearchCriterion 124](#)

[SendOpt 180](#)

[SeverityOpt 96](#)

[StartType 203](#)

[StateOpt 180](#)

[StorageOpts 203](#)

[SyncLevel 180](#)

[TraceOpt 204](#)

[UpdateMode 91](#)

[WaitPostType 204](#)

[WaitPurgeability 204](#)

列挙型

[CVDA 247](#)

[IccValue 構造内 247](#)

[数字]

0 (ゼロ)

[actionOnConditionAsChar 158](#)

A

A

[actionOnConditionAsChar 158](#)

[operatingSystem 191](#)

abendCode

[IccAbendData クラス内 62](#)

abendCode (パラメーター)

[abend 194](#)

abendData

[IccTask クラス内 194](#)

AbendDumpOpt

[列挙型 202](#)

[IccTask クラス内 202](#)

AbendHandlerOpt

[列挙型 202](#)

[IccTask クラス内 202](#)

absTime

[IccClock クラス内 87](#)

[Type 235](#)

absTime (パラメーター)

[コンストラクター 67](#)

[operator= 69](#)

access

[IccFile クラス内 113](#)

Access

[列挙型 123](#)

[IccFile クラス内 123](#)

access (パラメーター)

[setAccess 120](#)

accessMethod

[IccFile クラス内 114](#)

action (パラメーター)

[setActionOnAnyCondition 160, 161](#)

[setActionOnCondition 161](#)

ActionOnCondition

[列挙型 162](#)

- ActionOnCondition (続き)
 - IccResource クラス内 [162](#)
- actionOnCondition
 - IccResource クラス内 [157](#)
- actionOnConditionAsChar
 - IccResource クラス内 [158](#)
- actions (パラメーター)
 - setActionsOnConditions [161](#)
- actionsOnConditionsText
 - IccResource クラス内 [158](#)
- addable
 - Access [123](#)
- address
 - IccProgram クラス内 [147](#)
- AID
 - IccTerminal クラス内 [212](#)
- aid (パラメーター)
 - waitForAID [224](#)
- AIDVal
 - 列挙型 [225](#)
 - IccTerminal クラス内 [225](#)
- allocate
 - IccSession クラス内 [171](#)
- AllocateOpt
 - 列挙型 [179](#)
 - IccSession クラス内 [179](#)
- alternateHeight
 - IccTerminalData クラス内 [227](#)
 - public メソッド [227](#)
- alternateWidth
 - IccTerminalData クラス内 [227](#)
 - public メソッド [227](#)
- append
 - IccBuf クラスにおける [78](#)
- applName
 - IccSystem クラス内 [188](#)
- ASRAInterrupt
 - IccAbendData クラス内 [62](#)
 - public メソッド [62](#)
- ASRAKeyType
 - IccAbendData クラス内 [63](#)
 - public メソッド [63](#)
- ASRAPSW
 - IccAbendData クラス内 [63](#)
- ASRARegisters
 - IccAbendData クラス内 [63](#)
 - public メソッド [63](#)
- ASRASpaceType
 - IccAbendData クラス内 [64](#)
 - public メソッド [64](#)
- ASRAStorageType
 - IccAbendData クラス内 [65](#)
 - public メソッド [65](#)
- auxStorage
 - Location [208](#)

B

- baseName (パラメーター)
 - NameOpt [77](#)
- BASESPACE
 - ASRASpaceType [64](#)
- BDAM [15](#)
- beginBrowse

- beginBrowse (続き)
 - IccSystem クラス内 [188, 189](#)
- beginInsert (VSAM のみ)
 - IccFile クラス内 [114](#)
 - public メソッド [114](#)
- below
 - StorageOpts [203](#)
- blink
 - Highlight [226](#)
- blue
 - Color [226](#)
- Bool
 - 列挙型 [60](#)
 - Icc 構造内 [60](#)
- BoolSet
 - 列挙型 [61](#)
 - Icc 構造内 [61](#)
- boolText
 - 関数内 [58](#)
 - Icc 構造内 [58](#)
- browsable
 - Access [123](#)
- buf (パラメーター)
 - dump [195](#)
 - put [216](#)
 - send3270Data [219](#)
 - sendLine [220](#)
 - setData [183](#)
 - setData の [183](#)
- byAddress
 - LockType [169](#)
- byValue
 - LockType [169](#)

C

- C++ 例外 [33](#)
- callingProgramId
 - IccControl クラス内 [97](#)
 - public メソッド [97](#)
- cancel
 - IccRequestId クラス内 [155](#)
 - IccStartRequestQ クラス内 [180, 181](#)
- cancelAbendHandler
 - IccControl クラス内 [97](#)
- cancelAlarm
 - IccClock クラス内 [87](#)
- Case
 - 列挙型 [225](#)
 - IccTerminal クラス内 [225](#)
- caseOpt (パラメーター)
 - receive [217](#)
 - receive3270Data [217](#)
- catchException
 - 関数内 [58](#)
 - Icc 構造内 [58](#)
- ch (パラメーター)
 - operator« [83, 214](#)
- changePassword
 - IccUser クラス内 [243](#)
 - public メソッド [243](#)
- CheckOpt
 - 列挙型 [186](#)
 - IccStartRequestQ クラス内 [186](#)

CICS のその他のデータ・セット
インストール済みの内容 [3](#)
CICS リソース [11](#)
CICSDataKey
StorageOpts [203](#)
CICSEXECKEY
ASRAKeyType [63](#)
CICSInternalTask
StartType [203](#)
ClassMemoryMgmt
列挙型 [61](#)
Icc 構造内 [61](#)
className
IccBase クラス内 [74](#)
IccEvent クラス内 [108](#)
IccException クラス内 [110](#)
IccMessage クラス内 [143](#)
className (パラメーター)
コンストラクター [109](#), [110](#), [143](#)
setClassName [75](#)
classType
IccBase クラス内 [74](#)
IccEvent クラス内 [108](#)
IccException クラス内 [110](#)
ClassType
列挙型 [75](#)
IccBase クラス内 [75](#)
classType (パラメーター)
コンストラクター [109](#), [110](#), [157](#)
CLEAR
AIDVal [225](#)
clearData
IccStartRequestQ クラス内 [181](#)
clearInputMessage
IccProgram クラス内 [147](#)
clearPrefix
IccJournal クラス内 [131](#)
closed
Status [124](#)
col (パラメーター)
send [218](#)
send3270Data [219](#)
sendLine [220](#)
setCursor [221](#)
Color
列挙型 [226](#)
IccTerminal クラス内 [226](#)
color (パラメーター)
operator« [214](#)
setColor [221](#)
commArea
IccControl クラス内 [97](#)
commArea (パラメーター)
link [148](#)
setNextCommArea [222](#)
commitOnReturn
CommitOpt [150](#)
CommitOpt
列挙型 [150](#)
IccProgram クラス内 [150](#)
commitUOW
IccTask クラス内 [194](#)
completeLength
IccKey クラス内 [139](#)

completeLength (続き)
public メソッド [139](#)
completeLength (パラメーター)
コンストラクター [138](#)
condition (パラメーター)
actionOnCondition [158](#)
actionOnConditionAsChar [158](#)
conditionText [58](#), [59](#)
setActionOnCondition [161](#)
conditionText
関数内 [58](#)
Icc 構造内 [58](#)
IccEvent クラス内 [108](#)
IccResource クラス内 [159](#)
ConditionType
列挙型 [163](#)
IccResource クラス内 [163](#)
confirmation
SendOpt [180](#)
connectProcess
IccSession クラス内 [171](#), [172](#)
public メソッド [171](#), [172](#)
converse
IccSession クラス内 [172](#)
convId
IccSession クラス内 [173](#)
convId (パラメーター)
コンストラクター [102](#)
convName (パラメーター)
コンストラクター [101](#)
operator= [102](#)
copt (パラメーター)
setStartOpts [184](#)
createDump
AbendDumpOpt [202](#)
current (パラメーター)
setPrefix [132](#)
customClassNum
IccBase クラス内 [74](#)
public メソッド [74](#)
CVDA
列挙型 [247](#)
IccValue 構造内 [247](#)
cyan
Color [226](#)

D

data (パラメーター)
enterTrace [196](#)
put [176](#)
dataArea
IccBuf クラスにおける [80](#)
dataArea (パラメーター)
コンストラクター [77](#)
append [78](#)
assign [79](#), [138](#), [139](#)
insert [81](#)
overlay [85](#)
replace [85](#)
dataAreaLength
IccBuf クラスにおける [80](#)
public メソッド [80](#)
DataAreaOwner

- DataAreaOwner (続き)
 - 列挙型 [86](#)
 - IccBuf クラスにおける [86](#)
- DataAreaType
 - 列挙型 [86](#)
 - IccBuf クラスにおける [86](#)
- dataLength
 - IccBuf クラスにおける [80](#)
- dataqueue
 - FacilityType [203](#)
- dataQueueTrigger
 - StartType [203](#)
- date
 - IccAbsTime クラス内 [67](#)
 - IccClock クラス内 [88](#)
- DateFormat
 - 列挙型 [90](#)
 - IccClock クラス内 [90](#)
- dateFormat
 - IccSystem クラス内 [189](#)
- DayOfWeek
 - 列挙型 [91](#)
 - IccClock クラス内 [91](#)
- daysUntilPasswordExpires
 - IccUser クラス内 [243](#)
- dComplete
 - DumpOpts [202](#)
- dDCT
 - DumpOpts [203](#)
- dDefault
 - DumpOpts [202](#)
- defaultColor
 - Color [226](#)
- defaultHeight
 - IccTerminalData クラス内 [227](#)
 - public メソッド [227](#)
- defaultHighlight
 - Highlight [226](#)
- defaultWidth
 - IccTerminalData クラス内 [228](#)
 - public メソッド [228](#)
- deletable
 - Access [123](#)
- deleteLockedRecord
 - ロックされたレコードの削除における [17](#)
 - IccFile クラス内 [114](#)
- deleteRecord メソッド [17](#)
- dFCT
 - DumpOpts [203](#)
- DFHCURDI [3](#)
- DFHCURDS [2, 3](#)
- dPCT
 - DumpOpts [203](#)
- DPL
 - StartType [203](#)
- dPPT
 - DumpOpts [203](#)
- dProgram
 - DumpOpts [202](#)
- dSIT
 - DumpOpts [203](#)
- dStorage
 - DumpOpts [202](#)
- dTables

- dTables (続き)
 - DumpOpts [203](#)
- dTask
 - DumpOpts [202](#)
- dTCT
 - DumpOpts [203](#)
- dTerminal
 - DumpOpts [202](#)
- dTRT
 - DumpOpts [203](#)
- dump
 - IccTask クラス内 [195](#)
- dumpCode (パラメーター)
 - dump [195](#)
- DumpOpts
 - 列挙型 [202](#)
 - IccTask クラス内 [202](#)

E

- ECBList (パラメーター)
 - waitExternal [200](#)
- EDF (パラメーター)
 - initializeEnvironment 内 [59](#)
- enableStatus
 - IccFile クラス内 [115](#)
- endBrowse
 - IccSystem クラス内 [189](#)
- endInsert (VSAM のみ)
 - IccFile クラス内 [115](#)
 - public メソッド [115](#)
- ENTER
 - AIDVal [225](#)
- enterTrace
 - IccTask クラス内 [195](#)
- entryPoint
 - IccProgram クラス内 [147](#)
- equalToKey
 - SearchCriterion [124](#)
- errorCode
 - IccSession クラス内 [173](#)
- ESDS ファイル [15](#)
- ESMReason
 - IccUser クラス内 [244](#)
- ESMResponse
 - IccUser クラス内 [244](#)
- event (パラメーター)
 - handleEvent 内 [159](#)
- exception (パラメーター)
 - catchException [58](#)
- exceptionNum (パラメーター)
 - コンストラクター [109, 110](#)
- exceptionType (パラメーター)
 - コンストラクター [109, 110](#)
- extensible
 - DataAreaType 内 [86](#)
- extractProcess
 - IccSession クラス内 [173](#)
- extractState
 - StateOpt [180](#)

F

FacilityType
 [列挙型 203](#)
 IccTask クラス内 [203](#)
facilityType
 IccTask クラス内 [196](#)
fam (パラメーター)
 initializeEnvironment 内 [59](#)
FamilySubset
 [列挙型 61](#)
 Icc 構造内 [61](#)
FEPIRequest
 StartType [203](#)
fileName (パラメーター)
 コンストラクター [113](#), [124](#)
 getFile [190](#)
 operator= [125](#)
fixed
 DataAreaType 内 [86](#)
free
 IccSession クラス内 [173](#)
freeStorage
 IccSystem クラス内 [189](#)
 IccTask クラス内 [196](#)
fsDefault
 FamilySubset [61](#)
fullAccess
 Access [123](#)

G

getFile
 IccSystem クラス内 [189](#), [190](#)
getNextFile
 IccSystem クラス内 [190](#)
GetOpt
 [列挙型 61](#)
 Icc 構造内 [61](#)
getStorage
 IccSystem クラス内 [190](#)
 IccTask クラス内 [196](#)
gid (パラメーター)
 コンストラクター [243](#)
graphicCharCodeSet
 IccTerminalData クラス内 [228](#)
graphicCharSetId
 IccTerminalData クラス内 [228](#)
green
 Color [226](#)
groupId
 IccUser クラス内 [244](#)
groupName (パラメーター)
 コンストラクター [129](#), [243](#)
 operator= [129](#)
gteqToKey
 SearchCriterion [124](#)

H

H
 actionOnConditionAsChar [158](#)
HandleEventReturnOpt

HandleEventReturnOpt (続き)
 [列挙型 163](#)
 IccResource クラス内 [163](#)
handPost
 WaitPostType [204](#)
height
 IccTerminal クラス内 [213](#)
Highlight
 [列挙型 226](#)
 IccTerminal クラス内 [226](#)
highlight (パラメーター)
 operator« [214](#)
 setHighlight [221](#)
hold
 LoadOpt [150](#)
hours
 IccAbsTime クラス内 [68](#)
 IccTime クラス内 [233](#)
hours (パラメーター)
 コンストラクター [233](#), [235](#), [237](#)
 set [236](#), [238](#)

I

Icc 構造
 プラットフォーム [61](#)
 Bool [60](#)
 BoolSet [61](#)
 boolText [58](#)
 catchException [58](#)
 ClassMemoryMgmt [61](#)
 conditionText [58](#)
 FamilySubset [61](#)
 GetOpt [61](#)
 initializeEnvironment [59](#)
 isClassMemoryMgmtOn [59](#)
 isEDFOn [59](#)
 isFamilySubsetEnforcementOn [59](#)
 returnToCICS [60](#)
 setEDF [60](#)
 unknownException [60](#)
ICC\$BUF [3](#)
ICC\$CLK [3](#)
ICC\$HEL [3](#)
ICC\$SES1 [3](#)
ICC\$SES2 [3](#)
IccAbendData クラス
 コンストラクター [62](#)
 abendCode [62](#)
 ASRAInterrupt [62](#)
 ASRAKeyType [63](#)
 ASRAPSW [63](#)
 ASRARegisters [63](#)
 ASRASpaceType [64](#)
 ASRAStorageType [65](#)
 instance [65](#)
 isDumpAvailable [65](#)
 originalAbendCode [65](#)
 programName [65](#)
IccAbendData コンストラクター (protected)
 コンストラクター [62](#)
 IccAbendData クラス内 [62](#)
IccAbsTime クラス
 コンストラクター [67](#)

IccAbsTime クラス (続き)
 date [67](#)
 dayOfMonth [67](#)
 dayOfWeek [68](#)
 daysSince1900 [68](#)
 hours [68](#)
 milliseconds [68](#)
 minutes [68](#)
 monthOfYear [68](#)
 operator= [69](#)
 packedDecimal [69](#)
 seconds [69](#)
 time [69](#)
 timeInHours [69](#)
 timeInMinutes [69](#)
 timeInSeconds [70](#)
 year [70](#)

IccAbsTime コンストラクター
 コンストラクター [67](#)
 IccAbsTime クラス内 [67](#)

IccAlarmRequestId
 IccAlarmRequestId クラス内 [71](#)

IccAlarmRequestId クラス
 コンストラクター [71](#)
 isExpired [72](#)
 operator= [72](#)
 setTimerEca [72](#)
 timerEca [73](#)

IccAlarmRequestId コンストラクター
 コンストラクター [71](#)
 IccAlarmRequestId クラス内 [71](#)

IccBase コンストラクター (protected)
 コンストラクター [73](#)
 IccBase クラス内 [73](#)

IccBuf 参照 [44](#)

IccClock クラス
 更新 [89](#)
 コンストラクター [87](#)
 absTime [87](#)
 cancelAlarm [87](#)
 date [88](#)
 DateFormat [90](#)
 dayOfMonth [88](#)
 dayOfWeek [88](#)
 DayOfWeek [91](#)
 daysSince1900 [88](#)
 milliseconds [88](#)
 monthOfYear [89](#)
 MonthOfYear [91](#)
 setAlarm [89](#)
 time [89](#)
 UpdateMode [91](#)
 year [90](#)

IccClock コンストラクター
 コンストラクター [87](#)
 IccClock クラス内 [87](#)

IccCondition 構造体
 コード [92](#)
 範囲 [93](#)

IccConsole コンストラクター (protected)
 コンストラクター [93](#)
 IccConsole クラス内 [93](#)

IccControl コンストラクター (protected)
 コンストラクター [97](#)

IccControl コンストラクター (protected) (続き)
 IccControl クラス内 [97](#)

IccControl::run
 EXEC CICS 呼び出しとファウンデーション・クラス・メソッドのマッピング [46](#)

IccConvId
 IccConvId クラス内 [101](#)

IccConvId クラス
 コンストラクター [101](#), [102](#)
 operator= [102](#)

IccConvId コンストラクター
 コンストラクター [101](#), [102](#)
 IccConvId クラス内 [101](#)

IccDataQueue クラス
 コンストラクター [103](#)
 clear [103](#)
 empty [103](#)
 get [104](#)
 put [104](#)
 readItem [104](#)
 writeItem [104](#)

IccDataQueue コンストラクター
 コンストラクター [103](#)
 IccDataQueue クラス内 [103](#)

IccDataQueueId クラス
 コンストラクター [106](#)
 operator= [106](#)

IccDataQueueId コンストラクター
 コンストラクター [106](#)
 IccDataQueueId クラス内 [106](#)

IccEvent クラス
 コンストラクター [107](#)
 className [108](#)
 classType [108](#)
 condition [108](#)
 conditionText [108](#)
 methodName [108](#)
 summary [108](#)

IccEvent コンストラクター
 コンストラクター [107](#)
 IccEvent クラス内 [107](#)

IccException コンストラクター
 コンストラクター [109](#)
 IccException クラス内 [109](#)

ICCFCC [3](#)

ICCFCL [3](#)

ICCFGL [3](#)

ICCFIMP [3](#)

ICCFCL [3](#)

IccFile コンストラクター
 コンストラクター [113](#)
 IccFile クラス内 [112](#)

IccFileId
 基本クラスにおける [7](#)
 ファイル制御における [15](#)
 リソース識別クラスにおける [7](#), [8](#)
 IccFileId クラス内 [124](#)

IccFileId クラス
 概要 [7](#), [15](#)
 コンストラクター [124](#)
 読み取り、レコードの [15](#)
 operator= [125](#)

IccFileId コンストラクター
 コンストラクター [124](#)

IccFileId コンストラクター (続き)
 IccFileId クラス内 [124](#)
 IccFileIterator コンストラクター
 コンストラクター [126](#)
 IccFileIterator クラス内 [126](#)
 IccGroupId
 IccGroupId クラス内 [128](#)
 IccGroupId クラス
 コンストラクター [128](#), [129](#)
 operator= [129](#)
 IccGroupId コンストラクター
 コンストラクター [128](#), [129](#)
 IccGroupId クラス内 [128](#)
 IccJournal クラス
 オプション [134](#)
 コンストラクター [130](#)
 clearPrefix [131](#)
 journalTypeId [131](#)
 put [131](#)
 registerPrefix [131](#)
 setJournalTypeId [132](#)
 setPrefix [132](#)
 wait [132](#)
 writeRecord [133](#)
 IccJournal コンストラクター
 コンストラクター [130](#)
 IccJournal クラス内 [130](#)
 IccJournalId
 IccJournalId クラス内 [134](#)
 IccJournalId クラス
 コンストラクター [134](#), [135](#)
 number [135](#)
 operator= [135](#)
 IccJournalId コンストラクター
 コンストラクター [134](#), [135](#)
 IccJournalId クラス内 [134](#)
 IccJournalTypeId
 ファウンデーション・クラスの参照情報内 [45](#)
 IccJournalTypeId クラス内 [136](#)
 IccJournalTypeId クラス
 コンストラクター [136](#)
 operator= [137](#)
 IccJournalTypeId コンストラクター
 コンストラクター [136](#)
 IccJournalTypeId クラス内 [136](#)
 IccKey コンストラクター
 コンストラクター [138](#)
 IccKey クラス内 [138](#)
 IccLockId
 IccLockId クラス内 [141](#)
 IccLockId クラス
 コンストラクター [141](#), [142](#)
 operator= [142](#)
 IccLockId コンストラクター
 コンストラクター [141](#), [142](#)
 IccLockId クラス内 [141](#)
 IccMessage クラス
 コンストラクター [143](#)
 className [143](#)
 methodName [144](#)
 number [144](#)
 summary [144](#)
 text [144](#)
 IccMessage コンストラクター
 IccMessage コンストラクター (続き)
 コンストラクター [143](#)
 IccMessage クラス内 [143](#)
 IccPartnerId
 IccPartnerId クラス内 [145](#)
 IccPartnerId クラス
 コンストラクター [145](#)
 operator= [145](#)
 IccPartnerId コンストラクター
 コンストラクター [145](#)
 IccPartnerId クラス内 [145](#)
 IccProgram コンストラクター
 コンストラクター [146](#), [147](#)
 IccProgram クラス内 [146](#)
 IccProgramId クラス
 コンストラクター [150](#)
 operator= [151](#)
 IccProgramId コンストラクター
 コンストラクター [150](#)
 IccProgramId クラス内 [150](#)
 IccRBA コンストラクター
 コンストラクター [152](#)
 IccRBA クラス内 [152](#)
 IccRecordIndex クラス
 コンストラクター [154](#)
 タイプ [155](#)
 length [154](#)
 type [154](#)
 IccRecordIndex コンストラクター (protected)
 コンストラクター [154](#)
 IccRecordIndex クラス内 [154](#)
 IccRequestId クラス
 コンストラクター [155](#), [156](#)
 operator= [156](#)
 IccRequestId コンストラクター
 コンストラクター [155](#), [156](#)
 IccRequestId クラス内 [155](#)
 IccResource コンストラクター (protected)
 コンストラクター [157](#)
 IccResource クラス内 [157](#)
 IccResourceId コンストラクター (protected)
 コンストラクター [163](#), [164](#)
 IccResourceId クラス内 [163](#)
 IccRRN コンストラクター
 コンストラクター [165](#)
 IccRRN クラス内 [165](#)
 IccSemaphore クラス
 コンストラクター [167](#)
 lifeTime [168](#)
 LifeTime [170](#)
 lock [168](#)
 LockType [169](#)
 tryLock [168](#)
 type [168](#)
 unlock [168](#)
 IccSemaphore コンストラクター
 コンストラクター [167](#)
 IccSemaphore クラス内 [167](#)
 IccSession クラス
 コンストラクター [170](#), [171](#)
 受信 [176](#)
 状態 [178](#)
 送信 [176](#)
 allocate [171](#)

IccSession クラス (続き)
 AllocateOpt [179](#)
 connectProcess [171](#), [172](#)
 converse [172](#)
 convId [173](#)
 errorCode [173](#)
 extractProcess [173](#)
 flush [173](#)
 free [173](#)
 get [174](#)
 isErrorSet [174](#)
 isNoDataSet [174](#)
 isSignalSet [174](#)
 issueAbend [174](#)
 issueConfirmation [174](#)
 issueError [175](#)
 issuePrepare [175](#)
 issueSignal [175](#)
 PIPList [175](#)
 process [175](#)
 put [175](#)
 sendInvite [176](#), [177](#)
 sendLast [177](#)
 SendOpt [180](#)
 StateOpt [180](#)
 stateText [178](#)
 syncLevel [179](#)
 SyncLevel [180](#)
 IccSession コンストラクター (protected)
 コンストラクター [171](#)
 IccSession クラス内 [171](#)
 IccSession コンストラクター (public)
 コンストラクター [170](#)
 IccSession クラス内 [170](#)
 IccStartRequestQ コンストラクター (protected)
 コンストラクター [181](#)
 IccStartRequestQ クラス内 [180](#)
 IccSysId クラス
 コンストラクター [186](#), [187](#)
 operator= [187](#)
 IccSysId コンストラクター
 コンストラクター [186](#), [187](#)
 IccSysId クラス内 [186](#)
 IccSystem コンストラクター (protected)
 コンストラクター [188](#)
 IccSystem クラス内 [188](#)
 IccTask コンストラクター (protected)
 コンストラクター [194](#)
 IccTask クラス内 [194](#)
 IccTempStore クラス
 コンストラクター [204](#)
 場所 [208](#)
 clear [205](#)
 empty [205](#)
 get [205](#)
 NoSpaceOpt [208](#)
 numberOfItems [205](#)
 put [205](#)
 readItem [206](#)
 readNextItem [206](#)
 rewriteItem [206](#)
 writeItem [206](#), [207](#)
 IccTempStore コンストラクター
 コンストラクター [204](#)
 IccTempStore コンストラクター (続き)
 IccTempStore クラス内 [204](#)
 IccTempStoreId クラス
 コンストラクター [208](#), [209](#)
 operator= [209](#)
 IccTempStoreId コンストラクター
 コンストラクター [208](#), [209](#)
 IccTempStoreId クラス内 [208](#)
 IccTermId コンストラクター
 コンストラクター [210](#)
 IccTermId クラス内 [210](#)
 IccTerminal クラス
 コンストラクター [212](#)
 受信 [217](#)
 送信 [217](#), [218](#)
 データ [212](#)
 AID [212](#)
 AIDVal [225](#)
 Case [225](#)
 clear [212](#)
 Color [226](#)
 cursor [212](#)
 erase [212](#)
 freeKeyboard [213](#)
 get [213](#)
 height [213](#)
 Highlight [226](#)
 inputCursor [213](#)
 instance [213](#)
 line [213](#)
 netName [213](#)
 NextTransIdOpt [226](#)
 operator« [214–216](#)
 put [216](#)
 receive3270Data [217](#)
 registerInputMessage [148](#)
 send3270Data [218](#), [219](#)
 sendLine [219](#), [220](#)
 setColor [221](#)
 setCursor [221](#)
 setHighlight [221](#)
 setLine [222](#)
 setNewLine [222](#)
 setNextCommArea [222](#)
 setNextInputMessage [222](#)
 setNextTransId [223](#)
 signoff [223](#)
 signon [223](#)
 waitForAID [224](#)
 width [224](#)
 workArea [224](#)
 IccTerminal コンストラクター (protected)
 コンストラクター [212](#)
 IccTerminal クラス内 [212](#)
 IccTerminalData クラス
 コンストラクター [227](#)
 alternateHeight [227](#)
 alternateWidth [227](#)
 defaultHeight [227](#)
 defaultWidth [228](#)
 graphicCharCodeSet [228](#)
 graphicCharSetId [228](#)
 isAPLKeyboard [228](#)
 isAPLText [229](#)

IccTerminalData クラス (続き)

- [isBTrans 229](#)
- [isColor 229](#)
- [isEWA 229](#)
- [isExtended3270 229](#)
- [isFieldOutline 230](#)
- [isGoodMorning 230](#)
- [isHighlight 230](#)
- [isKatakana 230](#)
- [isMSRControl 231](#)
- [isPS 231](#)
- [isSOSI 231](#)
- [isTextKeyboard 231](#)
- [isTextPrint 231](#)
- [isValidation 232](#)

IccTerminalData コンストラクター (protected)

- [コンストラクター 227](#)

- [IccTerminalData クラス内 227](#)

IccTime コンストラクター (protected)

- [コンストラクター 233](#)

- [IccTime クラス内 233](#)

IccTimeInterval クラス

- [コンストラクター 235](#)

- [operator= 236](#)

- [set 236](#)

IccTimeInterval コンストラクター

- [コンストラクター 235](#)

- [IccTimeInterval クラス内 235](#)

IccTimeOfDay クラス

- [コンストラクター 237, 238](#)

- [operator= 238](#)

- [set 238](#)

IccTimeOfDay コンストラクター

- [コンストラクター 237, 238](#)

- [IccTimeOfDay クラス内 237](#)

IccTPNameId

- [IccTPNameId クラス内 239](#)

IccTPNameId クラス

- [コンストラクター 239, 240](#)

- [operator= 240](#)

IccTPNameId コンストラクター

- [コンストラクター 239, 240](#)

- [IccTPNameId クラス内 239](#)

IccTransId コンストラクター

- [コンストラクター 241](#)

- [IccTransId クラス内 241](#)

IccUser クラス

- [コンストラクター 242, 243](#)

- [changePassword 243](#)

- [daysUntilPasswordExpires 243](#)

- [ESMReason 244](#)

- [ESMResponse 244](#)

- [groupId 244](#)

- [invalidPasswordAttempts 244](#)

- [language 244](#)

- [lastPasswordChange 244](#)

- [lastUseTime 244](#)

- [passwordExpiration 244](#)

- [setLanguage 245](#)

- [verifyPassword 245](#)

IccUser コンストラクター

- [コンストラクター 242, 243](#)

- [IccUser クラス内 242](#)

IccUserId

IccUserId (続き)

- [IccUserId クラス内 246](#)

IccUserId クラス

- [コンストラクター 246](#)

- [operator= 246, 247](#)

IccUserId コンストラクター

- [コンストラクター 246](#)

- [IccUserId クラス内 246](#)

IccValue

- [ファウンデーション・クラス: 参照 45](#)

IccValue 構造

- [CVDA 247](#)

id

- [IccResource クラス内 159](#)

id (パラメーター)

- [コンストラクター 71, 72, 103, 106, 113, 124, 125, 129, 130, 135, 137, 142, 145, 146, 150, 151, 156, 163, 168, 170, 187, 204, 209, 210, 240, 241, 243, 246](#)

- [getFile 190](#)

- [operator= 72, 102, 107, 125, 129, 135, 137, 142, 146, 151, 156, 164, 187, 209, 211, 240, 242, 247](#)

- [setJournalTypeId 132](#)

- [signon 223](#)

- [waitOnAlarm 201](#)

ifSOSReturnCondition

- [StorageOpts 203](#)

ignoreAbendHandler

- [AbendHandlerOpt 202](#)

immediate

- [NextTransIdOpt 226](#)

index (パラメーター)

- [コンストラクター 113, 126](#)

- [registerRecordIndex 120](#)

- [reset 127](#)

initByte (パラメーター)

- [getStorage 190, 196, 197](#)

initData

- [IccControl クラス内 98](#)

- [public メソッド 98](#)

initRBA (パラメーター)

- [コンストラクター 152](#)

initRRN (パラメーター)

- [コンストラクター 165](#)

initValue (パラメーター)

- [コンストラクター 138](#)

inputCursor

- [IccTerminal クラス内 213](#)

interval (パラメーター)

- [setReplyTimeout 94](#)

invalidPasswordAttempts

- [IccUser クラス内 244](#)

isAPLKeyboard

- [IccTerminalData クラス内 228](#)

- [public メソッド 228](#)

isAPLText

- [IccTerminalData クラス内 229](#)

- [public メソッド 229](#)

isBrowsable

- [IccFile クラス内 115](#)

isBTrans

- [IccTerminalData クラス内 229](#)

isClassMemoryMgmtOn

- [関数内 59](#)

- [Icc 構造内 59](#)

- isColor
 - IccTerminalData クラス内 [229](#)
- isCommandSecurityOn
 - IccTask クラス内 [197](#)
- isCommitSupported
 - IccTask クラス内 [197](#)
- isCreated
 - IccControl クラス内 [98](#)
- isDeletable
 - IccFile クラス内 [116](#)
- isDumpAvailable
 - IccAbendData クラス内 [65](#)
- isEDFOn
 - 関数内 [59](#)
 - Icc 構造内 [59](#)
 - IccResource クラス内 [159](#)
- isEmptyOnOpen
 - IccFile クラス内 [116](#)
- isErrorSet
 - IccSession クラス内 [174](#)
- isEWA
 - IccTerminalData クラス内 [229](#)
- isExpired
 - IccAlarmRequestId クラス内 [72](#)
- isExtended3270
 - IccTerminalData クラス内 [229](#)
 - public メソッド [229](#)
- isFamilySubsetEnforcementOn
 - 関数内 [59](#)
 - Icc 構造内 [59](#)
- isFieldOutline
 - IccTerminalData クラス内 [230](#)
 - public メソッド [230](#)
- isFMHContained
 - IccBuf クラスにおける [81](#)
 - public メソッド [81](#)
- isGoodMorning
 - IccTerminalData クラス内 [230](#)
 - public メソッド [230](#)
- isHighlight
 - IccTerminalData クラス内 [230](#)
- isKatakana
 - IccTerminalData クラス内 [230](#)
- isMSRControl
 - IccTerminalData クラス内 [231](#)
- isNoDataSet
 - IccSession クラス内 [174](#)
- isPS
 - IccTerminalData クラス内 [231](#)
- isReadable メソッド [16](#)
- isRecoverable
 - IccFile クラス内 [117](#)
- isResourceSecurityOn
 - IccTask クラス内 [197](#)
- isRestarted
 - IccTask クラス内 [198](#)
- isRouteOptionOn
 - IccResource クラス内 [160](#)
 - public メソッド [160](#)
- isSignalSet
 - IccSession クラス内 [174](#)
- isSOSI
 - IccTerminalData クラス内 [231](#)
- isStartDataAvailable

- isStartDataAvailable (続き)
 - IccTask クラス内 [198](#)
- issueAbend
 - IccSession クラス内 [174](#)
- issueConfirmation
 - IccSession クラス内 [174](#)
- issueError
 - IccSession クラス内 [175](#)
- issuePrepare
 - IccSession クラス内 [175](#)
- issueSignal
 - IccSession クラス内 [175](#)
- isTextKeyboard
 - IccTerminalData クラス内 [231](#)
 - public メソッド [231](#)
- isTextPrint
 - IccTerminalData クラス内 [231](#)
 - public メソッド [231](#)
- isUpdatable
 - IccFile クラス内 [117](#)
- isValidation
 - IccTerminalData クラス内 [232](#)
- item (パラメーター)
 - rewriteItem [206](#)
 - writeItem [104](#), [207](#)
- itemNum (パラメーター)
 - readItem [206](#)
 - rewriteItem [206](#)

J

- journalNum (パラメーター)
 - コンストラクター [131](#), [135](#)
 - operator= [135](#)
- journalTypeId
 - IccJournal クラス内 [131](#)
- journalTypeName (パラメーター)
 - コンストラクター [136](#)
 - operator= [137](#)
- jtypeid (パラメーター)
 - setJournalTypeId [132](#)

K

- keyLength メソッド [16](#)
- keyPosition メソッド [16](#)
- kind
 - IccKey クラス内 [139](#)
- Kind
 - 列挙型 [141](#)
 - IccKey クラス内 [141](#)
- kind (パラメーター)
 - コンストラクター [138](#)
 - setKind [140](#)
- KSDS ファイル [15](#)

L

- language
 - IccUser クラス内 [244](#)
- language (パラメーター)
 - setLanguage [245](#)
- lastCommand

- lastCommand (続き)
 - StateOpt [180](#)
- lastPasswordChange
 - IccUser クラス内 [244](#)
- lastUseTime
 - IccUser クラス内 [244](#)
- length
 - IccProgram クラス内 [147](#)
 - IccRecordIndex クラス内 [154](#)
- length (パラメーター)
 - コンストラクター [77](#)
 - append [78](#)
 - assign [79](#), [138](#)
 - cut [79](#)
 - insert [81](#)
 - overlay [85](#)
 - replace [85](#)
 - setDataLength [85](#)
- level (パラメーター)
 - connectProcess [171](#), [172](#)
- level0
 - SyncLevel [180](#)
- level1
 - SyncLevel [180](#)
- level2
 - SyncLevel [180](#)
- life (パラメーター)
 - コンストラクター [167](#), [168](#)
- lifeTime
 - IccSemaphore クラス内 [168](#)
- LifeTime
 - 列挙型 [170](#)
 - IccSemaphore クラス内 [170](#)
- lineNum (パラメーター)
 - setLine [222](#)
- link
 - IccProgram クラス内 [148](#)
- load
 - IccProgram クラス内 [148](#)
- LoadOpt
 - 列挙型 [150](#)
 - IccProgram クラス内 [150](#)
- loc (パラメーター)
 - コンストラクター [204](#), [205](#)
- lock
 - IccSemaphore クラス内 [168](#)
- LockType
 - 列挙型 [169](#)
 - IccSemaphore クラス内 [169](#)

M

- majorCode
 - ConditionType [163](#)
- manual
 - UpdateMode [91](#)
- maxValue
 - Range [93](#)
- mem (パラメーター)
 - initializeEnvironment 内 [59](#)
- message
 - IccException クラス内 [110](#)
- message (パラメーター)
 - コンストラクター [109](#), [110](#)

- message (パラメーター) (続き)
 - setNextInputMessage [222](#)
- methodName
 - IccEvent クラス内 [108](#)
 - IccException クラス内 [110](#)
 - IccMessage クラス内 [144](#)
- methodName (パラメーター)
 - コンストラクター [107](#), [109](#), [110](#), [143](#)
- milliseconds
 - IccAbsTime クラス内 [68](#)
 - IccClock クラス内 [88](#)
- minorCode
 - ConditionType [163](#)
- minutes
 - IccAbsTime クラス内 [68](#)
 - IccTime クラス内 [233](#)
- minutes (パラメーター)
 - コンストラクター [233](#), [235](#), [237](#)
 - set [236](#), [238](#)
- mixed
 - Case [226](#)
- mode (パラメーター)
 - readNextRecord [126](#)
 - readPreviousRecord [127](#)
 - readRecord [118](#)
- MonthOfYear
 - 列挙型 [91](#)
 - IccClock クラス内 [91](#)
- msg (パラメーター)
 - clearInputMessage [147](#)
 - registerInputMessage [148](#)
 - setInputMessage [149](#)
- MVSPost
 - WaitPostType [204](#)

N

- N
 - operatingSystem [191](#)
- name
 - IccResource クラス内 [160](#)
 - IccResourceId クラス内 [164](#)
- name (パラメーター)
 - コンストラクター [71](#), [141](#), [142](#), [186](#), [187](#), [209](#), [210](#), [239](#), [241](#), [246](#)
 - operator= [142](#), [187](#), [209](#), [211](#), [240-242](#), [247](#)
 - setWaitText [199](#)
- nameLength
 - IccResourceId クラス内 [164](#)
- NameOpt
 - 列挙型 [76](#)
 - IccBase クラス内 [76](#)
- netName
 - IccTerminal クラス内 [213](#)
- neutral
 - Color [226](#)
- newPassword (パラメーター)
 - changePassword [243](#)
 - signon [223](#), [224](#)
- NextTransIdOpt
 - 列挙型 [226](#)
 - IccTerminal クラス内 [226](#)
- noAccess
 - Access [123](#)

- noCommitOnReturn
 - CommitOpt [150](#)
- NONCICS
 - ASRAKeyType [63](#)
- noQueue
 - AllocateOpt [180](#)
- normal
 - ReadMode [123](#)
 - SendOpt [180](#)
 - TraceOpt [204](#)
- NoSpaceOpt
 - 列挙型 [208](#)
 - IccTempStore クラス内 [208](#)
- noSuspend
 - Options [134](#)
- notAddable
 - Access [123](#)
- NOTAPPLIC
 - ASRAKeyType [63](#)
 - ASRASpaceType [64](#)
 - ASRAStorageType [64](#)
- notBrowsable
 - Access [123](#)
- notDeletable
 - Access [123](#)
- notPurgeable
 - WaitPurgeability [204](#)
- notReadable
 - Access [123](#)
- notUpdatable
 - Access [123](#)
- num (パラメーター)
 - operator« [84](#), [85](#), [215](#), [216](#)
 - operator!= [153](#)
 - operator= [152](#), [166](#)
 - operator== [153](#)
- number (パラメーター)
 - コンストラクター [143](#)
 - setCustomClassNum [75](#)
- numberOfItems
 - IccTempStore クラス内 [205](#)
- numEvents (パラメーター)
 - waitExternal [200](#)
- numLines (パラメーター)
 - setNewLine [222](#)
- numRoutes (パラメーター)
 - setRouteCodes [95](#)

O

- object (パラメーター)
 - コンストラクター [107](#), [109](#), [110](#)
 - operator delete [74](#)
- offset (パラメーター)
 - cut [79](#), [80](#)
 - dataArea [80](#)
 - insert [81](#)
 - replace [85](#)
 - setCursor [221](#)
- onOff (パラメーター)
 - setEDF [60](#), [161](#)
- openStatus
 - IccFile クラス内 [118](#)
- operatingSystem

- operatingSystem (続き)
 - IccSystem クラス内 [191](#)
 - public メソッド [191](#)
- operatingSystemLevel
 - IccSystem クラス内 [191](#)
- operator«
 - IccBuf クラスにおける [83](#), [84](#)
 - IccResource サブクラスの処理における [15](#)
 - IccTerminal クラス内 [214](#)–[216](#)
- operator const char*
 - IccBuf クラスにおける [81](#)
- operator delete
 - IccBase クラス内 [74](#)
 - public メソッド [74](#)
- operator new
 - IccBase クラス内 [75](#)
- operator!=
 - IccBuf クラスにおける [82](#)
 - IccKey クラス内 [140](#)
 - IccRBA クラス内 [153](#)
 - IccRRN クラス内 [166](#)
 - public メソッド [82](#)
- operator+=
 - IccBuf クラスにおける [82](#)
- operator==
 - IccBuf クラスにおける [82](#)
 - IccKey クラス内 [139](#), [140](#)
 - IccRBA クラス内 [152](#), [153](#)
 - IccRRN クラス内 [166](#)
- opt (パラメーター)
 - abendCode [62](#)
 - access [113](#)
 - accessMethod [114](#)
 - alternateHeight [227](#)
 - alternateWidth [227](#)
 - ASRAInterrupt [63](#)
 - ASRAKeyType [63](#)
 - ASRAPSW [63](#)
 - ASRARegisters [64](#)
 - ASRASpaceType [64](#)
 - ASRAStorageType [65](#)
 - className [74](#)
 - defaultHeight [227](#)
 - defaultWidth [228](#)
 - enableStatus [115](#)
 - enterTrace [196](#)
 - graphicCharCodeSet [228](#)
 - graphicCharSetId [228](#)
 - height [213](#)
 - isAddable [115](#)
 - isAPLKeyboard [228](#)
 - isAPLText [229](#)
 - isBrowsable [116](#)
 - isBTrans [229](#)
 - isColor [229](#)
 - isDeletable [116](#)
 - isDumpAvailable [65](#)
 - isEmptyOnOpen [116](#)
 - isEWA [229](#)
 - isExtended3270 [230](#)
 - isFieldOutline [230](#)
 - isGoodMorning [230](#)
 - isHighlight [230](#)
 - isKatakana [230](#)

opt (パラメーター) (続き)

- isMSRControl [231](#)
- isPS [231](#)
- isReadable [116](#)
- isRecoverable [117](#)
- isSOSI [231](#)
- isTextKeyboard [231](#)
- isTextPrint [232](#)
- isUpdatable [117](#)
- isValidation [232](#)
- keyLength [117](#)
- keyPosition [118](#)
- link [148](#)
- load [148](#)
- openStatus [118](#)
- originalAbendCode [65](#)
- principalSysId [198](#)
- priority [198](#)
- programName [65](#)
- recordFormat [119](#)
- recordLength [119](#)
- rewriteItem [206](#)
- setNextTransId [223](#)
- type [121](#)
- userId [200](#)
- waitExternal [200](#)
- width [224](#)
- write [95](#)
- writeAndGetReply [95](#)
- writeItem [207](#)

opt1 (パラメーター)

- abend [194](#)

opt2 (パラメーター)

- abend [194](#)

option (パラメーター)

- allocate [171](#)
- retrieveData [182](#)
- send [176](#)
- sendInvite [177](#)
- sendLast [177, 178](#)
- state [178](#)
- stateText [178](#)
- wait [132](#)
- writeRecord [133](#)

options (パラメーター)

- コンストラクター [130, 131](#)

opts (パラメーター)

- setDumpOpts [199](#)

originalAbendCode

- IccAbendData クラス内 [65](#)

overlay

- IccBuf クラスにおける [85](#)

P

PA1 から PA3

- AIDVal [225](#)

packedDecimal

- IccAbsTime クラス内 [69](#)

partnerName (パラメーター)

- コンストラクター [145](#)

- operator= [145](#)

password (パラメーター)

- changePassword [243](#)

password (パラメーター) (続き)

- signon [223, 224](#)

- verifyPassword [245](#)

passwordExpiration

- IccUser クラス内 [244](#)

PF1 から PF24

- AIDVal [225](#)

pink

- Color [226](#)

PIP (パラメーター)

- connectProcess [171, 172](#)

PIPList

- IccSession クラス内 [175](#)

platformError

- Type [112](#)

popt (パラメーター)

- setStartOpts [184](#)

prefix (パラメーター)

- registerPrefix [131](#)

- setPrefix [132](#)

pri (パラメーター)

- setPriority [199](#)

principalSysId

- IccTask クラス内 [198](#)

- public メソッド [198](#)

priority

- IccTask クラス内 [198](#)

- public メソッド [198](#)

process

- IccSession クラス内 [175](#)

profile (パラメーター)

- コンストラクター [170, 171](#)

progName (パラメーター)

- コンストラクター [147, 150](#)

- operator= [151](#)

programId (パラメーター)

- setAbendHandler [99](#)

programName

- IccAbendData クラス内 [65](#)

- public メソッド [65](#)

programName (パラメーター)

- setAbendHandler [100](#)

protected メソッド

- IccBase クラス内 [75](#)

- IccResourceId クラス内 [164](#)

- operator= [164](#)

- setClassName [75](#)

- setCustomClassNum [75](#)

ProtectOpt

- 列挙型 [186](#)

- IccStartRequestQ クラス内 [186](#)

pStorage (パラメーター)

- freeStorage [189](#)

public メソッド

- 異常終了 [194](#)

- 更新 [89](#)

- コンソール [98](#)

- システム [100](#)

- 受信 [176, 217](#)

- 状態 [178](#)

- 送信 [176, 217, 218](#)

- タスク [100](#)

- データ [181, 212](#)

- abendCode [62](#)

public メソッド (続き)

[abendData 194](#)
[absTime 87](#)
[access 113](#)
[accessMethod 114](#)
[actionOnCondition 157](#)
[actionOnConditionAsChar 158](#)
[actionsOnConditionsText 158](#)
[address 147](#)
[AID 212](#)
[allocate 171](#)
[alternateHeight 227](#)
[alternateWidth 227](#)
[append 78](#)
[applName 188](#)
[ASRAInterrupt 62](#)
[ASRAKeyType 63](#)
[ASRAPSW 63](#)
[ASRARegisters 63](#)
[ASRASpaceType 64](#)
[ASRAStorageType 65](#)
[assign 79, 138](#)
[beginBrowse 188, 189](#)
[beginInsert \(VSAM のみ\) 114](#)
[callingProgramId 97](#)
[cancel 181](#)
[cancelAbendHandler 97](#)
[cancelAlarm 87](#)
[changePassword 243](#)
[className 74, 108, 110, 143](#)
[classType 74, 108, 110](#)
[clear 103, 158, 205, 212](#)
[clearData 181](#)
[clearInputMessage 147](#)
[clearPrefix 131](#)
[commArea 97](#)
[commitUOW 194](#)
[completeLength 139](#)
[condition 108, 159](#)
[conditionText 108, 159](#)
[connectProcess 171, 172](#)
[converse 172](#)
[convId 173](#)
[cursor 212](#)
[customClassNum 74](#)
[cut 79](#)
[dataArea 80](#)
[dataAreaLength 80](#)
[dataAreaOwner 80](#)
[dataAreaType 80](#)
[dataLength 80](#)
[date 67, 88](#)
[dateFormat 189](#)
[dayOfMonth 67, 88](#)
[dayOfWeek 68, 88](#)
[daysSince1900 68, 88](#)
[daysUntilPasswordExpires 243](#)
[defaultHeight 227](#)
[defaultWidth 228](#)
[delay 195](#)
[deleteLockedRecord 114](#)
[deleteRecord 114](#)
[dump 195](#)
[empty 103, 205](#)

public メソッド (続き)

[enableStatus 115](#)
[endBrowse 189](#)
[endInsert \(VSAM のみ\) 115](#)
[enterTrace 195](#)
[entryPoint 147](#)
[erase 212](#)
[errorCode 173](#)
[ESMReason 244](#)
[ESMResponse 244](#)
[extractProcess 173](#)
[facilityType 196](#)
[flush 173](#)
[free 173](#)
[freeKeyboard 213](#)
[freeStorage 189, 196](#)
[get 104, 159, 174, 205, 213](#)
[getFile 189, 190](#)
[getNextFile 190](#)
[getStorage 190, 196](#)
[graphicCharCodeSet 228](#)
[graphicCharSetId 228](#)
[groupId 244](#)
[handleEvent 159](#)
[height 213](#)
[hours 68, 233](#)
[IccAbendData クラス内 62](#)
[IccAbsTime クラス内 67](#)
[IccAlarmRequestId クラス内 72](#)
[IccBase クラス内 74](#)
[IccBuf クラスにおける 78](#)
[IccClock クラス内 87](#)
[IccConsole クラス内 93](#)
[IccControl クラス内 97](#)
[IccConvId クラス内 102](#)
[IccDataQueue クラス内 103](#)
[IccDataQueueId クラス内 106](#)
[IccEvent クラス内 108](#)
[IccException クラス内 110](#)
[IccFile クラス内 113](#)
[IccFileId クラス内 125](#)
[IccFileIterator クラス内 126](#)
[IccGroupId クラス内 129](#)
[IccJournal クラス内 131](#)
[IccJournalId クラス内 135](#)
[IccJournalTypeId クラス内 137](#)
[IccKey クラス内 138](#)
[IccLockId クラス内 142](#)
[IccMessage クラス内 143](#)
[IccPartnerId クラス内 145](#)
[IccProgram クラス内 147](#)
[IccProgramId クラス内 151](#)
[IccRBA クラス内 152](#)
[IccRecordIndex クラス内 154](#)
[IccRequestId クラス内 156](#)
[IccResource クラス内 157](#)
[IccResourceId クラス内 164](#)
[IccRRN クラス内 165](#)
[IccSemaphore クラス内 168](#)
[IccSession クラス内 171](#)
[IccStartRequestQ クラス内 181](#)
[IccSysId クラス内 187](#)
[IccSystem クラス内 188](#)
[IccTask クラス内 194](#)

public メソッド (続き)

IccTempStore クラス内 [205](#)
IccTempStoreId クラス内 [209](#)
IccTermId クラス内 [210](#)
IccTerminal クラス内 [212](#)
IccTerminalData クラス内 [227](#)
IccTime クラス内 [233](#)
IccTimeInterval クラス内 [236](#)
IccTimeOfDay クラス内 [238](#)
IccTPNameId クラス内 [240](#)
IccTransId クラス内 [241](#)
IccUser クラス内 [243](#)
IccUserId クラス内 [246](#)
id [159](#)
initData [98](#)
inputCursor [213](#)
insert [80](#)
instance [65](#), [93](#), [98](#), [181](#), [191](#),
[197](#), [213](#)
invalidPasswordAttempts [244](#)
isAddable [115](#)
isAPLKeyboard [228](#)
isAPLText [229](#)
isBrowsable [115](#)
isBTrans [229](#)
isColor [229](#)
isCommandSecurityOn [197](#)
isCommitSupported [197](#)
isCreated [98](#)
isDeletable [116](#)
isDumpAvailable [65](#)
isEDFOn [159](#)
isEmptyOnOpen [116](#)
isErrorSet [174](#)
isEWA [229](#)
isExpired [72](#)
isExtended3270 [229](#)
isFieldOutline [230](#)
isFMHContained [81](#)
isGoodMorning [230](#)
isHighlight [230](#)
isKatakana [230](#)
isMSRControl [231](#)
isNoDataSet [174](#)
isPS [231](#)
isReadable [116](#)
isRecoverable [117](#)
isResourceSecurityOn [197](#)
isRestarted [198](#)
isRouteOptionOn [160](#)
isSignalSet [174](#)
isSOSI [231](#)
isStartDataAvailable [198](#)
issueAbend [174](#)
issueConfirmation [174](#)
issueError [175](#)
issuePrepare [175](#)
issueSignal [175](#)
isTextKeyboard [231](#)
isTextPrint [231](#)
isUpdatable [117](#)
isValidation [232](#)
journalTypeId [131](#)
keyLength [117](#)

public メソッド (続き)

keyPosition [117](#)
kind [139](#)
language [244](#)
lastPasswordChange [244](#)
lastUseTime [244](#)
length [147](#), [154](#)
lifeTime [168](#)
line [213](#)
link [148](#)
load [148](#)
lock [168](#)
message [110](#)
methodName [108](#), [110](#), [144](#)
milliseconds [68](#), [88](#)
minutes [68](#), [233](#)
monthOfYear [68](#), [89](#)
name [160](#), [164](#)
nameLength [164](#)
netName [213](#)
number [111](#), [135](#), [144](#), [153](#),
[166](#), [198](#)
numberOfItems [205](#)
openStatus [118](#)
operatingSystem [191](#)
operatingSystemLevel [191](#)
operator« [83](#), [84](#), [214–216](#)
operator const char* [81](#)
operator delete [74](#)
operator new [75](#)
operator!= [82](#), [140](#), [153](#), [166](#)
operator+= [82](#)
operator= [69](#), [72](#), [81](#), [102](#), [106](#),
[125](#), [129](#), [135](#), [137](#), [139](#), [142](#),
[145](#), [151](#), [152](#), [156](#), [165](#), [187](#),
[209–211](#), [236](#), [238](#), [240–242](#),
[246](#), [247](#)
operator== [82](#), [139](#), [140](#), [152](#),
[153](#), [166](#)
originalAbendCode [65](#)
overlay [85](#)
packedDecimal [69](#)
passwordExpiration [244](#)
PIPList [175](#)
principalSysId [198](#)
priority [198](#)
process [175](#)
programId [98](#)
programName [65](#)
put [94](#), [104](#), [131](#), [160](#), [175](#),
[205](#), [216](#)
queueName [182](#)
readItem [104](#), [206](#)
readNextItem [206](#)
readNextRecord [126](#)
readPreviousRecord [127](#)
readRecord [118](#)
receive3270Data [217](#)
recordFormat [119](#)
recordIndex [119](#)
recordLength [119](#)
registerData [182](#)
registerInputMessage [148](#)
registerPrefix [131](#)

public メソッド (続き)

[registerRecordIndex](#) [119](#)
[release](#) [191](#)
[releaseText](#) [192](#)
[replace](#) [85](#)
[replyTimeout](#) [94](#)
[reset](#) [127](#), [182](#)
[resetAbendHandler](#) [99](#)
[resetRouteCodes](#) [94](#)
[retrieveData](#) [182](#)
[returnProgramId](#) [99](#)
[returnTermId](#) [183](#)
[returnTransId](#) [183](#)
[rewriteItem](#) [206](#)
[rewriteRecord](#) [120](#)
[rollBackUOW](#) [199](#)
[routeOption](#) [160](#)
[run](#) [99](#)
[seconds](#) [69](#), [233](#)
[send3270Data](#) [218](#), [219](#)
[sendInvite](#) [176](#), [177](#)
[sendLast](#) [177](#)
[sendLine](#) [219](#), [220](#)
[session](#) [99](#)
[set](#) [236](#), [238](#)
[setAbendHandler](#) [99](#)
[setAccess](#) [120](#)
[setActionOnAnyCondition](#) [160](#)
[setActionOnCondition](#) [161](#)
[setActionsOnConditions](#) [161](#)
[setAlarm](#) [89](#)
[setAllRouteCodes](#) [94](#)
[setColor](#) [221](#)
[setCursor](#) [221](#)
[setData](#) [183](#)
[setDataLength](#) [85](#)
[setDumpOpts](#) [199](#)
[setEDF](#) [161](#)
[setEmptyOnOpen](#) [120](#)
[setFMHContained](#) [86](#)
[setHighlight](#) [221](#)
[setInputMessage](#) [149](#)
[setJournalTypeId](#) [132](#)
[setKind](#) [140](#)
[setLanguage](#) [245](#)
[setLine](#) [222](#)
[setNewLine](#) [222](#)
[setNextCommArea](#) [222](#)
[setNextInputMessage](#) [222](#)
[setNextTransId](#) [223](#)
[setPrefix](#) [132](#)
[setPriority](#) [199](#)
[setQueueName](#) [183](#)
[setReplyTimeout](#) [94](#)
[setReturnTermId](#) [183](#)
[setReturnTransId](#) [184](#)
[setRouteCodes](#) [95](#)
[setRouteOption](#) [161](#), [162](#)
[setStartOpts](#) [184](#)
[setStatus](#) [121](#)
[setTimerECA](#) [72](#)
[setWaitText](#) [199](#)
[signoff](#) [223](#)
[signon](#) [223](#)

public メソッド (続き)

[start](#) [184](#)
[startRequestQ](#) [100](#)
[startType](#) [199](#)
[stateText](#) [178](#)
[summary](#) [108](#), [111](#), [144](#)
[suspend](#) [200](#)
[syncLevel](#) [179](#)
[sysId](#) [192](#)
[terminal](#) [100](#)
[text](#) [144](#)
[time](#) [69](#), [89](#)
[timeInHours](#) [69](#), [234](#)
[timeInMinutes](#) [69](#), [234](#)
[timeInSeconds](#) [70](#), [234](#)
[timerECA](#) [73](#)
[transId](#) [200](#)
[triggerDataQueueId](#) [200](#)
[tryLock](#) [168](#)
[type](#) [111](#), [121](#), [154](#), [168](#), [234](#)
[typeText](#) [111](#)
[unload](#) [149](#)
[unlock](#) [168](#)
[unlockRecord](#) [121](#)
[userId](#) [200](#)
[value](#) [140](#)
[verifyPassword](#) [245](#)
[wait](#) [132](#)
[waitExternal](#) [200](#)
[waitForAID](#) [224](#)
[waitOnAlarm](#) [201](#)
[width](#) [224](#)
[workArea](#) [192](#), [201](#), [224](#)
[write](#) [95](#)
[writeAndGetReply](#) [95](#)
[writeItem](#) [104](#), [206](#), [207](#)
[writeRecord](#) [122](#), [133](#)
[year](#) [70](#), [90](#)

purgeable

[WaitPurgeability](#) [204](#)

Q

queue

[AllocateOpt](#) [179](#)
[NextTransIdOpt](#) [226](#)
[queueName](#) (パラメーター)
[コンストラクター](#) [103](#), [106](#)
[operator=](#) [106](#)
[setQueueName](#) [183](#)

R

rAbendTask

[HandleEventReturnOpt](#) [163](#)

RBA [15](#)

rba (パラメーター)

[operator!=](#) [153](#)
[operator=](#) [152](#)
[operator==](#) [152](#)

rContinue

[HandleEventReturnOpt](#) [163](#)

readable

readable (続き)
Access [123](#)
ReadMode
 列挙型 [123](#)
 IccFile クラス内 [123](#)
readNextRecord メソッド [17](#)
READONLY
 ASRAStorageType [64](#)
readPreviousRecord
 レコードの参照における [18](#)
 IccFileIterator クラス内 [127](#)
readRecord メソッド [16](#)
receive3270Data
 IccTerminal クラス内 [217](#)
 public メソッド [217](#)
record (パラメーター)
 writeRecord [133](#)
recordFormat メソッド [16](#)
recordIndex メソッド [16](#)
recordLength メソッド [16](#)
red
 Color [226](#)
registerInputMessage
 IccTerminal クラス内 [148](#)
registerPrefix
 IccJournal クラス内 [131](#)
 public メソッド [131](#)
registerRecordIndex メソッド [16](#)
release
 IccSystem クラス内 [191](#)
releaseAtTaskEnd
 LoadOpt [150](#)
releaseText
 IccSystem クラス内 [192](#)
replyTimeout
 IccConsole クラス内 [94](#)
reqName (パラメーター)
 operator= [156](#)
reqId (パラメーター)
 cancel [181](#)
 cancelAlarm [87](#)
 delay [195](#)
 setAlarm [89](#)
 start [185](#)
requestName (パラメーター)
 コンストラクター [156](#)
 operator= [72](#), [156](#)
requestNum (パラメーター)
 wait [132](#)
resetAbendHandler
 IccControl クラス内 [99](#)
resetRouteCodes
 IccConsole クラス内 [94](#)
 public メソッド [94](#)
resId (パラメーター)
 beginBrowse [188](#), [189](#)
resName (パラメーター)
 コンストラクター [164](#)
 beginBrowse [189](#)
resource (パラメーター)
 コンストラクター [167](#)
 beginBrowse [188](#), [189](#)
 endBrowse [189](#)
 enterTrace [196](#)

ResourceType
 列挙型 [193](#)
 IccSystem クラス内 [193](#)
respectAbendHandler
 AbendHandlerOpt [202](#)
RetrieveOpt
 列挙型 [186](#)
 IccStartRequestQ クラス内 [186](#)
return
 EXEC CICS 呼び出しとファウンデーション・クラス・メソッドのマッピング [46](#)
returnCondition
 NoSpaceOpt [208](#)
returnProgramId
 IccControl クラス内 [99](#)
 public メソッド [99](#)
returnToCICS
 関数内 [60](#)
 Icc 構造内 [60](#)
reverse
 Highlight [226](#)
rewriteRecord メソッド [17](#)
rollBackUOW
 IccTask クラス内 [199](#)
routeOption
 IccResource クラス内 [160](#)
row (パラメーター)
 send [218](#)
 setCursor [221](#)
RRN [15](#)
rrn (パラメーター)
 operator!= [166](#)
 operator= [165](#)
 operator== [166](#)
rThrowException
 HandleEventReturnOpt [163](#)

S

SDFHLOAD [3](#)
SDFHPROC [3](#)
SDFHSDCK [3](#)
search (パラメーター)
 コンストラクター [126](#)
 reset [127](#)
SearchCriterion
 列挙型 [124](#)
 IccFile クラス内 [124](#)
seconds
 IccAbsTime クラス内 [69](#)
 IccTime クラス内 [233](#)
seconds (パラメーター)
 コンストラクター [233](#), [235](#), [237](#)
 set [236](#), [238](#)
 setReplyTimeout [94](#), [95](#)
send (パラメーター)
 converse [172](#)
 put [94](#)
 send [176](#)
 sendInvite [177](#)
 sendLast [177](#)
 write [95](#)
 writeAndGetReply [95](#)
send3270Data

send3270Data (続き)
 IccTerminal クラス内 [218, 219](#)
 sendInvite
 IccSession クラス内 [176, 177](#)
 sendLast
 IccSession クラス内 [177](#)
 SendOpt
 列挙型 [180](#)
 IccSession クラス内 [180](#)
 session
 FacilityType [203](#)
 IccControl クラス内 [99](#)
 set
 IccTimeInterval クラス内 [236](#)
 IccTimeOfDay クラス内 [238](#)
 set (パラメーター)
 boolText [58](#)
 setAbendHandler
 IccControl クラス内 [99](#)
 setAccess
 IccFile クラス内 [120](#)
 setActionOnAnyCondition
 IccResource クラス内 [160](#)
 setActionOnCondition
 IccResource クラス内 [161](#)
 setActionsOnConditions
 IccResource クラス内 [161](#)
 setAlarm
 IccAlarmRequestId クラス内 [71](#)
 IccClock クラス内 [89](#)
 setAllRouteCodes
 IccConsole クラス内 [94](#)
 setClassName
 IccBase クラス内 [75](#)
 protected メソッド [75](#)
 setCursor
 IccTerminal クラス内 [221](#)
 setCustomClassNum
 IccBase クラス内 [75](#)
 protected メソッド [75](#)
 setDataLength
 IccBuf クラスにおける [85](#)
 setDumpOpts
 IccTask クラス内 [199](#)
 setEDF
 関数内 [60](#)
 Icc 構造内 [60](#)
 IccResource クラス内 [161](#)
 setEmptyOnOpen
 IccFile クラス内 [120](#)
 public メソッド [120](#)
 setFMHContained
 IccBuf クラスにおける [86](#)
 public メソッド [86](#)
 setInputMessage
 IccProgram クラス内 [149](#)
 public メソッド [149](#)
 setJournalTypeId
 IccJournal クラス内 [132](#)
 setLanguage
 IccUser クラス内 [245](#)
 setLine
 IccTerminal クラス内 [222](#)
 setNewLine
 IccTerminal クラス内 [222](#)
 setNextCommArea
 IccTerminal クラス内 [222](#)
 public メソッド [222](#)
 setNextInputMessage
 IccTerminal クラス内 [222](#)
 setNextTransId
 IccTerminal クラス内 [223](#)
 setPrefix
 IccJournal クラス内 [132](#)
 setPriority
 IccTask クラス内 [199](#)
 public メソッド [199](#)
 setReplyTimeout
 IccConsole クラス内 [94](#)
 setRouteCodes
 IccConsole クラス内 [95](#)
 setStartOpts
 IccStartRequestQ クラス内 [184](#)
 setStatus
 IccFile クラス内 [121](#)
 setTimerECA
 IccAlarmRequestId クラス内 [72](#)
 setWaitText
 IccTask クラス内 [199](#)
 SeverityOpt
 列挙型 [96](#)
 IccConsole クラス内 [96](#)
 signoff
 IccTerminal クラス内 [223](#)
 signon
 IccTerminal クラス内 [223](#)
 public メソッド [223](#)
 singleton クラス [12](#)
 size (パラメーター)
 getStorage [190, 196, 197](#)
 operator new [75](#)
 startIO
 Options [134](#)
 startRequest
 StartType [203](#)
 StartType
 列挙型 [203](#)
 IccTask クラス内 [203](#)
 StateOpt
 列挙型 [180](#)
 IccSession クラス内 [180](#)
 stateText
 IccSession クラス内 [178](#)
 status (パラメーター)
 setStatus [121](#)
 StorageOpts
 列挙型 [203](#)
 IccTask クラス内 [203](#)
 storageOpts (パラメーター)
 getStorage [190, 196, 197](#)
 storeName (パラメーター)
 コンストラクター [205](#)
 SUBSPACE
 ASRASpaceType [64](#)
 summary
 IccEvent クラス内 [108](#)
 IccException クラス内 [111](#)

summary (続き)
 IccMessage クラス内 [144](#)
suppressDump
 AbendDumpOpt [202](#)
suspend
 IccTask クラス内 [200](#)
 NoSpaceOpt [208](#)
syncLevel
 IccSession クラス内 [179](#)
SyncLevel
 [列挙型 180](#)
 IccSession クラス内 [180](#)
sysId
 IccSystem クラス内 [192](#)
sysId (パラメーター)
 コンストラクター [170](#)
 setRouteOption [161](#)
sysName (パラメーター)
 コンストラクター [170](#), [171](#)
 setRouteOption [162](#)

T

termId (パラメーター)
 setReturnTermId [183](#)
 start [185](#)
terminal
 FacilityType [203](#)
 IccControl クラス内 [100](#)
terminalInput
 StartType [203](#)
termName (パラメーター)
 setReturnTermId [184](#)
test (パラメーター)
 boolText [58](#)
text
 IccMessage クラス内 [144](#)
text (パラメーター)
 コンストラクター [78](#), [143](#)
 operator« [83](#), [215](#)
 operator!= [140](#)
 operator+= [82](#)
 operator= [81](#), [82](#)
 operator== [139](#), [140](#)
 writeItem [104](#), [105](#), [207](#)
time
 IccAbsTime クラス内 [69](#)
 IccClock クラス内 [89](#)
time (パラメーター)
 コンストラクター [67](#), [236](#), [238](#)
 delay [195](#)
 setAlarm [89](#)
 start [185](#)
timeInHours
 IccAbsTime クラス内 [69](#)
 IccTime クラス内 [234](#)
timeInMinutes
 IccAbsTime クラス内 [69](#)
 IccTime クラス内 [234](#)
timeInSeconds
 IccAbsTime クラス内 [70](#)
 IccTime クラス内 [234](#)
timeInterval
 Type [235](#)

timeInterval (パラメーター)
 operator= [236](#)
timeOfDay
 Type [235](#)
timeOfDay (パラメーター)
 operator= [238](#)
timerECA
 IccAlarmRequestId クラス内 [73](#)
timerECA (パラメーター)
 コンストラクター [71](#)
 setTimerECA [72](#)
timeSeparator (パラメーター)
 time [69](#), [89](#)
TPName (パラメーター)
 connectProcess [172](#)
traceNum (パラメーター)
 enterTrace [196](#)
TraceOpt
 [列挙型 204](#)
 IccTask クラス内 [204](#)
transId
 IccTask クラス内 [200](#)
transid (パラメーター)
 setNextTransId [223](#)
transId (パラメーター)
 cancel [181](#)
 connectProcess [172](#)
 link [148](#)
 setNextTransId [223](#)
 setReturnTransId [184](#)
 start [185](#)
transName (パラメーター)
 setReturnTransId [184](#)
triggerDataQueueId
 IccTask クラス内 [200](#)
trueFalse (パラメーター)
 setEmptyOnOpen [120](#)
tryLock
 IccSemaphore クラス内 [168](#)
type (パラメーター)
 コンストラクター [73](#), [77](#), [78](#), [154](#), [163](#), [164](#), [167](#)
 条件の [108](#), [159](#)
 waitExternal [200](#)
typeText
 IccException クラス内 [111](#)

U

underscore
 Highlight [226](#)
unknownException
 関数内 [60](#)
 Icc 構造内 [60](#)
unload
 IccProgram クラス内 [149](#)
unlock
 IccSemaphore クラス内 [168](#)
unlockRecord
 IccFile クラス内 [121](#)
UOW
 LifeTime [170](#)
updatable
 Access [123](#)
update (パラメーター)

update (パラメーター) (続き)
コンストラクター [87](#)

UpdateMode

列挙型 [91](#)

IccClock クラス内 [91](#)

updateToken (パラメーター)

deleteLockedRecord [114](#)

readNextRecord [126](#), [127](#)

readPreviousRecord [127](#)

readRecord [118](#)

rewriteRecord [120](#)

unlockRecord [122](#)

upper

Case [225](#)

USER

ASRAStorageType [64](#)

user (パラメーター)

signon [223](#), [224](#)

userDataKey

StorageOpts [203](#)

USEREXECKEY

ASRAKeyType [63](#)

userId

IccTask クラス内 [200](#)

userId (パラメーター)

start [185](#)

userName (パラメーター)

コンストラクター [243](#)

V

value

IccKey クラス内 [140](#)

value (パラメーター)

operator= [139](#)

variable (パラメーター)

ファウンデーション・クラスの参照情報内 [45](#)

verifyPassword

IccUser クラス内 [245](#)

public メソッド [245](#)

VSAM [15](#)

W

wait

IccJournal クラス内 [132](#)

SendOpt [180](#)

waitExternal

ECBList (パラメーター)

waitExternal [200](#)

IccTask クラス内 [200](#)

numEvents (パラメーター)

waitExternal [200](#)

opt (パラメーター)

waitExternal [200](#)

type (パラメーター)

waitExternal [200](#)

waitOnAlarm

IccAlarmRequestId クラス内 [71](#)

IccTask クラス内 [201](#)

WaitPostType

列挙型 [204](#)

IccTask クラス内 [204](#)

WaitPurgeability

列挙型 [204](#)

IccTask クラス内 [204](#)

width

IccTerminal クラス内 [224](#)

workArea

IccSystem クラス内 [192](#)

IccTask クラス内 [201](#)

IccTerminal クラス内 [224](#)

write

IccConsole クラス内 [95](#)

writeAndGetReply

IccConsole クラス内 [95](#)

X

X

actionOnConditionAsChar [158](#)

operatingSystem [191](#)

XPLINK [3](#)

Y

year

IccAbsTime クラス内 [70](#)

IccClock クラス内 [90](#)

yellow

Color [226](#)

yesNo (パラメーター)

setFMHContained [86](#)

[特殊文字]

MVS/ESA

ストレージ管理における [43](#)

ClassMemoryMgmt [61](#)

UNIX

ストレージ管理における [43](#)

ClassMemoryMgmt [61](#)

abendTask

ActionOnCondition [163](#)

CICS 条件における [35](#)

buffer

トランザクション開始の例における [22](#), [23](#)

callHandleEvent

ActionOnCondition [163](#)

CICS 条件における [35](#)

CICSCondition

C++ 例外およびファウンデーション・クラスにおける [35](#)

Type [112](#)

cmmCICS

ストレージ管理における [43](#)

ClassMemoryMgmt [61](#)

cmmDefault

ストレージ管理における [43](#)

ClassMemoryMgmt [61](#)

cmmNonCICS

ストレージ管理における [43](#)

ClassMemoryMgmt [61](#)

ESDS

ファイル制御における [15](#)

familyConformanceError

familyConformanceError (続き)
 C++ 例外およびファウンデーション・クラスにおける [35](#)
 Type [112](#)

fsAllowPlatformVariance
 プラットフォームの相違における [37](#)
 FamilySubset [61](#)

fsEnforce
 プラットフォームの相違における [37](#)
 FamilySubset [61](#)

IccAbsTime
 基本クラスにおける [7](#)
 サポート・クラスにおける [10](#)
 日時サービスにおける [29](#)
 delay [195](#)
 IccTime クラス内 [233](#)

IccCondition
 C++ 例外およびファウンデーション・クラスにおける [35](#)

IccControl
 基本クラスにおける [7](#)
 サポート・クラスにおける [11](#)
 トランザクション開始の例における [22, 23](#)
 メソッド・レベルにおける [38](#)
 EXEC CICS 呼び出しとファウンデーション・クラス・メソッドのマッピング [46](#)
 IccControl クラス内 [97](#)
 IccProgram クラス内 [146](#)
 main 関数における [248](#)
 singleton クラスにおける [12](#)

IccControl クラス
 概要 [7, 12](#)
 コンストラクター [97](#)
 コンソール [98](#)
 システム [100](#)
 タスク [100](#)
 callingProgramId [97](#)
 cancelAbendHandler [97](#)
 commArea [97](#)
 initData [98](#)
 instance [98](#)
 isCreated [98](#)
 programId [98](#)
 resetAbendHandler [99](#)
 returnProgramId [99](#)
 run [99](#)
 session [99](#)
 setAbendHandler [99](#)
 startRequestQ [100](#)
 terminal [100](#)

IccException クラス
 コンストラクター [109](#)
 タイプ [112](#)
 CICSCondition タイプ [35](#)
 className [110](#)
 classType [110](#)
 familyConformanceError タイプ [35](#)
 internalError タイプ [35](#)
 invalidArgument タイプ [34](#)
 invalidMethodCall タイプ [35](#)
 message [110](#)
 methodName [110](#)
 number [111](#)
 objectCreationError タイプ [34](#)

IccException クラス (続き)
 summary [111](#)
 type [111](#)
 typeText [111](#)

IccRecordIndex
 C++ 例外およびファウンデーション・クラスにおける [35](#)
 IccRecordIndex クラス内 [154](#)

IccRequestId
 トランザクション開始の例における [22, 23](#)
 パラメーター受け渡し規則内 [44](#)
 IccRequestId クラス内 [155](#)

IccTempStoreId
 一時記憶域における [25](#)
 一時記憶域の例における [26, 27](#)
 基本クラスにおける [7](#)
 IccTempStoreId クラス内 [208](#)

IccTime
 基本クラスにおける [7](#)
 サポート・クラスにおける [10](#)
 パラメーター受け渡し規則内 [44](#)
 IccTime クラス内 [233](#)

IccTime クラス
 概要 [7](#)
 コンストラクター [233](#)
 タイプ [235](#)
 hours [233](#)
 minutes [233](#)
 seconds [233](#)
 timeInHours [234](#)
 timeInMinutes [234](#)
 timeInSeconds [234](#)
 type [234](#)

IccTimeInterval
 基本クラスにおける [7](#)
 サポート・クラスにおける [10](#)
 トランザクション開始の例における [22, 23](#)
 delay [195](#)
 IccTime クラス内 [233](#)

IccTimeOfDay
 基本クラスにおける [7](#)
 サポート・クラスにおける [10](#)
 delay [195](#)
 IccTime クラス内 [233](#)

internalError
 C++ 例外およびファウンデーション・クラスにおける [35](#)
 Type [112](#)

invalidArgument
 C++ 例外およびファウンデーション・クラスにおける [34](#)
 Type [112](#)

invalidMethodCall
 C++ 例外およびファウンデーション・クラスにおける [35](#)
 Type [112](#)

ISR2
 トランザクション開始の例における [22](#)

ITMP
 トランザクション開始の例における [22](#)

KSDS
 ファイル制御における [15](#)

noAction
 ActionOnCondition [162](#)

noAction (続き)
 CICS 条件における [35](#)

objectCreationError
 C++ 例外およびファウンデーション・クラスにおける [34](#)
 Type [112](#)

remoteTermId
 トランザクション開始の例における [22](#)

req
 トランザクション開始の例における [23](#)

req1
 トランザクション開始の例における [22](#)

req2
 トランザクション開始の例における [22](#)

RRDS ファイル
 ファイル制御における [15](#)

run
 一時記憶域の例における [26, 27](#)
 一時データ管理の例における [25](#)
 基本クラスにおける [7](#)
 端末管理の例における [28, 29](#)
 トランザクション開始の例における [22](#)
 日時サービスの例における [29, 30](#)
 ファイル制御の例における [18, 19](#)
 プログラム制御における [20](#)
 ポリモフィック動作の例における [42](#)
 C++ 例外およびファウンデーション・クラスにおける [34](#)
 EXEC CICS 呼び出しとファウンデーション・クラス・メソッドのマッピング [46](#)
 IccControl クラス内 [97, 99](#)
 main 関数における [248](#)

throwException
 ActionOnCondition [163](#)
 CICS 条件における [35](#)

ti
 トランザクション開始の例における [22, 23](#)

empty
 一時記憶域における [25](#)
 一時データにおける [24](#)
 キューの削除における [24](#)
 項目の削除における [26](#)
 IccDataQueue クラス内 [103](#)
 IccTempStore クラス内 [205](#)

IccAbendData
 singleton クラスにおける [12](#)

IccConsole
 オブジェクト・レベルにおける [38](#)
 バッファ・オブジェクトにおける [12](#)
 singleton クラスにおける [12](#)

IccConsole クラス
 概要 [12](#)
 コンストラクター [93](#)
 instance [93](#)
 put [94](#)
 replyTimeout [94](#)
 resetRouteCodes [94](#)
 setAllRouteCodes [94](#)
 setReplyTimeout [94](#)
 setRouteCodes [95](#)
 SeverityOpt [96](#)
 write [95](#)
 writeAndGetReply [95](#)

IccDataQueue (続き)
 一時記憶域における [26](#)
 一時データ管理の例における [24, 25](#)
 一時データにおける [24](#)
 データの書き込みにおける [24](#)
 バッファ・オブジェクトにおける [12](#)
 ポリモフィック動作の例における [42](#)
 リソース・クラスにおける [9](#)
 IccResource サブクラスの処理における [15](#)

IccProgram
 バッファ・オブジェクトにおける [12](#)
 プログラム制御における [19, 20](#)
 リソース・クラスにおける [9](#)
 IccProgram クラス内 [146](#)

IccStartRequestQ
 開始データへのアクセスにおける [21](#)
 トランザクション開始の例における [22, 23](#)
 トランザクションの非同期の開始における [21](#)
 バッファ・オブジェクトにおける [12](#)
 パラメーター受け渡し規則内 [44](#)
 EXEC CICS 呼び出しとファウンデーション・クラス・メソッドのマッピング [46](#)
 IccRequestId クラス内 [155](#)
 IccStartRequestQ クラス内 [180](#)
 singleton クラスにおける [12](#)

IccStartRequestQ クラス
 概要 [12](#)
 コンストラクター [181](#)
 データ [181](#)
 cancel [181](#)
 CheckOpt [186](#)
 clearData [181](#)
 instance [181](#)
 ProtectOpt [186](#)
 queueName [182](#)
 registerData [182](#)
 reset [182](#)
 retrieveData [182](#)
 RetrieveOpt [186](#)
 returnTermId [183](#)
 returnTransId [183](#)
 setData [183](#)
 setQueueName [183](#)
 setReturnTermId [183](#)
 setReturnTransId [184](#)
 setStartOpts [184](#)
 start [184](#)

IccSystem
 singleton クラスにおける [12](#)

IccSystem クラス
 概要 [12](#)
 コンストラクター [188](#)
 applName [188](#)
 beginBrowse [188, 189](#)
 dateFormat [189](#)
 endBrowse [189](#)
 freeStorage [189](#)
 getFile [189, 190](#)
 getNextFile [190](#)
 getStorage [190](#)
 instance [191](#)
 operatingSystem [191](#)
 operatingSystemLevel [191](#)
 release [191](#)

IccSystem クラス (続き)

releaseText [192](#)
ResourceType [193](#)
sysId [192](#)
workArea [192](#)

IccTask クラス

異常終了 [194](#)
概要 [12](#)
コンストラクター [194](#)
abendData [194](#)
AbendDumpOpt [202](#)
AbendHandlerOpt [202](#)
commitUOW [194](#)
delay [195](#)
dump [195](#)
DumpOpts [202](#)
enterTrace [195](#)
FacilityType [203](#)
facilityType [196](#)
freeStorage [196](#)
getStorage [196](#)
instance [197](#)
isCommandSecurityOn [197](#)
isCommitSupported [197](#)
isResourceSecurityOn [197](#)
isRestarted [198](#)
isStartDataAvailable [198](#)
number [198](#)
principalSysId [198](#)
priority [198](#)
rollBackUOW [199](#)
setDumpOpts [199](#)
setPriority [199](#)
setWaitText [199](#)
StartType [203](#)
startType [199](#)
StorageOpts [203](#)
suspend [200](#)
TraceOpt [204](#)
transId [200](#)
triggerDataQueueId [200](#)
userId [200](#)
waitExternal [200](#)
waitOnAlarm [201](#)
WaitPostType [204](#)
WaitPurgeability [204](#)
workArea [201](#)

IccTask::commitUOW

「read」メソッドから返される IccBuf 参照内のデータの有効範囲内 [44](#)

IccTempStore

一時記憶域における [25, 26](#)
一時記憶域の例における [26, 27](#)
一時データにおける [24](#)
項目の書き込みにおける [26](#)
項目の更新における [26](#)
項目の削除における [26](#)
項目の読み取りにおける [26](#)
自動条件処理 (callHandleEvent) における [36](#)
バッファー・オブジェクトにおける [12](#)
ポリモアフィック動作の例における [42](#)
リソース・クラスにおける [9](#)
C++ 例外およびファウンデーション・クラスにおける [34](#)

IccTempStore (続き)

IccResource サブクラスの処理における [14](#)
IccTempStore クラス内 [204](#)

IccUserControl

一時記憶域の例における [26](#)
一時データ管理の例における [25](#)
端末管理の例における [28](#)
トランザクション開始の例における [22](#)
日時サービスの例における [29](#)
ファイル制御の例における [18](#)
プログラム制御における [20](#)
ポリモアフィック動作の例における [42](#)
C++ 例外およびファウンデーション・クラスにおける [34](#)
main 関数における [248](#)
singleton クラスにおける [12](#)

Id

リソース識別クラスにおける [7](#)

main

一時記憶域の例における [26](#)
一時データ管理の例における [24](#)
ストレージ管理における [43](#)
端末管理の例における [28](#)
トランザクション開始の例における [22](#)
日時サービスの例における [29](#)
ファイル制御の例における [18](#)
プログラム制御における [20](#)
ヘッダー・ファイル内 [2, 32](#)
ポリモアフィック動作の例における [42](#)
C++ 例外およびファウンデーション・クラスにおける [33](#)

queueName

開始データへのアクセスにおける [21](#)
IccStartRequestQ クラス内 [182](#)

readNextItem

「read」メソッドから返される IccBuf 参照内のデータの有効範囲内 [44](#)
一時記憶域における [25](#)
IccTempStore クラス内 [206](#)

registerData

トランザクション開始の例における [22](#)
トランザクションの開始における [21](#)
IccStartRequestQ クラス内 [182](#)

returnTermId

開始データへのアクセスにおける [21](#)
IccStartRequestQ クラス内 [183](#)

returnTransId

開始データへのアクセスにおける [21](#)
IccStartRequestQ クラス内 [183](#)

rewriteItem

一時記憶域における [25](#)
一時記憶域の例における [27](#)
項目の書き込みにおける [26](#)
項目の更新における [26](#)
IccTempStore クラス内 [206](#)

setData

トランザクションの開始における [21](#)
IccStartRequestQ クラス内 [183](#)

setQueueName

トランザクション開始の例における [22](#)
トランザクションの開始における [21](#)
IccStartRequestQ クラス内 [183](#)

setReturnTermId

トランザクション開始の例における [22](#)

setReturnTermId (続き)
 トランザクションの開始における [21](#)
 IccStartRequestQ クラス内 [183](#)
 setReturnTransId
 トランザクション開始の例における [22](#)
 トランザクションの開始における [21](#)
 IccStartRequestQ クラス内 [184](#)
 データ
 開始データへのアクセスにおける [21](#)
 端末に関する情報の検索における [28](#)
 IccStartRequestQ クラス内 [181](#)
 IccTerminal クラス内 [212](#)
 abend
 パラメーター・レベルにおける [38](#)
 AIX、CICS
 プラットフォームの相違における [37](#)
 assign
 ファイル制御の例における [18](#)
 IccBuf クラスにおける [79](#)
 IccKey クラス内 [138](#)
 beginInsert
 レコードの書き込みにおける [16](#)
 C++ 例外およびファウンデーション・クラス
 条件、エラー、および例外における [33](#)
 catch
 例外処理 (throwException) における [37](#)
 C++ 例外およびファウンデーション・クラスにおける [33](#),
 [34](#)
 main 関数における [248](#)
 char*
 C++ 例外およびファウンデーション・クラスにおける
 [34](#)
 CICS
 プラットフォームの相違における [37](#)
 ASRAStorageType [64](#)
 GetOpt [61](#)
 CICS (AIX 版)
 プラットフォームの相違における [37](#)
 CICS Service の使用
 一時記憶域の例 [26](#)
 一時データ管理の例 [24](#)
 開始データへのアクセス [21](#)
 キューの削除 [24](#)
 項目の書き込み [26](#)
 項目の更新 [26](#)
 項目の削除 [26](#)
 項目の読み取り [26](#)
 端末からのデータの受信 [27](#)
 端末管理の例 [28](#)
 端末に関する情報の検索 [28](#)
 端末へのデータの送信 [27](#)
 データの書き込み [24](#)
 データの読み取り [24](#)
 トランザクション開始の例 [21](#)
 トランザクションの開始 [21](#)
 日時サービスの例 [29](#)
 ファイル制御の例 [18](#)
 満了していない開始要求のキャンセル [21](#)
 レコードの書き込み [16](#)
 レコードの更新 [17](#)
 レコードの削除 [17](#)
 レコードのブラウズ [17](#)
 レコードの読み取り [15](#)
 CICS 条件
 CICS 条件 (続き)
 自動条件処理 [36](#)
 自動条件処理 (callHandleEvent) [36](#)
 重大エラー処理 [37](#)
 重大エラー処理 (abendTask) [37](#)
 手動条件処理 [36](#)
 手動条件処理 (noAction) [36](#)
 条件、エラー、および例外における [35](#)
 例外処理 [36](#)
 例外処理 (throwException) [36](#)
 abendTask [37](#)
 callHandleEvent [36](#)
 noAction [36](#)
 throwException [36](#)
 CICS リソースの使用
 ファウンデーション・クラスの概要における [11](#)
 リソース・オブジェクトでのメソッドの呼び出し [12](#)
 リソース・オブジェクトの作成 [11](#)
 singleton クラス [12](#)
 cursor
 端末に関する情報の検索における [28](#)
 IccTerminal クラス内 [212](#)
 cut
 IccBuf クラスにおける [79](#)
 IccBuf コンストラクターにおける [14](#)
 dataAreaOwner
 データ域の所有権における [13](#)
 IccBuf クラスにおける [80](#)
 dataAreaType
 データ域の拡張性における [13](#)
 IccBuf クラスにおける [80](#)
 dataItems
 ポリモフィック動作の例における [42](#)
 dateSeparator (パラメーター)
 日時サービスの例における [29](#)
 date [67](#), [88](#)
 dayOfMonth
 日時サービスの例における [30](#)
 IccAbsTime クラス内 [67](#)
 IccClock クラス内 [88](#)
 dayOfWeek
 日時サービスの例における [30](#)
 IccAbsTime クラス内 [68](#)
 IccClock クラス内 [88](#)
 daysSince1900
 日時サービスの例における [30](#)
 IccAbsTime クラス内 [68](#)
 IccClock クラス内 [88](#)
 delay
 サポート・クラスにおける [10](#)
 IccTask クラス内 [195](#)
 delete
 オブジェクトの削除における [6](#)
 ストレージ管理における [43](#)
 deleteRecord
 通常レコードの削除における [17](#)
 IccFile クラス内 [114](#)
 doSomething
 オブジェクトの使用における [6](#)
 endInsert
 レコードの書き込みにおける [16](#)
 endl
 端末管理の例における [29](#)
 erase

erase (続き)
 端末管理の例における [29](#)
 端末へのデータの送信における [27](#)
 IccTerminal クラス内 [212](#)
ESDS レコードの書き込み
 ファイル制御における [17](#)
 レコードの書き込みにおける [17](#)
ESDS レコードの読み取り
 ファイル制御における [16](#)
 レコードの読み取りにおける [16](#)
file (パラメーター)
 コンストラクター [126](#)
 ファイル制御の例における [18](#)
flush
 端末管理の例における [29](#)
 IccSession クラス内 [173](#)
format (パラメーター)
 日時サービスの例における [29](#)
 append [79](#)
 assign [79](#)
 date [67](#), [88](#)
 send [217](#), [218](#)
 send3270Data [219](#)
 sendLine [220](#)
freeKeyboard
 端末へのデータの送信における [27](#)
 IccTerminal クラス内 [213](#)
handleEvent
 自動条件処理 (callHandleEvent) における [36](#)
 IccResource クラス内 [159](#)
Icc
 ファウンデーション・クラス: 参照 [45](#)
 ファウンデーション・クラスの概要における [6](#)
 ファウンデーション・クラスの参照情報内 [45](#)
 メソッド・レベルにおける [38](#)
Icc::initializeEnvironment
 ストレージ管理における [43](#)
IccBase
 基本クラスにおける [6](#), [7](#)
 サポート・クラスにおける [10](#)
 ストレージ管理における [43](#)
 ファウンデーション・クラスの参照情報内 [45](#)
 リソース・クラスにおける [9](#)
 リソース識別クラスにおける [7](#)
 IccAbendData クラス内 [62](#)
 IccAbsTime クラス内 [66](#)
 IccAlarmRequestId クラス内 [71](#)
 IccBase クラス内 [73](#)
 IccBuf クラスにおける [77](#)
 IccClock クラス内 [87](#)
 IccConsole クラス内 [93](#)
 IccControl クラス内 [97](#)
 IccConvId クラス内 [101](#)
 IccDataQueue クラス内 [103](#)
 IccDataQueueId クラス内 [106](#)
 IccEvent クラス内 [107](#)
 IccException クラス内 [109](#)
 IccFile クラス内 [112](#)
 IccFileId クラス内 [124](#)
 IccFileIterator クラス内 [126](#)
 IccGroupId クラス内 [128](#)
 IccJournal クラス内 [130](#)
 IccJournalId クラス内 [134](#)
 IccJournalTypeId クラス内 [136](#)

IccBase (続き)
 IccKey クラス内 [138](#)
 IccLockId クラス内 [141](#)
 IccMessage クラス内 [143](#)
 IccPartnerId クラス内 [145](#)
 IccProgram クラス内 [146](#)
 IccProgramId クラス内 [150](#)
 IccRBA クラス内 [152](#)
 IccRecordIndex クラス内 [154](#)
 IccRequestId クラス内 [155](#)
 IccResource クラス内 [157](#)
 IccResourceId クラス内 [163](#)
 IccRRN クラス内 [165](#)
 IccSemaphore クラス内 [167](#)
 IccSession クラス内 [170](#)
 IccStartRequestQ クラス内 [180](#)
 IccSysId クラス内 [186](#)
 IccSystem クラス内 [188](#)
 IccTask クラス内 [193](#)
 IccTempStore クラス内 [204](#)
 IccTempStoreId クラス内 [208](#)
 IccTermId クラス内 [210](#)
 IccTerminal クラス内 [211](#)
 IccTerminalData クラス内 [226](#)
 IccTime クラス内 [233](#)
 IccTimeInterval クラス内 [235](#)
 IccTimeOfDay クラス内 [237](#)
 IccTPNameId クラス内 [239](#)
 IccTransId クラス内 [241](#)
 IccUser クラス内 [242](#)
 IccUserId クラス内 [246](#)

IccBase クラス
 概要 [6](#)
 コンストラクター [73](#)
 className [74](#)
 classType [74](#)
 ClassType [75](#)
 customClassNum [74](#)
 NameOpt [76](#)
 operator delete [74](#)
 operator new [75](#)
 setClassName [75](#)
 setCustomClassNum [75](#)

IccBuf
 「read」メソッドから返される IccBuf 参照内のデータの有効範囲内 [44](#)
 一時記憶域の例における [27](#)
 一時データ管理の例における [25](#)
 項目の読み取りにおける [26](#)
 サポート・クラスにおける [11](#)
 端末管理の例における [28](#)
 データ域の拡張性における [13](#)
 データ域の所有権における [13](#)
 データの読み取りにおける [24](#)
 トランザクション開始の例における [22](#), [23](#)
 バッファー・オブジェクトにおける [12](#)
 ファイル制御の例における [18](#)
 ポリモフィック動作の例における [42](#)
 C++ 例外およびファウンデーション・クラスにおける [34](#)
 IccBuf クラスにおける [13](#), [77](#)
 IccBuf コンストラクターにおける [13](#), [14](#)
 IccBuf メソッドにおける [14](#)
 IccResource サブクラスの処理における [14](#), [15](#)

IccBuf クラス

- コンストラクター [13](#), [77](#), [78](#)
- データ域の拡張性 [13](#)
- データ域の所有権 [13](#)
- バッファ・オブジェクトにおける [13](#)
- メソッド [14](#)
- append [78](#)
- assign [79](#)
- cut [79](#)
- dataArea [80](#)
- dataAreaLength [80](#)
- dataAreaOwner [80](#)
- DataAreaOwner [86](#)
- dataAreaType [80](#)
- DataAreaType [86](#)
- dataLength [80](#)
- IccBuf コンストラクター [13](#)
- IccBuf メソッド [14](#)
- IccResource サブクラスの処理 [14](#)
- insert [80](#)
- isFMHContained [81](#)
- operator const char* [81](#)
- operator« [83](#), [84](#)
- operator!= [82](#)
- operator+= [82](#)
- operator= [81](#)
- operator== [82](#)
- overlay [85](#)
- replace [85](#)
- setDataLength [85](#)
- setFMHContained [86](#)

IccBuf コンストラクター

- コンストラクター [77](#), [78](#)
- バッファ・オブジェクトにおける [13](#)
- IccBuf クラスにおける [13](#), [77](#)

IccBuf メソッド

- バッファ・オブジェクトにおける [14](#)
- IccBuf クラスにおける [14](#)

IccClock

- 日時サービスにおける [29](#)
- 日時サービスの例における [29](#), [30](#)
- IccAlarmRequestId クラス内 [71](#)
- IccClock クラス内 [87](#)

IccDataQueueId

- 一時データ管理の例における [24](#)
- 一時データにおける [24](#)
- IccDataQueueId クラス内 [106](#)

IccEvent

- サポート・クラスにおける [11](#)
- IccEvent クラス内 [107](#)

IccException

- オブジェクト・レベルにおける [38](#)
- サポート・クラスにおける [11](#)
- パラメーター・レベルにおける [38](#), [39](#)
- メソッド・レベルにおける [38](#)
- C++ 例外およびファウンデーション・クラスにおける [34](#), [35](#)
- IccException クラス内 [109](#)
- IccMessage クラス内 [143](#)
- main 関数における [249](#)

IccFile

- 通常レコードの削除における [17](#)
- バッファ・オブジェクトにおける [12](#)
- ファイル制御における [15](#)

IccFile (続き)

- ファイル制御の例における [18](#)
- リソース識別クラスにおける [8](#)
- レコードの書き込みにおける [16](#)
- レコードの更新における [17](#)
- レコードの参照における [17](#)
- レコードの読み取りにおける [15](#), [16](#)
- ロックされたレコードの削除における [17](#)
- C++ 例外およびファウンデーション・クラスにおける [35](#)
- ESDS レコードの書き込みにおける [17](#)
- ESDS レコードの読み取りにおける [16](#)
- IccFile クラス内 [112](#)
- IccFileIterator クラス内 [126](#)
- KSDS レコードの書き込みにおける [17](#)
- KSDS レコードの読み取りにおける [16](#)
- RRDS レコードの書き込みにおける [17](#)
- RRDS レコードの読み取りにおける [16](#)
- singleton クラスにおける [12](#)

IccFile クラス

- コンストラクター [113](#)
- 状況 [124](#)
- access [113](#)
- Access [123](#)
- accessMethod [114](#)
- beginInsert (VSAM のみ) [114](#)
- deleteLockedRecord [17](#), [114](#)
- deleteRecord [114](#)
- deleteRecord メソッド [17](#)
- enableStatus [115](#)
- endInsert (VSAM のみ) [115](#)
- isAddable [115](#)
- isBrowsable [115](#)
- isDeletable [116](#)
- isEmptyOnOpen [116](#)
- isReadable [116](#)
- isReadable メソッド [16](#)
- isRecoverable [117](#)
- isUpdatable [117](#)
- keyLength [117](#)
- keyLength メソッド [16](#)
- keyPosition [117](#)
- keyPosition メソッド [16](#)
- openStatus [118](#)
- ReadMode [123](#)
- readRecord [118](#)
- readRecord メソッド [16](#)
- recordFormat [119](#)
- recordFormat メソッド [16](#)
- recordIndex [119](#)
- recordIndex メソッド [16](#)
- recordLength [119](#)
- recordLength メソッド [16](#)
- registerRecordIndex [16](#), [119](#)
- registerRecordIndex メソッド [16](#)
- rewriteRecord [120](#)
- rewriteRecord メソッド [17](#)
- SearchCriterion [124](#)
- setAccess [120](#)
- setEmptyOnOpen [120](#)
- setStatus [121](#)
- type [121](#)
- unlockRecord [121](#)
- writeRecord [122](#)

IccFile クラス (続き)

writeRecord メソッド [16](#)

IccFile::readRecord

「read」メソッドから返される IccBuf 参照内のデータの有効範囲内 [44](#)

IccFileIterator

バッファ・オブジェクトにおける [12](#)

ファイル制御における [15](#)

ファイル制御の例における [18, 19](#)

レコードの参照における [17, 18](#)

IccFileIterator クラス内 [126](#)

IccFileIterator クラス

概要 [15](#)

コンストラクター [126](#)

readNextRecord [126](#)

readNextRecord メソッド [17](#)

readPreviousRecord [17, 127](#)

reset [127](#)

IccJournal

オブジェクト・レベルにおける [38](#)

バッファ・オブジェクトにおける [12](#)

IccJournal クラス内 [130](#)

IccKey

通常レコードの削除における [17](#)

ファイル制御における [15](#)

レコードの書き込みにおける [16](#)

レコードの参照における [17, 18](#)

レコードの読み取りにおける [15](#)

IccKey クラス内 [138](#)

IccRecordIndex クラス内 [154](#)

KSDS レコードの書き込みにおける [16](#)

KSDS レコードの読み取りにおける [16](#)

IccKey クラス

コンストラクター [138](#)

読み取り、レコードの [15](#)

assign [138](#)

completeLength [139](#)

kind [139](#)

Kind [141](#)

operator!= [140](#)

operator= [139](#)

operator== [139, 140](#)

setKind [140](#)

value [140](#)

IccMessage

サポート・クラスにおける [11](#)

IccMessage クラス内 [143](#)

IccProgramId

リソース識別クラスにおける [7](#)

IccProgramId クラス内 [150](#)

IccRBA

ファイル制御における [15](#)

レコードの書き込みにおける [16](#)

レコードの参照における [17, 18](#)

レコードの読み取りにおける [15](#)

ESDS レコードの書き込みにおける [17](#)

ESDS レコードの読み取りにおける [16](#)

IccRBA クラス内 [152](#)

IccRecordIndex クラス内 [154](#)

RRDS レコードの書き込みにおける [17](#)

IccRBA クラス

コンストラクター [152](#)

読み取り、レコードの [15](#)

number [153](#)

IccRBA クラス (続き)

operator!= [153](#)

operator= [152](#)

operator== [152, 153](#)

IccResource

「read」メソッドから返される IccBuf 参照内のデータの有効範囲内 [44](#)

基本クラスにおける [6, 7](#)

ポリモフィック動作における [41](#)

ポリモフィック動作の例における [42](#)

リソース・クラスにおける [9](#)

IccResource クラス内 [157](#)

IccResource クラス

概要 [6, 7](#)

コンストラクター [157](#)

サブクラスの処理 [14](#)

actionOnCondition [157](#)

ActionOnCondition [162](#)

actionOnConditionAsChar [158](#)

actionsOnConditionsText [158](#)

clear [158](#)

condition [159](#)

conditionText [159](#)

ConditionType [163](#)

get [159](#)

handleEvent [159](#)

HandleEventReturnOpt [163](#)

id [159](#)

isEDFOn [159](#)

isRouteOptionOn [160](#)

name [160](#)

put [160](#)

routeOption [160](#)

setActionOnAnyCondition [160](#)

setActionOnCondition [161](#)

setActionsOnConditions [161](#)

setEDF [161](#)

setRouteOption [161, 162](#)

IccResource サブクラスの処理

バッファ・オブジェクトにおける [14](#)

IccBuf クラスにおける [14](#)

IccResourceId

基本クラスにおける [6, 7](#)

リソース識別クラスにおける [7](#)

C++ 例外およびファウンデーション・クラスにおける [34](#)

IccResourceId クラス

概要 [6, 7](#)

コンストラクター [163, 164](#)

name [164](#)

nameLength [164](#)

operator= [164](#)

IccRRN

通常レコードの削除における [17](#)

ファイル制御における [15](#)

レコードの書き込みにおける [16](#)

レコードの参照における [17](#)

レコードの読み取りにおける [15](#)

IccRecordIndex クラス内 [154](#)

IccRRN クラス内 [165](#)

RRDS レコードの読み取りにおける [16](#)

IccRRN クラス

コンストラクター [165](#)

読み取り、レコードの [15](#)

IccRRN クラス (続き)
 number [166](#)
 operator!= [166](#)
 operator= [165](#)
 operator== [166](#)

IccSession
 バッファ・オブジェクトにおける [12](#)

IccSysId
 プログラム制御における [20](#)
 IccSysId クラス内 [186](#)

IccTask
 サポート・クラスにおける [10](#)
 トランザクション開始の例における [23](#)
 パラメーター・レベルにおける [38](#)
 C++ 例外およびファウンデーション・クラスにおける [34](#)
 IccAlarmRequestId クラス内 [71](#)
 IccTask クラス内 [193](#)
 singleton クラスにおける [12](#)

IccTempStore::readItem
 「read」メソッドから返される IccBuf 参照内のデータの有効範囲内 [44](#)

IccTempStore::readNextItem
 「read」メソッドから返される IccBuf 参照内のデータの有効範囲内 [44](#)

IccTermId
 基本クラスにおける [6](#)
 端末管理における [27](#)
 端末管理の例における [28](#)
 トランザクション開始の例における [22](#)
 C++ 例外およびファウンデーション・クラスにおける [34](#), [35](#)
 IccTermId クラス内 [210](#)

IccTermId クラス
 概要 [6](#)
 コンストラクター [210](#)
 operator= [210](#), [211](#)

IccTerminal
 端末からのデータの受信における [27](#)
 端末管理における [27](#)
 端末管理の例における [28](#)
 端末に関する情報の検索における [28](#)
 バッファ・オブジェクトにおける [12](#)
 リソース・クラスにおける [9](#)
 IccTerminalData クラス内 [226](#)
 singleton クラスにおける [12](#)

IccTerminal::receive
 「read」メソッドから返される IccBuf 参照内のデータの有効範囲内 [44](#)

IccTerminalData
 端末管理における [27](#)
 端末管理の例における [28](#)
 端末に関する情報の検索における [28](#)
 IccTerminalData クラス内 [226](#)

IccTransId
 基本クラスにおける [6](#)
 トランザクション開始の例における [22](#)
 パラメーター受け渡し規則内 [44](#)
 IccResourceId クラス内 [163](#)
 IccTransId クラス内 [241](#)

IccTransId クラス
 概要 [6](#)
 コンストラクター [241](#)
 operator= [241](#), [242](#)

initializeEnvironment
 関数内 [59](#)
 ストレージ管理における [43](#)
 メソッド・レベルにおける [38](#)
 Icc 構造内 [59](#)

insert
 一時記憶域の例における [27](#)
 IccBuf クラスにおける [80](#)
 IccBuf コンストラクターにおける [14](#)

instance
 IccAbendData クラス内 [65](#)
 IccConsole クラス内 [93](#)
 IccControl クラス内 [98](#)
 IccStartRequestQ クラス内 [181](#)
 IccSystem クラス内 [191](#)
 IccTask クラス内 [197](#)
 IccTerminal クラス内 [213](#)
 singleton クラスにおける [12](#)

isAddable
 ESDS レコードの書き込みにおける [17](#)
 IccFile クラス内 [115](#)
 KSDS レコードの書き込みにおける [17](#)
 RRDS レコードの書き込みにおける [17](#)

isReadable
 ESDS レコードの読み取りにおける [16](#)
 IccFile クラス内 [116](#)
 KSDS レコードの読み取りにおける [16](#)
 RRDS レコードの読み取りにおける [16](#)

key (パラメーター)
 コンストラクター [138](#)
 ファイル制御の例における [18](#)
 operator!= [140](#)
 operator= [139](#)
 operator== [139](#)

keyLength
 IccFile クラス内 [117](#)
 KSDS レコードの書き込みにおける [17](#)
 KSDS レコードの読み取りにおける [16](#)

keyPosition
 IccFile クラス内 [117](#)
 KSDS レコードの書き込みにおける [17](#)
 KSDS レコードの読み取りにおける [16](#)

KSDS レコードの書き込み
 ファイル制御における [16](#)
 レコードの書き込みにおける [16](#)

KSDS レコードの読み取り
 ファイル制御における [16](#)
 レコードの読み取りにおける [16](#)

line
 端末に関する情報の検索における [28](#)
 IccTerminal クラス内 [213](#)

monthOfYear
 日時サービスの例における [30](#)
 IccAbsTime クラス内 [68](#)
 IccClock クラス内 [89](#)

MyTempStore
 自動条件処理 (callHandleEvent) における [36](#)

new
 ストレージ管理における [43](#)

number
 IccException クラス内 [111](#)
 IccJournalId クラス内 [135](#)
 IccMessage クラス内 [144](#)
 IccRBA クラス内 [153](#)

number (続き)
 IccRRN クラス内 [166](#)
 IccTask クラス内 [198](#)
 RRDS レコードの書き込みにおける [17](#)

obj (パラメーター)
 オブジェクトの使用における [6](#)

programId
 メソッド・レベルにおける [38](#)
 IccControl クラス内 [98](#)
 public メソッド [98](#)

readNextRecord
 レコードの参照における [18](#)
 IccFileIterator クラス内 [126](#)
 public メソッド [126](#)

readRecord
 レコードの更新における [17](#)
 レコードの読み取りにおける [16](#)
 ロックされたレコードの削除における [17](#)
 C++ 例外およびファウンデーション・クラスにおける [35](#)
 IccFile クラス内 [118](#)
 「read」メソッドから返される IccBuf 参照内のデータの有効
 範囲
 その他 [44](#)

receive
 端末からのデータの受信における [27](#)

receive3270data
 端末からのデータの受信における [27](#)

recordFormat
 ESDS レコードの書き込みにおける [17](#)
 ESDS レコードの読み取りにおける [16](#)
 IccFile クラス内 [119](#)
 RRDS レコードの書き込みにおける [17](#)
 RRDS レコードの読み取りにおける [16](#)

recordIndex
 ESDS レコードの書き込みにおける [17](#)
 ESDS レコードの読み取りにおける [16](#)
 IccFile クラス内 [119](#)
 KSDS レコードの書き込みにおける [17](#)
 KSDS レコードの読み取りにおける [16](#)
 RRDS レコードの書き込みにおける [17](#)
 RRDS レコードの読み取りにおける [16](#)

recordLength
 ESDS レコードの書き込みにおける [17](#)
 ESDS レコードの読み取りにおける [16](#)
 IccFile クラス内 [119](#)
 KSDS レコードの書き込みにおける [17](#)
 KSDS レコードの読み取りにおける [16](#)
 RRDS レコードの書き込みにおける [17](#)
 RRDS レコードの読み取りにおける [16](#)

registerRecordIndex
 レコードの書き込みにおける [16](#)
 ESDS レコードの書き込みにおける [17](#)
 ESDS レコードの読み取りにおける [16](#)
 IccFile クラス内 [119](#)
 KSDS レコードの書き込みにおける [17](#)
 KSDS レコードの読み取りにおける [16](#)
 RRDS レコードの書き込みにおける [17](#)
 RRDS レコードの読み取りにおける [16](#)

replace
 IccBuf クラスにおける [85](#)
 IccBuf コンストラクターにおける [14](#)

reset
 レコードの参照における [18](#)

reset (続き)
 IccFileIterator クラス内 [127](#)
 IccStartRequestQ クラス内 [182](#)

retrieveData
 開始データへのアクセスにおける [21](#)
 EXEC CICS 呼び出しとファウンデーション・クラス・メ
 ソッドのマッピング [46](#)
 IccStartRequestQ クラス内 [180, 182](#)

rewriteRecord
 レコードの更新における [17](#)
 IccFile クラス内 [120](#)

RRDS レコードの書き込み
 ファイル制御における [17](#)
 レコードの書き込みにおける [17](#)

RRDS レコードの読み取り
 ファイル制御における [16](#)
 レコードの読み取りにおける [16](#)

send
 端末管理の例における [28](#)

sendLine
 端末管理の例における [28](#)
 ファイル制御の例における [19](#)
 IccTerminal クラス内 [219, 220](#)

set...
 端末へのデータの送信における [27](#)

setColor
 端末管理の例における [28](#)
 IccTerminal クラス内 [221](#)

setHighlight
 端末管理の例における [28](#)
 IccTerminal クラス内 [221](#)

setKind
 ファイル制御の例における [18](#)
 IccKey クラス内 [140](#)

setRouteOption
 トランザクション開始の例における [22, 23](#)
 プログラム制御における [20](#)
 IccResource クラス内 [161, 162](#)
 public メソッド [161, 162](#)

Singleton クラス
 リソース・オブジェクトの作成における [12](#)
 CICS リソースの使用における [12](#)

start
 トランザクション開始の例における [23](#)
 トランザクションの開始における [21](#)
 パラメーター受け渡し規則内 [44](#)
 EXEC CICS 呼び出しとファウンデーション・クラス・メ
 ソッドのマッピング [46](#)
 IccRequestId クラス内 [155](#)
 IccStartRequestQ クラス内 [180, 184](#)

startRequestQ
 トランザクション開始の例における [22, 23](#)
 IccControl クラス内 [100](#)

startType
 トランザクション開始の例における [23](#)
 IccTask クラス内 [199](#)

throw
 例外処理 (throwException) における [37](#)
 C++ 例外およびファウンデーション・クラスにおける [33](#)

try
 例外処理 (throwException) における [37](#)
 C++ 例外およびファウンデーション・クラスにおける [33, 34](#)

- try (続き)
 - main 関数における [248](#)
- tryNumber
 - C++ 例外およびファウンデーション・クラスにおける [33, 34](#)
- type
 - C++ 例外およびファウンデーション・クラスにおける [34](#)
 - IccException クラス内 [111](#)
 - IccFile クラス内 [121](#)
 - IccRecordIndex クラス内 [154](#)
 - IccSemaphore クラス内 [168](#)
 - IccTime クラス内 [234](#)
- waitForAID
 - 端末管理の例における [29](#)
 - IccTerminal クラス内 [224](#)
- writeItem
 - 一時記憶域における [25](#)
 - 一時データにおける [24](#)
 - 項目の書き込みにおける [26](#)
 - データの書き込みにおける [24](#)
 - リソース・オブジェクトでのメソッドの呼び出しにおける [12](#)
 - C++ 例外およびファウンデーション・クラスにおける [34](#)
 - IccDataQueue クラス内 [104](#)
 - IccResource サブクラスの処理における [14, 15](#)
 - IccTempStore クラス内 [206, 207](#)
- writeRecord
 - ファイル制御の例における [18](#)
 - レコードの書き込みにおける [16](#)
 - IccFile クラス内 [122](#)
 - IccJournal クラス内 [133](#)
 - KSDS レコードの書き込みにおける [17](#)
 - RRDS レコードの書き込みにおける [17](#)
- writeRecord メソッド
 - IccFile クラス [16](#)
- 一時記憶域
 - 一時記憶域の例 [26](#)
 - 概要 [25](#)
 - 項目の書き込み [26](#)
 - 項目の更新 [26](#)
 - 項目の削除 [26](#)
 - 項目の読み取り [26](#)
 - 例 [26](#)
 - CICS Service の使用における [25](#)
- 一時記憶域の例
 - 一時記憶域における [26](#)
 - CICS Service の使用における [26](#)
- 一時データ
 - 一時データ管理の例 [24](#)
 - 概要 [24](#)
 - キューの削除 [24](#)
 - データの書き込み [24](#)
 - データの読み取り [24](#)
 - 例 [24](#)
 - CICS Service の使用における [24](#)
- 一時データ管理の例
 - 一時データにおける [24](#)
 - CICS Service の使用における [24](#)
- オブジェクト
 - 削除 [6](#)
 - 作成 [5](#)
 - 使用 [6](#)
- オブジェクトの削除
 - C++ オブジェクト内 [6](#)
- オブジェクトの作成
 - C++ オブジェクト内 [5](#)
- オブジェクト・レベル
 - 条件、エラー、および例外における [38](#)
 - プラットフォームの相違における [38](#)
- オブジェクトを使用
 - C++ オブジェクト内 [6](#)
- 開始データへのアクセス
 - トランザクションの非同期の開始における [21](#)
 - CICS Service の使用における [21](#)
- キー
 - 完全 [16](#)
 - 総称 [16](#)
- 基本クラス
 - 概要 [6](#)
 - ファウンデーション・クラスの概要における [6](#)
- キャンセル
 - 満了していない開始要求のキャンセルにおける [21](#)
- キューの削除
 - 一時データにおける [24](#)
 - CICS Service の使用における [24](#)
- クラス
 - 基本 [6](#)
 - サポート [10](#)
 - リソース [9](#)
 - リソース識別 [7](#)
 - singleton [12](#)
- 項目の書き込み
 - 一時記憶域における [26](#)
 - CICS Service の使用における [26](#)
- 項目の更新
 - 一時記憶域における [26](#)
 - CICS Service の使用における [26](#)
- 項目の削除
 - 一時記憶域における [26](#)
 - CICS Service の使用における [26](#)
- 項目の読み取り
 - 一時記憶域における [26](#)
 - CICS Service の使用における [26](#)
- サポート・クラス
 - ファウンデーション・クラスの概要における [10](#)
- 自動条件処理 (callHandleEvent)
 - 条件、エラー、および例外における [36](#)
 - CICS 条件における [36](#)
- 重大エラー処理 (abendTask)
 - 条件、エラー、および例外における [37](#)
 - CICS 条件における [37](#)
- 手動条件処理 (noAction)
 - 条件、エラー、および例外における [36](#)
 - CICS 条件における [36](#)
- 条件、エラー、および例外
 - オブジェクト・レベル [38](#)
 - 自動条件処理 (callHandleEvent) [36](#)
 - 重大エラー処理 (abendTask) [37](#)
 - 手動条件処理 (noAction) [36](#)
 - パラメーター・レベル [38](#)
 - メソッド・レベル [38](#)
 - 例外処理 (throwException) [36](#)
- ストレージ管理
 - その他 [43](#)
- その他
 - ポリモフィック動作の例 [42](#)

端末	
検索対象 28	
データの受信 27	
データの送信 27	
端末からのデータの受信	
端末管理における 27	
CICS Service の使用における 27	
端末管理	
概要 27	
情報の検索 28	
端末からのデータの受信 27	
端末管理の例 28	
端末に関する情報の検索 28	
端末へのデータの送信 27	
データの受信 27	
データの送信 27	
例 28	
CICS Service の使用における 27	
端末管理の例	
端末管理における 28	
CICS Service の使用における 28	
端末に関する情報の検索	
端末管理における 28	
CICS Service の使用における 28	
端末へのデータの送信	
端末管理における 27	
CICS Service の使用における 27	
通常レコードの削除	
ファイル制御における 17	
レコードの削除における 17	
データ域の拡張性	
バッファ・オブジェクトにおける 13	
IccBuf クラスにおける 13	
データ域の所有権	
バッファ・オブジェクトにおける 13	
IccBuf クラスにおける 13	
データの書き込み	
一時データにおける 24	
CICS Service の使用における 24	
データの読み取り	
一時データにおける 24	
CICS Service の使用における 24	
テスト	
C++ 例外およびファウンデーション・クラスにおける 33 , 34	
トランザクション開始の例	
トランザクションの非同期の開始における 21	
CICS Service の使用における 21	
トランザクションの開始	
トランザクションの非同期の開始における 21	
CICS Service の使用における 21	
トランザクションの非同期の開始	
開始データへのアクセス 21	
トランザクション開始の例 21	
トランザクションの開始 21	
満了していない開始要求のキャンセル 21	
CICS Service の使用における 21	
トレース	
トレース出力の活動化 32	
日時サービス	
日時サービスの例 29	
CICS Service の使用における 29	
日時サービスの例	
日時サービスにおける 29	
日時サービスの例 (続き)	
CICS Service の使用における 29	
バッファ・オブジェクト	
データ域の拡張性 13	
データ域の所有権 13	
IccBuf コンストラクター 13	
IccBuf メソッド 14	
IccResource サブクラスの処理 14	
パラメーター受け渡し規則	
その他 44	
パラメーター・レベル	
条件、エラー、および例外における 38	
プラットフォームの相違における 38	
ファイル制御	
更新、レコードの 17	
削除、レコードの 17	
通常レコードの削除 17	
ファイル制御の例 18	
例 18	
レコードの書き込み 16	
レコードの更新 17	
レコードの再書き込み 17	
レコードの削除 17	
レコードのブラウズ 17	
レコードの読み取り 15	
ロックされたレコードの削除 17	
CICS Service の使用における 15	
ESDS レコードの書き込み 17	
ESDS レコードの読み取り 16	
KSDS レコードの書き込み 16	
KSDS レコードの読み取り 16	
RRDS レコードの書き込み 17	
RRDS レコードの読み取り 16	
ファイル制御の例	
ファイル制御における 18	
CICS Service の使用における 18	
ファウンデーション・クラスの異常終了コード	
条件、エラー、および例外における 33	
プラットフォームの相違	
オブジェクト・レベル 38	
条件、エラー、および例外における 37	
パラメーター・レベル 38	
メソッド・レベル 38	
プログラム制御	
概要 19	
例 19	
CICS Service の使用における 19	
プログラムのコンパイル	
コンパイル、実行、およびデバッグにおける 30	
プログラムの実行	
コンパイル、実行、およびデバッグにおける 32	
プログラムのデバッグ	
コンパイル、実行、およびデバッグにおける 32	
ヘッダー・ファイル	
インストール済みの内容 1 , 31	
場所 2	
ポリモフィック動作	
その他 39	
ポリモフィック動作の例 42	
ポリモフィック動作の例	
その他 42	
ポリモフィック動作における 42	
満了していない開始要求のキャンセル	
トランザクションの非同期の開始における 21	

満了していない開始要求のキャンセル (続き)

CICS Service の使用における [21](#)

メソッド・レベル

条件、エラー、および例外における [38](#)

プラットフォームの相違における [38](#)

目的

ファイル制御の例における [18](#)

リソース・オブジェクト

作成 [11](#)

リソース・オブジェクトでのメソッドの呼び出し

ファウンデーション・クラスの概要における [12](#)

CICS リソースの使用における [12](#)

リソース・オブジェクトの作成

ファウンデーション・クラスの概要における [11](#)

CICS リソースの使用における [11](#)

singleton クラス [12](#)

リソース・クラス

ファウンデーション・クラスの概要における [9](#)

リソース識別クラス

ファウンデーション・クラスの概要における [7](#)

例外処理 (throwException)

条件、エラー、および例外における [36](#)

CICS 条件における [36](#)

レコードの書き込み

ファイル制御における [16](#)

CICS Service の使用における [16](#)

ESDS レコードの書き込み [17](#)

KSDS レコードの書き込み [16](#)

RRDS レコードの書き込み [17](#)

レコードの更新

ファイル制御における [17](#)

CICS Service の使用における [17](#)

レコードの削除

通常レコードの削除 [17](#)

ファイル制御における [17](#)

ロックされたレコードの削除 [17](#)

CICS Service の使用における [17](#)

レコードのブラウズ

ファイル制御における [17](#)

CICS Service の使用における [17](#)

レコードの読み取り

ファイル制御における [15](#)

CICS Service の使用における [15](#)

ESDS レコードの読み取り [16](#)

KSDS レコードの読み取り [16](#)

RRDS レコードの読み取り [16](#)

ロックされたレコードの削除

ファイル制御における [17](#)

レコードの削除における [17](#)

buffer (パラメーター)

コンストラクター [78](#)

ポリモフィック動作における [41](#)

operator« [83](#), [214](#)

operator!= [82](#)

operator+= [82](#)

operator= [81](#)

operator== [82](#)

put [104](#), [131](#), [160](#), [205](#), [206](#)

registerData [182](#)

rewriteRecord [120](#)

send [217](#), [218](#)

send3270Data [218](#), [219](#)

sendLine [220](#)

writeRecord [122](#)

clear

ポリモフィック動作における [41](#)

ポリモフィック動作の例における [42](#)

IccDataQueue クラス内 [103](#)

IccResource クラス内 [158](#)

IccTempStore クラス内 [205](#)

IccTerminal クラス内 [212](#)

condition

手動条件処理 (noAction) における [36](#)

リソース・クラスにおける [9](#)

IccEvent クラス内 [108](#)

IccResource クラス内 [159](#)

Form

ポリモフィック動作における [41](#)

get

ポリモフィック動作における [41](#)

ポリモフィック動作の例における [42](#)

IccDataQueue クラス内 [104](#)

IccResource クラス内 [159](#)

IccSession クラス内 [174](#)

IccTempStore クラス内 [205](#)

IccTerminal クラス内 [213](#)

IccProgram クラス

コンストラクター [146](#), [147](#)

プログラム制御 [19](#)

address [147](#)

clearInputMessage [147](#)

CommitOpt [150](#)

entryPoint [147](#)

length [147](#)

link [148](#)

load [148](#)

LoadOpt [150](#)

setInputMessage [149](#)

unload [149](#)

IccTempstore

IccResource サブクラスの処理における [14](#)

operator=

ファイル制御の例における [18](#)

IccAbsTime クラス内 [69](#)

IccAlarmRequestId クラス内 [72](#)

IccBuf クラスにおける [81](#)

IccConvId クラス内 [102](#)

IccDataQueueId クラス内 [106](#)

IccFileId クラス内 [125](#)

IccGroupId クラス内 [129](#)

IccJournalId クラス内 [135](#)

IccJournalTypeId クラス内 [137](#)

IccKey クラス内 [139](#)

IccLockId クラス内 [142](#)

IccPartnerId クラス内 [145](#)

IccProgramId クラス内 [151](#)

IccRBA クラス内 [152](#)

IccRequestId クラス内 [156](#)

IccResource サブクラスの処理における [14](#), [15](#)

IccResourceId クラス内 [164](#)

IccRRN クラス内 [165](#)

IccSysId クラス内 [187](#)

IccTempStoreId クラス内 [209](#)

IccTermId クラス内 [210](#), [211](#)

IccTimeInterval クラス内 [236](#)

IccTimeOfDay クラス内 [238](#)

IccTPNameId クラス内 [240](#)

IccTransId クラス内 [241](#), [242](#)

- operator= (続き)
 - IccUserId クラス内 [246](#), [247](#)
 - protected メソッド [164](#)
 - public メソッド [69](#), [236](#)
- print
 - ポリモフィック動作における [41](#)
- put
 - ポリモフィック動作における [41](#)
 - ポリモフィック動作の例における [42](#)
 - IccConsole クラス内 [94](#)
 - IccDataQueue クラス内 [104](#)
 - IccJournal クラス内 [131](#)
 - IccResource クラス内 [160](#)
 - IccSession クラス内 [175](#)
 - IccTempStore クラス内 [205](#)
 - IccTerminal クラス内 [216](#)
- readItem
 - 「read」メソッドから返される IccBuf 参照内のデータの有効範囲内 [44](#)
 - 一時記憶域における [25](#)
 - 一時記憶域の例における [27](#)
 - 一時データにおける [24](#)
 - 項目の読み取りにおける [26](#)
 - データの読み取りにおける [24](#)
 - IccDataQueue クラス内 [104](#)
 - IccResource サブクラスの処理における [14](#), [15](#)
 - IccTempStore クラス内 [206](#)
- ... (パラメーター)
 - sendLine [220](#)

