# Nintendo Wi-Fi Connection

## TWL-DWC プログラミングマニュアル

## ― 簡易データベース編 ―

Ver 2.0.6

任天堂株式会社発行

# 目次



# 図

# 表



# コード

# 改訂履歴

| 版 | 改訂日 | 改 訂 内 容 | 承認者 | 担当者 |
|---|---|---|---|---|
| 2.0.6 | 2010-02-02 | 非 SSL 通信の機能を削除したことに伴い、3.1 簡易データベースライブラリの初期化 を修正。 | | |
| 2.0.5 | 2010-01-05 | 4 ガイドラインを追加。<br>SSL 対応に伴い 3 章のサンプルコード、説明文を修正。<br>ファイルアップロード/ダウンロード関数で進行状況を取得するコールバック関数を指定できるようになったことに伴い、コード 3.5 を修正。 | | |
| 2.0.4 | 2009-05-07 | 1 はじめに を修正。 まず GameSpy アカウントの取得及び NWCWMS での申請が必要な旨を追記。 | | |
| 2.0.3 | 2008-03-23 | 用語統一のため、「プロファイル ID」を「GS プロファイル ID」に変更。 | | |
| 2.0.2 | 2008-11-28 | GHTTP の非公開化に伴い、3.4 DWC_GHTTP への依存についての項目を改編。 | | |
| 2.0.1 | 2008-11-20 | 3.2 非同期処理が終わらないことがあるので、タイムアウト処理を入れる必要があることについて追記。 | | |
| 2.0.0 | 2008-10-10 | NITRO-DWC から TWL-DWC への移行に対応。 | | |
| 1.0.4 | 2008-7-25 | 2.3.1 backend ownertype が未実装な旨を注記 | | |
| 1.0.3 | 2008-6-18 | 表 2-2 Short と Int の説明の誤記を修正 | | |
| 1.0.2 | 2008-3-24 | 1 ランキングライブラリとの併用ができない旨を追記<br>3.2.3 ソース内にキャッシュフラグに関する記述を追加 | | |
| 1.0.1 | 2007-10-2 | 3.2.4 ファイルに関する項目をまとめて追加<br>3.2.3 誤字修正<br>2.4.2 誤字修正<br>2.3.6 言い回しの修正<br>2.3.5 フィールドに関する言い回しの修正<br>2.3.4 “プレファレンス”という言葉を置き換え<br>1 sake の表記追加 | | |
| 1.0.0 | 2007-9-28 | 初版 | | |

# 1　はじめに

本ドキュメントは、データベース管理サイトの使用方法、TWL-DWC(以下 DWC)の簡易データベースライブラリの使用方法について解説するものです。DWC の簡易データベースライブラリは、GameSpy の”sake”と呼ばれるデータベースを利用したライブラリで、主に以下の機能が提供されます。

- データ（ファイル）のアップロード
- データ（ファイル）のダウンロード
- データの評価
- データの検索

簡易データベースライブラリでは、整数、浮動小数、文字列、日付と時間、ファイルや任意のバイナリデータ等をデータベースサーバに格納、取得することが可能です（ファイルはデータベースとは別のファイルサーバーへ格納されます）。ユーザー固有データにも対応しており、他のユーザーへの公開、非公開を設定できます。

デベロッパーは、まず GameSpy の sake 管理サイト（http://tools.gamespy.net/SakeAdmin/）へログインするための GameSpy アカウントを http://login.gamespy.com より取得します。アカウントに使用するメールアドレスは、必ず御社ドメインのものを使用してください。

アカウント取得後、サーバ側の設定及び仕様の確認のため、NWCWMS(Nintendo WiFi Connection Workflow Management System : https://nwcwms.nintendo.co.jp/)　から、Sake の利用申請をして下さい。 フェーズが開発中になった時点で必要な設定情報を NWCWMS から取得することができます。

ゲームを開発する段階になったら、まず初めに、sake 管理サイトにて、データベーススキーマを設定する必要があります。設定後、ゲームからこのスキーマで定義したデータベースへ、簡易データベースライブラリを利用したアクセスが可能となります。

実際の使用においては、「Nintendo Wi-Fi Connection プログラミングガイドライン」に従ってください。
なお、ランキングライブラリとの併用はできませんのでご注意ください。

# 2　データベースサーバ

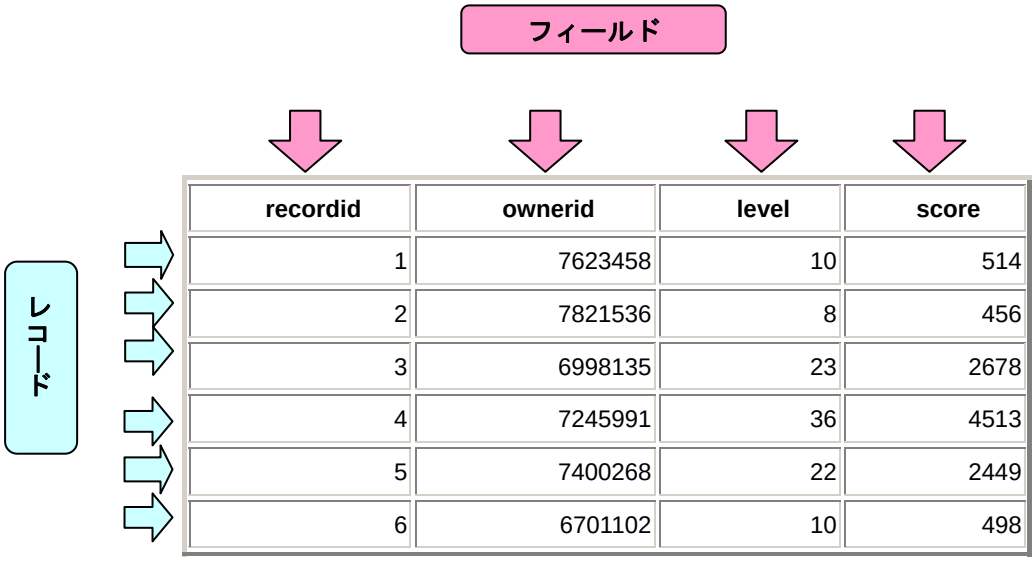
この章では、簡易データベースライブラリで使用するデータベースサーバの概要、及び管理ウェブサイトを用いての設定方法について解説します。

## 2.1　データベースの構成

各ゲームのデータベースは、デベロッパーが作成する複数のテーブルで構成されています。

## 2.2　テーブル（table）

テーブル内の各データエントリーを、レコードと呼びます。各テーブルには複数のフィールドがあり、各フィールドはそのテーブルのレコードそれぞれに格納するデータ内容を表します。SQL 用語では、フィールドは列、レコードは行に値します。




| recordid | ownerid | level | score |
|---|---|---|---|
| 1 | 7623458 | 10 | 514 |
| 2 | 7821536 | 8 | 456 |
| 3 | 6998135 | 23 | 2678 |
| 4 | 7245991 | 36 | 4513 |
| 5 | 7400268 | 22 | 2449 |
| 6 | 6701102 | 10 | 498 |

**表 2-1　テーブルのフィールドとレコード**

## 2.3　テーブルの設定

### 2.3.1　オーナータイプ（Owner type）

~~オーナータイプは、テーブル内の各レコードの所有者を示すもので、profile と backend の 2 種類があります。~~
オーナータイプはテーブルの基本的なプロパティで、テーブル作成時に必ず設定します。テーブル作成後の変更はできません。

- **profile owner type**

オーナータイプが profile に設定されたテーブルは、テーブル内の各レコードを特定のユーザーが所有します。各レコードには、そのレコードを所有するプレーヤーを特定する値が含まれます。オーナーはレコード作成時のプレーヤーになります。profile オーナータイプは、プレーヤー毎の情報格納に使用します。

オーナータイプを profile に設定してテーブルを作成した場合、オーナーID フィールドが当該テーブルに自動的に付加されます。本フィールドは Int 型で、レコードを作成したプロファイルを格納します。テーブル内にレコードを作成すると、自動的で値が格納されます。この値は、レコードが削除されるまで変更されることはありません。バックエンドでのレコードのアクセス管理に使用しますが、ゲーム側からも簡易データベースライブラリを通じ、レコード作成者を調べるために使用できます。

- ~~**backend owner type**~~

~~オーナータイプが backend に設定されテーブルは、テーブル内の全レコードをバックエンドが所有します。通常は、デベロッパーが定期的に更新するような一般的汎用情報の格納に使用します。~~

注意：backend owner type は未実装です。開発の際には使用しないようにして下さい。

### 2.3.2　パーミッション（Permissions）

テーブルのパーミッションは、レコード作成、読込み、更新、削除の権限を制御します。パーミッションには、パブリック作成（public create）、パブリック読込み（public read）、オーナー更新（owner update）、オーナー削除（owner delete）の 4 種類あり、それぞれ TRUE または FALSE に設定します。

- パブリック作成（**public create**）

TRUE の場合、誰でも当該テーブル内のレコード作成ができます。オーナータイプがプロファイルのテーブルの場合は、クライアントが新規レコードを作成できるよう通常は TRUE に設定され、オーナータイプがバックエンドのテーブルの場合は、通常 FALSE になります。

- パブリック読み込み（**public read**）

TRUE の場合、誰でも当該テーブル内のレコードを読込めます。FALSE の場合、レコードを読込めるのはそのオーナーのみです。テーブルに格納したユーザー固有データの公開／非公開の制御に使用できます。

- オーナー更新（**owner update**）

レコード作成後に、オーナーがそのレコードを更新可能かどうかを決定します。本パーミッションが TRUE の場合、オーナーはレコードを何回でも更新可能です。ユーザーのプレファレンスを格納するテーブル等に使用します。FALSE の場合、オーナーはレコードの作成はできますが、更新はできません。本設定は 1 回限りのイベントを記録するためのテーブル等に使用します。

- オーナー削除（**owner delete**）

TRUE の場合、当該テーブル内でそのユーザーが作成したレコードを削除できます。例えばユーザーが集めた各アイテムをレコードとして持つテーブルの場合、そのユーザーがアイテムを売ったり無くした場合、レコードを削除できます。このパーミッションが FALSE の場合、レコードは削除できません。例えば、ユーザーのハイスコアを各レコードに保存するテーブル等に使用できます。

### 2.3.3 評価可能(Retable)オプション

評価可能オプションは、テーブル内のレコードをユーザーが評価可能かどうかを制御します。例えばレーシングゲームで、ユーザーの誰もがベストレースのビデオをアップロードできるようになっているとします。ビデオはそれぞれテーブル内のレコードとして格納されます。そのテーブルのパブリック読込みパーミッションが TRUE で(他のユーザーからビデオにアクセス可能)、そのテーブルの評価可能オプションが TRUE の場合、ユーザーは好きな／嫌いなビデオのレーティングをポストできます。この場合他のユーザーは、ビデオの平均レーティングと評価回数を見ることができます。複数のビデオのリストをソートしたりフィルターをかけるためにレーティングを使用することもできます。

テーブルの評価可能オプションが TRUE の場合、レーティング数を表す（num_ratings）フィールドと、平均レートを表す（average_rating）フィールドが当該テーブルに自動的に付加されます。フィールドに関する詳細は、以下 2.3.5 を参照ください。

レーティングの最大幅は 0 から 255 です（ゲーム側で内部でより狭い範囲へ絞り込むことも可能です)。レーティングの値は整数ですが、平均レーティングは浮動小数点数として返ります。評価は自分が作ったレコードに対しても行えます。

１つのレコードに対して、評価は１度だけしかできません。レコードを更新した場合でも、評価済みのレコードを評価することはできません。評価をリセットするには、評価済みのレコードを削除し、新規にレコードを作成する処理が必要になります。

### 2.3.4 オーナー1人あたりの上限(Limit Per Owner)

オーナー１人あたりの上限（Limit per owner）オプションは、ユーザーがテーブル内に同時に持つレコード数を一定の数以内に制限したい場合に使用します。本オプションが 0 に設定されている場合、ユーザーはいくつでもレコードを持つことができます。0 より大きい数値に設定されている場合、その数値が 1 ユーザーの当該テーブル内に所有可能な最大レコード数となります。

例えば、あるステージのハイスコアを格納するテーブルの場合、ハイスコアは２つ以上必要ないので、このテーブルのオーナー１人当たりの上限は１にできます。

さらに例を挙げると、ユーザーのキャラクターが持っている特殊アイテムを各レコードが表すようなテーブルの場合で、ゲームでユーザーが持てる特殊アイテムは 5 個までという制限を設けたとき、テーブルのオーナー１人あたりの上限は 5 となり、ユーザーはこれを超える数のレコードを当該テーブル内に保存できなくなります。ユーザーがテーブル内に 5 個のレコードを持っている場合、新規レコードを追加するためには、まず 5 個のうち 1 個を削除する必要があります。

### 2.3.5 フィールド(Field)

フィールドは、テーブル内の各レコードに格納されるそれぞれのデータを指します。表 2-1 の例では、レコード ID（recordid)、オーナーID（ownerid)、レベル（level)、スコア（score）の 4 個のフィールドがあります。テーブルに格納されている各レコードの、フィールドそれぞれに値が入ります。

全てのテーブルには、自動的に作成されるフィールドが 1 個以上あります。表 2-1 の例では、レコード ID とオーナーID がそれにあたります。

テーブルには必ずレコード ID フィールドがあり、テーブル作成時に自動的に付加されます。本フィールドは整数で、テーブル内の各レコードを一意的に特定するためのものです。テーブルにレコードを追加すると、バックエンドでそのレコード ID フィールドに自動的に値を割り当てます。このレコード ID は、その後当該テーブル内で本レコードを特定するために常に使用します。

デベロッパーは、管理ウェブサイトからフィールドの追加・管理を行うことができます。フィールドには名前を付ける必要がありますが、フィールド名は、そのフィールドが所属するテーブル内において固有であり、かつ以下に示す禁止語句以外の名前を設定する必要があります。

- フィールド名に設定してはいけない名前

file, fileid, size, name, create_time, downloads, profileid, ownerid, recordid, num_ratings, average_rating, my_rating

また、フィールドにはタイプがあり、そのフィールド内に格納されるデータの種類を定義します。タイプによっては、フィールドには最大長やデフォルト値がある場合もあります。全フィールドタイプを、簡単な説明とともに下表に一覧にします。

| タイプ名 | 範囲 | Default 値の有無 | Max Length の有無 | 説明 |
|---|---|---|---|---|
| Byte | 0 to 255 | Yes | No | 1 バイト符号無し整数 |
| Short | -32,768 to 32,767 | Yes | No | 2 バイト符号付き整数 |
| Int | -2,147,483,648 to 2,147,483,647 | Yes | No | 4 バイト符号付き整数 |
| Float | -1.79E308 to 1.79E308 | Yes | No | 8 バイト浮動小数点数 |
| AsciiString | up to 1000 chars | Yes | Yes | ASCII 文字列 |
| UnicodeString | up to 1000 chars | Yes | Yes | マルチバイト文字列 |
| Boolean | true or false | Yes | No | |
| DateAndTime | 1970 through 2038 | Yes | No | 秒単位<br><br>デフォルトで、レコード作成時の協定世界時（UTC）が設定される |
| BinaryData | up to 2000 bytes | No | Yes | 任意のバイナリデータ |
| FileID | n/a | No | No | アップロードされたファイルへの参照。内部では int として扱われる。 |

**表 2-2　フィールドタイプ**

デベロッパーが定義するフィールドに加えて、バックエンドで自動的に管理されるフィールドがあります。FileID フィールドに関連して自動付加されるメタデータフィールドにつきましては、3.2.4　にて説明します。

- 評価可能テーブル

評価可能オプションが設定されたテーブルでは、my_rating というフィールドタグを用いることで、自分がどのような評価をしたかを確認することができます。これは、検索の条件に指定することもでき、例えば、””my_rating > 4”というのを検索のフィルタに設定すれば、自分がした評価が５以上のレコードを取得することができます。自分が評価していないレコードの my_rating フィールドはデフォルトで（-1）に設定さ

れています。

また、@rated、@unrated というキーワードを検索のフィルタに使用できます。これらを用いれば、評価済みかどうかを簡単に判断して、評価したもの、してないものを取得することができます。

- 行数の取得

DWC_GdbSearchRecordsAsync 関数で、row フィールドを用いることで、ソート後当該レコードが何番目にあるかを取得することができます。例えば、ユーザーの順位などを取得する場合に便利です。

### 2.3.6 レコード（Record）

テーブルにどのようなデータを格納するかを定義するのはフィールドですが、実際のデータはレコード単位でテーブルに格納します。テーブルに格納されている各レコードの、当該テーブルのフィールドそれぞれに値が入ります。レコード ID フィールドの値は、当該テーブル内のレコードを一意的に特定します。レコードにアクセスするには、簡易データベースライブラリを利用します。テーブルのパーミッションに応じて、レコード作成、読込み、更新、削除が可能です。詳細については、後述の API の説明を参照ください。

## 2.4 Webインターフェースによるデータベース管理

デベロッパーは管理ウェブサイトからゲームのデータベーススキーマを設定します。この設定は、テーブルを作成し、そのテーブルにフィールドを追加することで行います。テーブルとフィールドのプロパティを設定することも可能です。その後ゲーム側は簡易データベースライブラリを介して本スキーマを使用します。本ウェブサイトのアドレスは、http://tools.gamespy.net/SakeAdmin/です。


**図 2-1　管理者ログイン画面**

ログインが済んでいれば、ゲームのデータベーススキーマの編集が可能となります。メインのゲーム選択（Game Selection）ページからゲームを選び、ゲームテーブル（Game Tables）ページへ移動します。

### 2.4.1　ゲームテーブルページ

ゲームテーブルページでは、選択したゲーム向けにこれまで作成された全てのテーブルと各テーブルのプロパティを表示します。テーブル ID（Table ID）列は、当該ゲームデータベース内の各テーブルを一意的に特定する短い文字列です。説明（Description）の列には、デベロッパーによるコメントが入ります。これは管理サイトのみで利用できるもので、デベロッパーはテーブルの目的等を書きとめておくことができます。

Edit ボタン、Fields ボタンをクリックすると、それぞれ別ページへ移動し、当該テーブルのテーブル設定、フィールド設定が可能です。（詳細は後述。）リセット（Reset）ボタンをクリックすると、当該テーブルの全レコードを削除します。削除（Delete）ボタンをクリックすると、ゲームに関連付けられたテーブル一覧からそのテーブルを削除します。この操作ではテーブルとそのレコードが実際に削除されるわけではありません。削

除されたテーブルは一覧には表示されなくなり、簡易データベースライブラリからアクセスできなくなります。

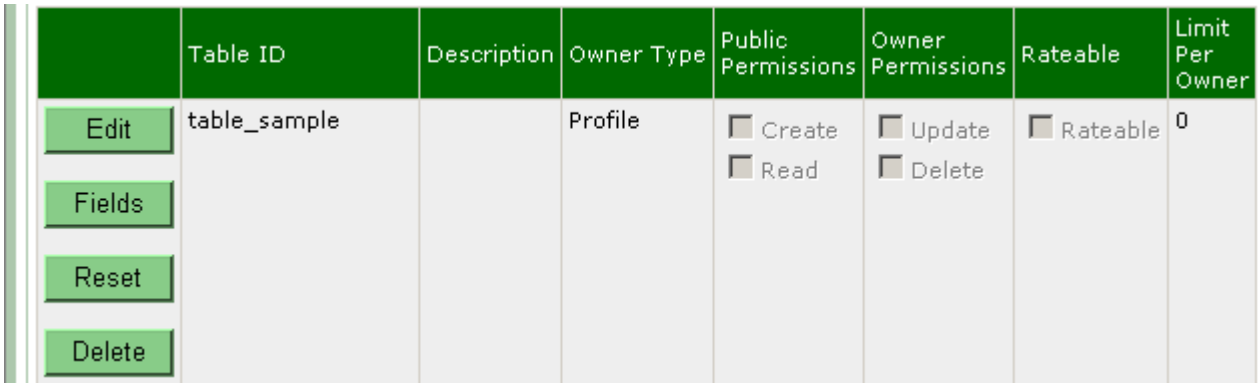



**図 2-2 ゲームテーブルページ(一部)**

## 2.4.2 ゲームテーブルの作成と更新

図 2-3 は、図 2-2 ゲームテーブル画面の下部にある、新規テーブル追加（Add a New Table）ボックスを示したものです。このボックスで、テーブル ID とオーナータイプを指定して新規テーブルを追加します。その他のプロパティは、テーブル作成後の設定が可能です。テーブル ID もテーブル作成後の変更は可能ですが、オーナータイプは変更できません。テーブル ID とオーナータイプ入力後、テーブル作成（Create Table）ボタンをクリックし、現在のゲームのテーブル一覧にテーブルを追加します。

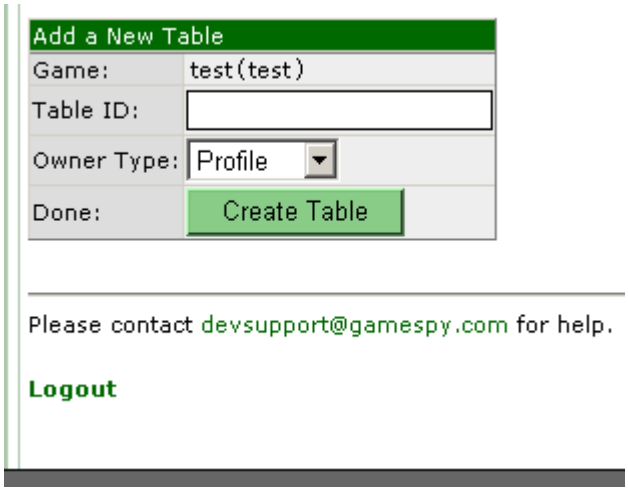



**図 2-3 テーブル作成フォーム**

図 2-2 ゲームテーブル画面において、各テーブルの左側にある Edit ボタンをクリックすると、テーブル設定ページに移ります（図 2-4）。テーブル設定では、オーナータイプを除き全て編集可能です。オーナータイプはテーブル作成時に設定します。編集が終わったら、更新（Update）ボタンをクリックして編集内容を保存するか、キャンセル（Cancel）ボタンをクリックして、編集内容をキャンセルします。

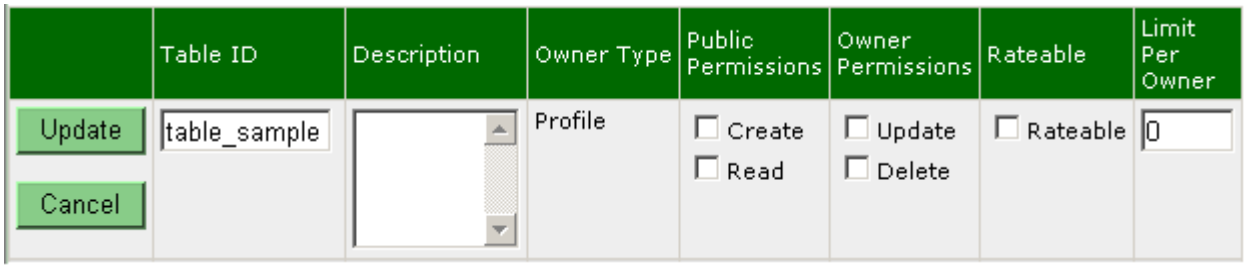



**図 2-4　テーブル設定画面（一部）**

### 2.4.3　フィールド設定

特定のテーブルのフィールド一覧を表示・編集するには、ゲームテーブルページでテーブルの左側にあるフィールド（Fields）ボタンをクリックします。選択したテーブルのゲームテーブルフィールド（Game Table Fields）ページが開きます。このページでは、そのテーブル内の全フィールドを含む一覧表が表示されます。この一覧表には、デベロッパー定義のフィールドに加え、バックエンドが自動的に作成したフィールドも含まれます。テーブルには必ずレコード ID フィールドがあり、テーブル内の各レコードを一意的に特定します。テーブルのオーナータイプがプロファイルの場合、そのレコードを作成したユーザーの GS プロファイル ID を含むオーナーID フィールドが表示されます。テーブルの評価可能オプションが TRUE の場合、前述のレーティング数と平均レーティングの 2 個のフィールドが含まれます。

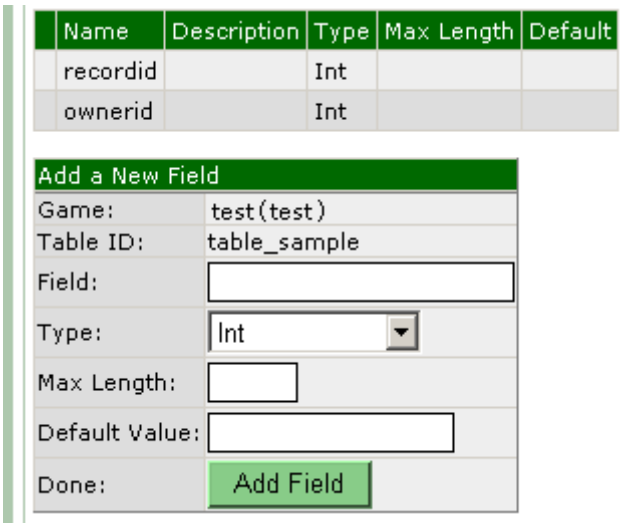


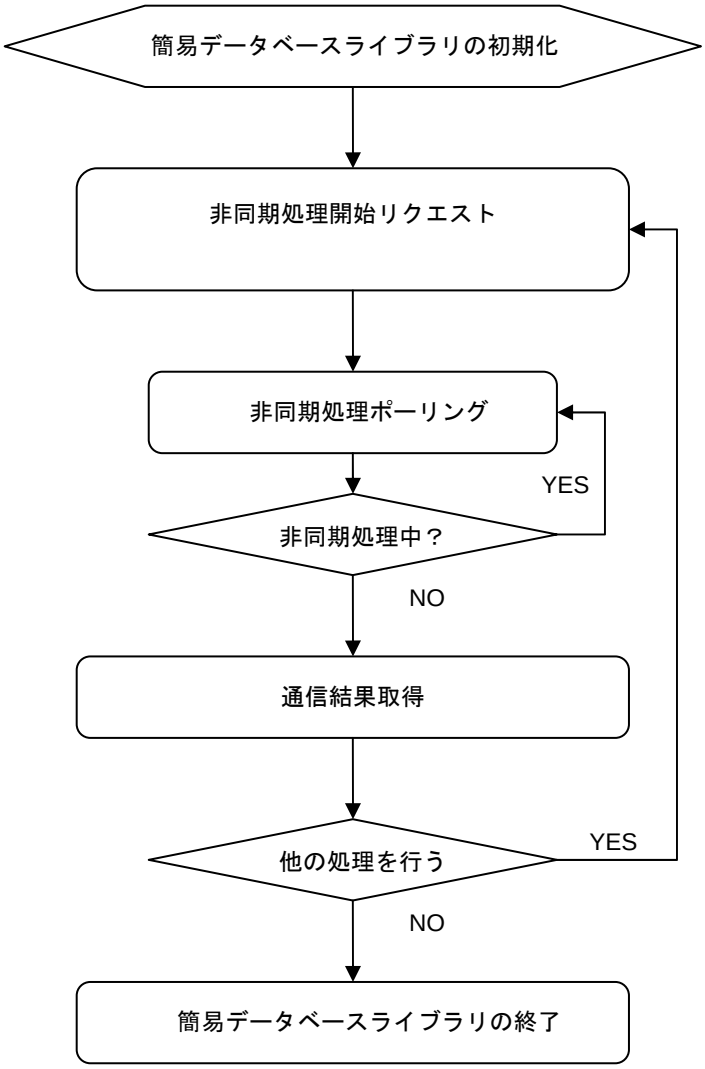

**図 2-5　フィールド設定フォーム**

名前（Name）列には、テーブル内固有のフィールド名が表示されます。説明（Description）の列には、デベロッパーによるコメントが入ります。これは管理サイトのみで利用できるもので、デベロッパーはフィールドの目的等を書きとめておくことができます。タイプ（Type）の列には、フィールド内に格納されたデータタイプが表示されます。前述のデータベースの項にて、全てのタイプを一覧にしてあります。最大長（Max Length）の列には、Ascii 文字列および Unicode 文字列フィールドの文字数上限と、バイナリデータフィールドの最大バイト数が表示されます。デフォルト（Default）列には、そのフィールドのタイプがデフォルト値に対応している場合、デフォルト値が表示されます。

一覧表中のほとんどのフィールドの左側には、ボタンが 2 個あります。編集（Edit）ボタンは、当該フィールドのプロパティ編集に使用します。フィールド名とその説明は常に編集可能です。最大長とデフォルト値に対応しているフィールドタイプの場合、双方とも常に編集可能です。フィールドタイプは作成後の編集はできません。削除（Delete）ボタンは、当該フィールドをテーブルから削除します。レコード ID とオーナーID のフィールドは、編集／削除できません。レーティング数と平均レーティングフィールドも、ゲームテーブルフィールドページから編集／削除できませんが、テーブルの評価可能プロパティを FALSE に設定することで双方のフィールドを削除することができます。

ゲームテーブルフィールドページの下の部分にある新規フィールドを追加（Add a New Field）ボックスで、新規フィールドを現在のテーブル内に作成します。まず新規フィールドの名前を入力し、ドロップダウンボックスからタイプを選択します。選択したタイプに応じ、最大長および／またはデフォルト値を入力します。これが必要かどうかは、表 2-2 を参照してください。名前、最大長、デフォルト値は、フィールド作成後の編集が可能ですが、タイプは作成時に限って設定できます。用意ができたら追加フィールド（Add field）ボタンをクリックして、フィールドを作成し一覧表に追加します。フィールドを作成したら、一覧表にそのフィールドの説明を入力できます。

# 3　クライアントプログラム




※非同期処理をキャンセルしたとき、またはエラーが発生したときには、
一度簡易データベースライブラリを終了する必要があります。

**図 3-1　処理の流れ**

## 3.1 簡易データベースライブラリの初期化

簡易データベースライブラリの使用にあたって、まず DWC_GdbInitialize 関数を呼び出して簡易データベースライブラリを初期化します。この関数の引数として渡すデータは以下のとおりです。

- ゲーム ID

弊社より返信された、Design Statement チェックシート A に記載されている gameID を指定します。このゲーム ID は、タイトルごとに固有に与えられるものなので、厳重に管理してください。

- ユーザーデータ

ユーザーデータの DWCUserData 構造体を指定します。このアカウントデータは認証済みのアカウントである必要があります。未だ一度も Nintendo Wi-Fi Connection に接続したことのない未認証のアカウントは初期化に失敗します。

- SSL 種別

サーバ認証のみ行う DWC_SSL_TYPE_SERVER_AUTH か、 サーバ認証とクライアント認証を行う DWC_SSL_TYPE_SERVER_CLIENT_AUTH を指定します。

※ Nitro/Twl では、クライアント認証に対応していないため、サーバ認証のみを行う DWC_SSL_TYPE_SERVER_AUTH を指定して下さい。

```
DWCGdbError res;

// 簡易部データベースライブラリの初期化
res = DWC_GdbInitialize(GAME_ID, &userdata, DWC_SSL_TYPE_SERVER_AUTH );

if( res != DWC_GDB_ERROR_NONE ){
        break; // 初期化失敗 エラー処理
}
```

**コード 3-1 簡易データベースライブラリ初期化例**

## 3.2 非同期処理

データベースサーバーとのやりとりには、非同期処理を用います。同時に実行できる非同期処理は一つまでです。また、非同期処理完了後に呼ばれるコールバック内で、非同期処理関数を呼び出すこともしないでください。

### 3.2.1 非同期処理ポーリング 通信結果状態の取得

非同期処理の関数を呼び出し、返値として DWC_GDB_ERROR_NONE が返された場合は、非同期処理が開始しています。DWC_GdbProcess 関数を、毎ゲームフレームを目安に呼び出し、ポーリングを行ってください。

DWC_GdbGetState 関数は、非同期処理中は DWC_GDB_STATE_IN_ASYNC_PROCESS を返します。この返り値を監視することで非同期処理の終了を検出できます。途中でエラーが発生した場合は、DWC_GDB_STATE_ERROR_OCCURED が返されます。エラーが発生すると、以後の処理を継続することができませんので、簡易データベースライブラリを終了した上で、初期化処理から再試行してください。

簡易データベースライブラリの非同期処理関数は、通信の切断等により稀にいつまでたっても終了しなくなる可能性がありますので、必ずアプリケーションでタイムアウトを設定し、DWC_GdbCancelRequest 関数によって非同期処理のキャンセルを行うようにして下さい。

以下、コード 3-2 は、非同期処理待ちの実装例を示したものです。

```
// 非同期処理関数呼び出し後

OSTick start_time = OS_GetTick();
DWCError err = DWC_ERROR_NONE;
int cancelflag = 1;

for (;;)
{
   DWCGdbState state  =  DWC_GdbGetState();
   if(state != DWC_GDB_STATE_IN_ASYNC_PROCESS &&
      state != DWC_GDB_STATE_IN_CANCEL_PROCESS)
   {
      break;
   }

   DWC_ProcessFriendsMatch();
   if ((err = DWC_GetLastError(NULL)) != DWC_ERROR_NONE)
  {
      // エラー処理
  }

   DWC_GdbProcess();

   // タイムアウト確認
   if(OS_TicksToMilliSeconds(OS_GetTick() - start_time) > TIMEOUT_MSEC
                              && cancelflag)
   {
      DWC_GdbCancelRequest();
      cancelflag = 0;
   }
   GameWaitVBlankIntr(); // Ｖブランク待ち
}

if(DWC_GdbGetState() != DWC_GDB_STATE_IDLE){
        break; // 失敗 エラー処理（コールバックは呼ばれていません）
}

if( DWC_GdbGetAsyncResult() != DWC_GDB_ASYNC_RESULT_SUCCESS){
        break; // 失敗、またはキャンセル（コールバックは呼ばれていません）
}
```

**コード 3-2 非同期処理ポーリング例**

### 3.2.2 レコードの作成

レコードの作成を行う非同期処理を開始するためには、DWC_GdbCreateRecordAsync 関数を呼び出します。引数の record_id に値が格納されるのは、DWC_GdbGetState 関数が DWC_GDB_STATE_IN_ASYNC_PROCESS を返さなくなり、コールバック関数が呼ばれた時点となります。

レコードを作成すると当該テーブルの recordid フィールドに、作成毎にインクリメントされた値が自動的に格納されます。この recordid の最大値は、2,147,483,648 であり、それを越えるとデータベースエラーとなりますので注意してください。

以下、コード 3-3 は、レコードの作成例を示したものです。

```
DWCGdbField fields[2];
int record_id;
DWCGdbError res;

fields[0].name = "score";
fields[0].type = DWC_GDB_FIELD_TYPE_INT;
fields[0].value.int_s32 = 150;
fields[1].name = "stage";
fields[1].type = DWC_GDB_FIELD_TYPE_BYTE;
fields[1].value.int_u8 = 5;


// レコード作成リクエスト開始
res = DWC_GdbCreateRecordAsync(“table_sample”, fields, 2, &record_id);

if( res != DWC_GDB_ERROR_NONE ){
        break; // 失敗 エラー処理
}

//  ここで非同期処理待ち
//  コード3-2参照
```

**コード 3-3 レコードの作成例**

### 3.2.3 指定した検索条件でレコードを取得する

レコードの検索、取得を行う非同期処理を開始するためには、DWC_GdbSearchRecordsAsync 関数を呼び出します。

処理が成功すると、引数 callback で指定したコールバック関数が呼ばれ、検索結果のレコードを受け取ることができます。 処理が失敗した場合、コールバックは呼ばれません。

以下、コード 3-4 はレコードの検索リクエスト例を示したものです。

```
DWCGdbError res;
field_num = 2;
const char* field_names[2] = {  "ownerid",  "my_score"  };
const DWCGdbSearchCond cond = {
        "my_date > '2007-9-11 00:00:00'",  // SQL の WHERE に相当
        "score",                           // SQL の ORDER BY に相当
        0,                          // 検索結果の何番目から取得するか
        50,                         // いくつ取得するか
        NULL,
        0,
        NULL,
        0,
        TRUE     // キャッシュフラグ。  サーバの負荷を低減するためにも、
                 // できるだけ有効にして下さい。
         };

DWC_GdbSearchRecordsAsync(       "table_sample",   // テーブル名
                                 field_names,      // フィールド名の配列
                                 field_num,        // フィールドの数
                                 &cond,         // 検索条件
                                 get_records_callback,  // コールバック（別途定義）
                                 NULL           // コールバックに渡すポインタ
);

if( res != DWC_GDB_ERROR_NONE ){
        break; // 失敗 エラー処理
}

//  ここで非同期処理待ち
//  コード3-2参照
```

**コード 3-4 レコード検索リクエスト例**

### 3.2.4 ファイルのアップロード／ダウンロード

DWC_GdbUploadFileAsync 関数を使うと、ファイルをファイルサーバーへアップロードすることができます。ファイルサーバーにアップロードされたファイルは、データベースに設定した FileID タイプのフィールドから当該ファイルを示す ID が無くなると、約 24 時間でサーバから削除されます。そのため、ファイルのアップロード後に、DWC_GdbCreateRecordAsync 関数を用いて、アップロードしたファイルの ID を FileID タイプのフィールドへ格納することを忘れないでください。

以下、コード 3-5 は、ファイルのアップロード例を示したものです。

```
int file_id;
s64 data = 83912736414123Ull;
DWCGdbField fields[1];
int record_id;
DWCGdbError res;

// ファイルアップロード
res = DWC_GdbUploadFileAsync(&data, sizeof(s64), NULL, &file_id, progress_callback);

if( res != DWC_GDB_ERROR_NONE ){
        break; // 失敗 エラー処理
}

//  ここで非同期処理待ち
//  コード3-2参照


fields[0].name = “file_data";
fields[0].type = DWC_GDB_FIELD_TYPE_INT;
fields[0].value.int_s32 = file_id;

// ファイル ID をデータベースへ登録
res = DWC_GdbCreateRecordAsync(“table_sample”, fields, 1, &record_id);

if( res != DWC_GDB_ERROR_NONE ){
        break; // 失敗 エラー処理
}

//  ここで非同期処理待ち
//  コード3-2参照


//
```

**コード 3-5 ファイルのアップロード/例**

また、ファイル ID は、メタデータフィールドを用いることで、当該ファイルについてサイズや名前などを取得することができます。取得できるメタデータは以下の５つになります。

- .size　—　ファイルサイズ（バイト単位）　Int 型
- .name　—　ファイル名
- .create_time　—　ファイルがアップロードされた時間（UTC）
- .downloads　—　ファイルがダウンロードされた回数　Int 型
- .profileid　—　ファイルをアップロードしたユーザのプロフィール ID　Int 型

これらメタデータの取得方法は、例えば、”sample_data”というフィールドにあるファイルのファイルサイズを取得する場合は、”sample_data.size”という指定の仕方をします。

コード 3-6 は、”sample_data” フィールド（FileID が格納されている）のメタデータフィールドを利用した検索を行うサンプルプログラムです。

```
DWCGdbError res;
field_num = 2;
const char* field_names[2] = {  "recordid",  "sample_data.name"  };
const DWCGdbSearchCond cond = {
        "sample_data.downloads > 10”,   // SQL の WHERE に相当
        "sample_data.size",           // SQL の ORDER BY に相当
        0,                          // 検索結果の何番目から取得するか
        50,                         // いくつ取得するか
       NULL, 0, NULL, 0, TRUE  };

DWC_GdbSearchRecordsAsync(  “table_sample”,   // テーブル名
                            field_names,     // フィールド名の配列
                            field_num,       // フィールドの数
                            &cond,           // 検索条件
                            get_records_callback,  // 完了コールバック（別途定義）
                            NULL             // コールバックに渡すポインタ
);

if( res != DWC_GDB_ERROR_NONE ){
        break; // 失敗 エラー処理
}

//  ここで非同期処理待ち
//  コード3-2参照
```

**コード 3-6 FileID メタデータ利用例**

## 3.3 簡易データベースライブラリの終了

すべての処理が完了した場合、あるいはエラーにより終了する場合は DWC_GdbShutdown 関数を呼び出して終了処理を行います。この関数を呼び出すことで、簡易データベースライブラリが使用したメモリの解放や内部状態のリセットが行われます。非同期処理中の場合は、DWC_GdbCancelRequest 関数で非同期処理のキャンセル処理を行ってから DWC_GdbShutdown 関数により終了処理を行う必要があります。

## 3.4 ランキングライブラリとの同時利用について

簡易データベースライブラリとランキングライブラリを、同時に利用することはできません。かならずどちらかのライブラリを終了させた上でご利用下さい。

# 4 ガイドライン

## 4.1 容量

DWC_GdbCreateRecordAsync 関数によるデータ格納に関しては、WMS で詳細な仕様を入力し、審査を受ける必要があります。 DWC_GdbUploadFileAsync 関数によるアップロードについても同様に WMS を通じて審査を受ける必要がありますが、1 ユーザ当たり 512kB を超えない範囲で仕様を検討して下さい。

## 4.2 アクセス頻度

WMS で詳細な仕様を入力し、審査を受ける必要がありますが、キャッシュフラグを ON にできるものについては極力 ON にするようにし、また、ユーザのアクションなしに自動でサーバに定期的にアクセスするような使い方は避けて下さい。