Solaris DHCP サービス開発ガイド

Sun Microsystems, Inc.
4150 Network Circle
Santa Clara, CA 95054
U.S.A.

Part No: 816–3970–11
2003 年 4 月

原典: *Solaris DHCP Service Developer's Guide*

Part No: 806-6829-11

Revision A

030227@5533

目次

表目次

図目次

はじめに

『*Solaris DHCP* サービス開発ガイド』は、現在の Solaris™DHCP サービスではサポートされていないデータストレージ機能について説明したものです。このマニュアルには、Solaris DHCP で使用されるデータアクセスフレームワークの概要や開発者向けの一般的な指針の他に、新しいデータ格納をサポートするモジュールの作成に必要な API 関数の説明が含まれています。

対象読者

このマニュアルは、Solaris DHCP サービスで使用できるデータストレージの選択に関心がある Solaris のプログラマを対象としています。

内容の紹介

このマニュアルは、次の章から構成されています。

第 1 章では、DHCP サービスのデータアクセスに使用されるアーキテクチャの概要を説明します。

第 2 章では、このアーキテクチャに求められる要件について説明します。

第 3 章では、ユーザーがエクスポートする API 関数について説明します。

第 4 章では、サンプルコードのテンプレートと、パブリックモジュールのためのコードの作成やデバッグを行う上で役立つ Sun™ の Web サイトの情報が記載されています。

Sun のオンラインマニュアル

docs.sun.com™ では、Sun が提供しているオンラインマニュアルを参照することができます。マニュアルのタイトルや特定の主題などをキーワードとして、検索を行うこともできます。URL は、http://docs.sun.com です。

表記上の規則

このマニュアルでは、次のような字体や記号を特別な意味を持つものとして使用します。

表 **P–1** 表記上の規則

字体または記号	意味	例
AaBbCc123	コマンド名、ファイル名、ディレクトリ名、画面上のコンピュータ出力、コード例を示します。	.login ファイルを編集します。 ls -a を使用してすべてのファイルを表示します。 system%
AaBbCc123	ユーザーが入力する文字を、画面上のコンピュータ出力と区別して示します。	system% **su** password:
AaBbCc123	変数を示します。実際に使用する特定の名前または値で置き換えます。	ファイルを削除するには、rm *filename* と入力します。
『 』	参照する書名を示します。	『コードマネージャ・ユーザーズガイド』を参照してください。
「 」	参照する章、節、ボタンやメニュー名、強調する単語を示します。	第 5 章「衝突の回避」を参照してください。 この操作ができるのは、「スーパーユーザー」だけです。
\	枠で囲まれたコード例で、テキストがページ行幅を超える場合に、継続を示します。	sun% **grep '^#define ** **XV_VERSION_STRING'**

コード例は次のように表示されます。

- C シェル

```
machine_name% command y|n [filename]
```

- C シェルのスーパーユーザー

```
machine_name# command y|n [filename]
```

- Bourne シェルおよび Korn シェル

```
$ command y|n [filename]
```

- Bourne シェルおよび Korn シェルのスーパーユーザー

```
# command y|n [filename]
```

[ ] は省略可能な項目を示します。上記の例は、*filename* は省略してもよいことを示しています。

| は区切り文字 (セパレータ) です。この文字で分割されている引数のうち 1 つだけを指定します。

キーボードのキー名は英文で、頭文字を大文字で示します (例: Shift キーを押します)。ただし、キーボードによっては Enter キーが Return キーの動作をします。

ダッシュ (-) は 2 つのキーを同時に押すことを示します。たとえば、Ctrl-D は Control キーを押したまま D キーを押すことを意味します。

一般規則

- このマニュアルでは、「x86」という用語は、Intel 32 ビット系列のマイクロプロセッサチップ、および AMD が提供する互換マイクロプロセッサチップを意味します。

第 1 章

Solaris DHCP データアクセスアーキテクチャの概要

この章では、Solaris 8 7/01 オペレーティング環境で導入された Solaris Dynamic Host Configuration Protocol (DHCP) サービスのアーキテクチャの概要を説明します。この概要によって、ユーザーの作業がアーキテクチャのどの部分に対応するのかがわかります。

Solaris DHCP サービスの一般情報については、『*Solaris* のシステム管理 *(IP* サービス*)*』の「Solaris DHCP (概要)」を参照してください。

この章は、次の節から構成されます。

モジューラのフレームワーク

Solaris DHCP サービスには、DHCP デーモン、管理ツール、および異なるデータストレージ機能のための別個のデータアクセスモジュール (パブリックモジュールと呼ばれる) が含まれます。Solaris DHCP は、必要なデータストレージ機能をサポートするための、共有オブジェクトとして実装される、ユーザー独自のパブリックモジュールを作成できる API を提供します。パブリックモジュールを Solaris DHCP フレームワークに組み込むと、DHCP サービスがそのデータをこのパブリックモジュールを使ってユーザーのデータベースに格納します。パブリックモジュールは Solaris DHCP サービスから独立して提供することができるため、誰でも任意のデータストレージ機能をサポートするためにモジュールを開発し、配布することができます。

このアーキテクチャを使用する Solaris DHCP の最初リリースでは、ASCII ファイル、NIS+、およびファイルシステムベースのバイナリデータ格納に対するパブリックモジュールが提供されます。このマニュアルでは、開発者が任意のデータベースに対する独自のパブリックモジュールを作成するときに必要となる情報を提供します。

DHCP サーバーのマルチスレッド処理

DHCP サーバーにはマルチスレッド機能が実装されているため、DHCP サーバーでは多数のクライアントに同時にサービスを提供することができます。DHCP サーバーによるマルチスレッド処理をサポートするためには、パブリックモジュールが MT-安全でなければならず、これだけに関して言えば DHCP サービスは多数のクライアントに対応することが可能ということになります。しかし、DHCP サーバーの実際の能力は、主にデータストレージ機能の能力と、データのアクセスに使用されるパブリックモジュールの効率に依存します。したがって、高速かつ大容量のデータストレージ機能を使用するパブリックモジュールを作成することで、Solaris DHCP サービスの性能と能力が向上する可能性があります。

データアクセス層

Solaris DHCP モジューラフレームワークの実装では、次のデータアクセス層が使用されます。

- アプリケーション/サービス層 - この層には、DHCP デーモン (`in.dhcpd`) や、コマンド行管理ユーティリティ (`pntadm`、`dhtadm`、`dhcpconfig`)、 `dhcpmgr`、レポートジェネレータなどの、DHCP サービスのデータを使用するすべての機能が含まれます。これらのデータを使用するものは、このアーキテクチャのフレームワーク構成層によって実装される API 関数の呼び出しを通して DHCP サービスと対話を行います。
- フレームワーク構成層 - 共有ライブラリ `libdhcpsvc.so` と構成ファイル `/etc/inet/dhcpsvc.conf` から構成されています。フレームワーク構成層は、アプリケーション/サービス層とサービスプロバイダ層を接続するものです。フレームワーク構成層についての詳細は、15 ページの「フレームワーク構成層」を参照してください。
- サービスプロバイダ層 - サービスプロバイダ API 関数を実装するパブリックモジュールから構成されています。これは、データ格納コンテナおよびそれらに関するレコードを操作するために、フレームワーク構成層を介してアプリケーション/サービス層によって使用されます。データ格納コンテナとは `dhcptab` および DHCP ネットワークテーブルのことです。

次の図に、個々のアーキテクチャ層の対話形式を示します。

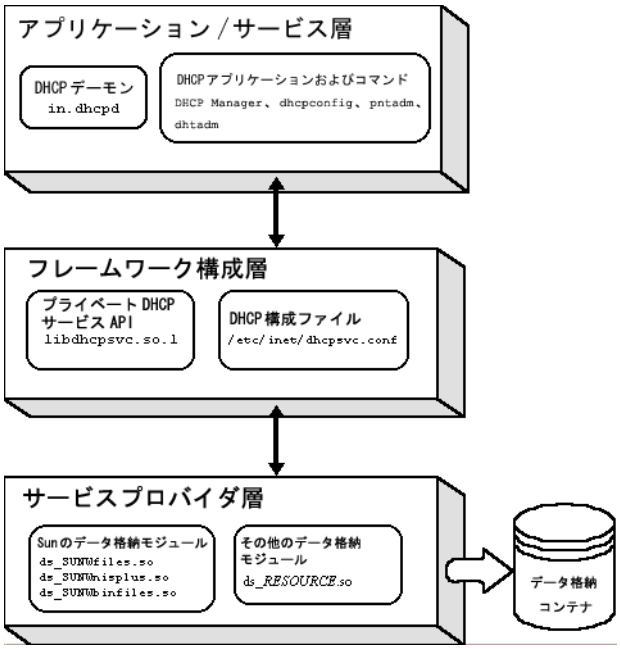

図 1–1 DHCP サービスにおけるデータ格納アクセスのアーキテクチャ

フレームワーク構成層

libdhcpsvc.so で実装される関数がアプリケーション/サービス層に使用され、次のことが行われます。

- パブリックモジュールの検出、ロード、アンロードを行う
- データコンテナのバージョン変更を管理する
- データ格納コンテナにアクセスする
- コンテナ内のデータ格納レコードを操作する

/etc/inet/dhcpsvc.conf には、パブリックモジュールの開発者に関連する次のキーワードを含む、DHCP サービスの構成パラメータが多数含まれています。

RESOURCE　ロードするパブリックモジュール。RESOURCE の値がパブリックモジュールの名前にマッチします。たとえば、RESOURCE=SUNWfiles は、パブリックモジュールが ds_SUNWfiles.so であることを表しています。パブリックモジュールの名前付けの規則は、22 ページの「パブリックモジュールとデータ格納コンテナの名前付け」に記載されています。

PATH
: パブリックモジュールがエクスポートするデータサービス内の DHCP コンテナの位置。PATH の値はそのデータサービス固有のものです。たとえば、SUNWfiles リソースの PATH には UNIX のパス名が割り当てられます。

RESOURCE_CONFIG
: パブリックモジュール固有の構成情報。このオプション (省略可能) のキーワードは、ユーザーからの認証など、データサービスで構成情報を必要とするときに使用できます。このキーワードを使用する場合には、このキーワードの値を設定するための情報をユーザーに要求するパブリックモジュール管理ビーンを提供する必要があります。25 ページの「データサービス構成ツールと DHCP 管理ツール」を参照してください。さらに、このモジュールでは、モジュールのロード時にこのキーワードの値を受け取るための configure() 関数をエクスポートする必要があります。詳細は、30 ページの「configure()」を参照してください。

フレームワーク構成層も、20 ページの「ファイルシステムベースのコンテナへのアクセスの同期化」に記述されているように、オプションの API 同期サービスをサービスプロバイダ層に提供します。

サービスプロバイダ層 API

サービスプロバイダ層 API は、関数、データ構造、および /usr/include/dhcp_svc_public.h ファイルに含まれる表記定数から構成されます。

次の表は関数の要約です。個々の関数の詳細については、それぞれ記載されている節を参照してください。

表 **1–1** サービスプロバイダ層 API 関数

API 関数	用途
すべてのデータ格納コンテナに対する一般的な関数	
30 ページの「configure()」	構成文字列をデータ格納に渡します。オプションの関数です。
30 ページの「mklocation()」	データ格納を置く場所を作成します。
31 ページの「status()」	データ格納の一般的なステータス情報を返します。
31 ページの「version()」	データ格納コンテナに実装されるサービスプロバイダ層 API のバージョンを返します。

表 1–1 サービスプロバイダ層 API 関数　(続き)

API 関数	用途
dhcptab コンテナ用の関数	
32 ページの「list_dt()」	dhcptab コンテナの名前を返します。
33 ページの「open_dt()」	dhcptab コンテナをオープンまたは作成します。
33 ページの「lookup_dt()」	dhcptab コンテナのレコードのクエリー検索を行います。
35 ページの「add_dt()」	dhcptab コンテナにレコードを追加します。
36 ページの「modify_dt()」	dhcptab コンテナ内の既存のレコードを変更します。
36 ページの「delete_dt()」	dhcptab コンテナからレコードを削除します。
37 ページの「close_dt()」	dhcptab コンテナをクローズします。
37 ページの「remove_dt()」	データ格納から dhcptab コンテナを削除します。
DHCP ネットワークコンテナ用の関数	
38 ページの「list_dn()」	DHCP ネットワークコンテナ名のリストを返します。
39 ページの「open_dn()」	DHCP ネットワークコンテナをオープンまたは作成します。
39 ページの「lookup_dn()」	DHCP ネットワークコンテナ内のレコードのクエリー検索を行います。
40 ページの「add_dn()」	DHCP ネットワークコンテナにレコードを追加します。
41 ページの「modify_dn()」	DHCP ネットワークコンテナ内の既存のレコードを変更します。
41 ページの「delete_dn()」	DHCP ネットワークコンテナからレコードを削除します。
42 ページの「close_dn()」	DHCP ネットワークコンテナをクローズします。
42 ページの「remove_dn()」	データ格納から DHCP ネットワークコンテナを削除します。

データ格納コンテナ

dhcptab および DHCP ネットワークテーブルは、総称的にデータ格納コンテナと呼ばれます。Solaris DHCP では、デフォルトで、次の表に示すコンテナの形式がサポートされます。

サポートされるデータサービス	パブリックモジュール
ファイルシステムベースの ASCII 形式	`ds_SUNWfiles.so`
NIS+ サービス	`ds_SUNWnisplus.so`
ファイルシステムベースのバイナリ形式	`ds_SUNWbinfiles.so`

第 2 章

モジュール作成者のためのアーキテクチャの機能

この章では、データサービス用にパブリックモジュールを作成するときに留意すべきアーキテクチャ上の詳細について説明します。

この章は、次の節から構成されます。

関数のカテゴリ

サービスプロバイダ層の API 関数は 3 つのカテゴリに分類されます。

- データ格納関数 - パブリックモジュールとその基盤となるデータサービス自体に関連する処理を行います。これらの関数には、`configure()`、`mklocation()`、`status()`、`version()` などが含まれます。
- `dhcptab` コンテナ関数 - `dhcptab` コンテナを作成したり、`dhcptab` コンテナに対してレコードを記載したり、`dhcptab` コンテナ内のレコードのクエリー検索を行なったりします。`open_dt()` 関数はコンテナのハンドルを作成し、その他の関数はそのハンドルへのポインタをとります。`close_dt()` 関数は、そのコンテナをクローズするときにハンドルを破棄します。
- ネットワークコンテナ関数 - DHCP ネットワークコンテナを作成したり、ネットワークコンテナにレコードを追加したり、ネットワークコンテナ内のレコードをクエリー検索したりします。`open_dn()` 関数ではコンテナのハンドルを作成し、他の関数ではそのハンドルへのポインタを使用します。`close_dn()` 関数では、コ

ンテナをクローズするときにハンドルを破棄します。

これらの関数の詳細は、第 3 章を参照してください。

マルチスレッド処理の考慮点

DHCP サーバーはマルチスレッド機能を実装しており、多数のクライアントを同時にサポートすることが可能です。DHCP サーバーによるマルチスレッド処理をサポートするためには、パブリックモジュールが MT-安全でなければなりません。

モジュールがMT-安全であるためには、 `add_d?()`、`delete_d?()`、`modify_d?()` への呼び出しを同期させ、それらがシリアルに呼び出されるようにする必要があります。 たとえば、あるスレッドが、ある DHCP ネットワークコンテナについて `add_dn()` 内に存在する場合は、他のスレッドが、その同じコンテナについて `add_dn()`、`delete_dn()`、`modify_dn()`、`lookup_dn()` 内に存在しません。パブリックモジュールがローカルファイルシステムベースのデータサービスをサポートする場合は、これに対処させるために同期サービスを使用することができます。詳細は、20 ページの「ファイルシステムベースのコンテナへのアクセスの同期化」を参照してください。

ファイルシステムベースのコンテナへのアクセスの同期化

ローカルのファイルシステムベースのデータサービス (データサービスが DHCP サーバーと同じマシンで動作する場合) 内のコンテナにアクセスするパブリックモジュールを作成する場合には、複数のプロセスやスレッドの間で基盤となるデータサービスへのアクセスを同期化するのは難しいことがあります。

Solaris DHCP 同期サービスでは、同期化をフレームワーク構成層へプッシュアップすることにより、ローカルのファイルシステムベースのデータサービスを使用するパブリックモジュールの設計をシンプルにします。このフレームワークを使用するようにモジュールを設計すれば、コードがシンプルになり、設計がよりクリアになります。

この同期サービスは、パブリックモジュールに、 `add_d?()`、`delete_d?()`、`modify_d?()` サービスプロバイダ層 API 呼び出しのすべての呼び出し元のコンテナ単位の排他的同期を提供します。つまり、あるスレッドが、ある DHCP ネットワークコンテナについて `add_dn()` 内に存在する場合は、他のスレッドが、その同じコンテナについて `add_dn()`、`delete_dn()`、`modify_dn()`、`lookup_dn()` 内に存在しません。ただし、`close_dn()` のように同期が保証されないルーチン内には、他のスレッドが存在することがあります。

lookup_d? () のすべての呼び出し元は、コンテナ単位で共有的に同期化され, 提供されます。したがって、多数のスレッドが同じコンテナについて検索を行なっていても、追加、削除、変更の操作は、同時に 1 つのスレッドしか実行できません。

同期サービスは、デーモン (/usr/lib/inet/dsvclockd) として実装されます。パブリックモジュールに代わってなされるロックマネージャの要求は、フレームワーク構成層 API 呼び出しを通して行われます。フレームワーク構成層とロックマネージャデーモン間のインタフェースには、Solaris の door プロセス間通信のメカニズムが使用されます。これについては、例として door_create(3DOOR) や door_call(3DOOR) などを参照してください。

パブリックモジュールが同期化を要求すると、フレームワーク構成層は、dsvclockd デーモンがまだ動作していなければ、これを起動します。デーモンがロックを扱わない時間が 15 分を過ぎると、デーモンは自動的に終了します。この時間を変更する場合は、/etc/default/dsvclockd ファイルを作成し、IDLE のデフォルトをデーモンが終了するまでにアイドルである分数に設定します。

同期サービスが必要なことをパブリックモジュールからフレームワーク構成層に知らせるには、モジュールのソースファイルの 1 つに次のグローバル変数を含めます。

```
dsvc_synchtype_t dsvc_synchtype = DSVC_SYNCH_DSVCD;
```

同期サービスが「必要ない」ことをパブリックモジュールからフレームワーク構成層に知らせるには、モジュールのソースファイルの 1 つに次のグローバル変数を含めます。

```
dsvc_synchtype_t dsvc_synchtype = DSVC_SYNCH_NONE;
```

現在のところ、同期タイプは DSVC_SYNCH_DSVCD と DSVC_SYNCH_NONE の 2 つだけです。

更新における衝突の回避

アーキテクチャは、ファイルベースのモジュールがレコード更新における衝突を回避するのを助ける機能を提供します。サービスプロバイダ API では、レコード単位の更新シグニチャ (符号なしの 64 ビット整数) を使用することによって、データの整合性を維持します。この更新シグニチャは、/usr/include/dhcp_svc_public.h で定義された、d?_rec_t コンテナレコードデータ構造の d?_sig 要素です。アプリケーション/サービス層からフレームワーク構成層 API、およびサービスプロバイダ層 API に至るまで、このアーキテクチャのすべての層が d?_rec_t を使用します。サービスプロバイダ層より上では、更新シグニチャは、フレームワーク構成層 API のユーザーからは操作されない不透明なオブジェクトです。

モジュールは、サービスプロバイダ層 API 関数呼び出しを通して d?_rec_t レコードを受け取ったら、d?_rec_t のキーフィールドとマッチするレコードをデータサービス内で検索し、内部レコードのシグニチャと呼び出しから渡された d?_rec_t を比

較する必要があります。内部レコードのシグニチャが、渡されたレコードのものとマッチしない場合、レコードは、その使用者がパブリックモジュールから取得した後で変更されていることを意味します。この場合には、モジュールは、レコードが前に取得されたときから変更されていることを呼び出し元に知らせる DSVC_COLLISION を返す必要があります。シグニチャがマッチする場合は、モジュールは、そのレコードを格納する前に引数レコードの更新シグニチャを増分する必要があります。

モジュールがサービスプロバイダ層 API を通して新しい d?_rec_t レコードを受け取ったら、そのモジュールはそれをコンテナに追加する前に、更新シグニチャに値を設定する必要があります。最もシンプルな方法は、値に 1 を設定する方法です。

ただし、まれなケースとはいえ、シグニチャが常に同じ初期値を持つと、更新における衝突を検出できないことがあります。たとえば、次のような場合です。まず、スレッド A がシグニチャ 1 のレコードを追加し、スレッド B がそのレコードを検索します。次に、スレッド A がそのレコードを削除し、同じキーフィールドでシグニチャ 1 の新しいレコードを作成します。その後スレッド B が、検索したレコードを変更しますが、それはすでに削除されています。モジュールは、スレッド B が検索したレコードのキーフィールドとシグニチャをデータ格納内のレコードと比較し、両者が同じなので変更を行います。しかし、これらのレコードは実際は同じものではないので、このような変更の試行は、本来は衝突とされるべきものです。

Solaris DHCP で提供される ds_SUNWfiles.so と ds_SUNWbinfiles.so モジュールは、こういった可能性を指摘します。これらのモジュールは、各レコードのシグニチャの一意性を確実にするために更新シグニチャを 2 つのフィールドに分割します。更新シグニチャの最初の 16 ビットフィールドに、ランダムに生成した数値を設定します。このフィールドは、設定後、レコード内で変更されません。シグニチャの下位 48 ビットフィールドには 1 が設定され、レコードが更新されるたびに 1 つずつ増分されます。

注 – Solaris DHCP で提供されるモジュールは、レコードの更新における衝突を避けるためのアプローチの 1 つです。ユーザーは、これと似た方法を使用したり、独自の方法を考案することができます。

パブリックモジュールとデータ格納コンテナの名前付け

パブリックモジュールおよびコンテナはどちらも、アーキテクチャのアップグレードメカニズムが機能するためのバージョン番号を含んでいなければなりません。

パブリックモジュール名

パブリックモジュールの名前は、次の形式に従っていなければなりません。

ds_*name*.so.*ver*

ここで、*name* はモジュールの名前、*ver* はコンテナ形式のバージョン番号です。*name* は、ユーザーの組織に対応する、国際的に認知された識別子である接頭辞を使用する必要があります。たとえば、Sun Microsystems が提供するパブリックモジュールには、Sun の株式のティッカーシンボルである SUNW の接頭辞が付いていなくてはなりません。たとえば、NIS+ のパブリックモジュール名は `ds_SUNWnisplus.so.1` となります。モジュール名にこのような識別子を含めることにより、パブリックモジュールディレクトリ `/usr/lib/inet/dhcp/svc` 内でパブリックモジュール名の衝突を避けることができます。

たとえば、会社名が Inet DataBase なら、モジュール名に `ds_IDBtrees.so.1` を付けることができます。

コンテナ名

管理インタフェースを通して管理者に表示されるコンテナの名前には、常に `dhcptab` と、DHCP ネットワークテーブルのドット形式の IP ネットワークアドレス (`10.0.0.0` など) を含まなければなりません。

内部的に、データ格納コンテナの名前には、必要に応じてコンテナ形式の改訂番号を生成できるようにバージョン番号を含む必要があります。この名前付けのスキームでは、1 つのコンテナの複数のバージョンが同時に存在することが可能で、これはアーキテクチャのコンテナのバージョンアップグレードメカニズムが機能するために必要です。

コンテナに使用される名前は、その名前が一意であるために、国際的に認識できるトークンを含む必要があります。

たとえば、Solaris DHCP で提供される NIS+ のパブリックモジュールは、`dhcptab` コンテナを内部的には `SUNWnisplus1_dhcptab` として作成します。172.21.174.0 ネットワークテーブルのコンテナは `SUNWnisplus1_172.21.174.0` になります。

会社名が Inet DataBase で、パブリックモジュールが `ds_IDBtrees.so.1` であれば、コンテナに `IDBtrees1_dhcptab` および `IDBtrees1_172.21.174.0` と名付けることができます。

コンテナレコードの形式

Solaris DHCP サービスでは、DHCP コンテナのタイプとして dhcptab コンテナと DHCP ネットワークコンテナの 2 つが使用されます。

dhcptab コンテナには、dhcptab のマニュアルページに記述されている、DHCP 構成データが格納されています。DHCP サービスでは、dhcptab コンテナの 1 つのインスタンスのみが維持されます。

dhcptab レコードは、フレームワーク構成層とサービスプロバイダ層の間で内部構造 dt_rec_t によって渡されます。インクルードファイル /usr/include/dhcp_svc_public.h がこの構造を示します。

パブリックモジュールは、重複した dhcptab レコードが無いことを確実にしなければなりません。2 つのレコードが、同じキーフィールド値を持ってはなりません。

DHCP ネットワークコンテナには、dhcp_network のマニュアルページに記載されているように、複数の IP アドレスレコードが含まれます。これらのコンテナには、データ格納と、これらの IP アドレスが属しているネットワークのドット形式の IP アドレス (10.0.0.0 など) を示す名前が付けられます。DHCP ネットワークコンテナはいくつでも存在できますが、DHCP サービスによってサポートされる各ネットワークに対しては 1 つです。

DHCP ネットワークレコードは、フレームワーク構成層とサービスプロバイダ層の間で内部構造 dn_rec_t によって渡されます。インクルードファイル /usr/include/dhcp_svc_public.h がこの構造を示します。

パブリックモジュールは、重複したネットワークコンテナレコードが無いことを確実にしなければなりません。2 つのレコードが、同じキーフィールド値を持ってはなりません。

データ格納構成データの受け渡し

Solaris DHCP データアクセスアーキテクチャは、データ格納固有の構成データをパブリックモジュールに (従って、データ格納に) 渡すためのオプションの機能を提供します。この機能は、ASCII 文字列として実装されます。ASCII 文字列は、DHCP サービス管理インタフェース (dhcpconfig または dhcpmgr) を介してフレームワーク構成層に渡され、DHCP サーバーマシンに格納されます。詳細は、dhcpsvc.conf(4) のマニュアルページを参照してください。この文字列でどのような情報を渡すのかは、ユーザーが決定し、DHCP 管理者は管理ツールを介してこの文字列の値を提供します。この文字列は、たとえば、データベースへのログインに必要なユーザー名とパスワードなどを含むことができます。

DHCP 管理者から情報を得るには、適切なダイアログを表示するために JavaBeans™ コンポーネントを作成する必要があります。この情報は、その後単一の ASCII 文字列として管理インタフェースに渡されます。この ASCII 文字列トークンの形式は、デバッグを容易にするために文書化されていなければなりません。この機能をサポートするためのパブリックモジュールは 第 3 章で記述される `configure()` 関数を実装し、エクスポートする必要があります。

注 – このアーキテクチャでは、ASCII 文字列は暗号化されません。クリアテキストで `/etc/inet/dhcpsvc.conf` ファイルに保管されます。暗号化された情報が必要な場合は、フレームワーク構成層に渡す前にビーンは情報を暗号化する必要があります。

コンテナバージョンのアップグレード

コンテナバージョンのアップグレードにユーザーが関与する必要はありません。このアーキテクチャでは、22 ページの「パブリックモジュールとデータ格納コンテナの名前付け」のガイドラインに従っている限り、異なるコンテナバージョンの共存を確立しているからです。管理ツールはアーキテクチャのこの機能を使用して、DHCP 管理者がコンテナのバージョンアップグレードを自動的に行えるようにします。

コンテナ形式のバージョンは、インストールによって (Solaris DHCP をアップグレードするとき)、または最初の DHCP サービス構成時の管理インタフェースを通して、フレームワーク構成層の構成ファイルに自動的に設定されます。新しいコンテナバージョンを含むパブリックモジュールの新しいバージョンをインストールしようとすると、新しいバージョンを管理インタフェースが自動的に検出し、パブリックモジュールのバージョンをアップグレードするかどうかを管理者にたずねます。アップグレードは延期することもできます。DHCP サービスは、管理者がパブリックモジュールをアップグレードするまで、元のバージョンのパブリックモジュールでの動作を継続します。

データサービス構成ツールと DHCP 管理ツール

DHCP サービスの構成機能は、`dhcpmgr` および `dhcpconfig` 構成ツールによりシステム管理者に提供されます。モジュールをツールのユーザーから使用できるようにし、基盤となるデータサービスの構成をユーザーが行えるようにしたい場合は、ビーンと呼ばれる JavaBeans™ コンポーネントをパブリックモジュールに対して提供する必要があります。

このビーンは、dhcpsvc.conf 内の PATH 変数と、オプションとして RESOURCE_CONFIG 変数を設定するために必要なコンテキストをパブリックモジュールに提供します。

パブリックモジュール管理ビーンの API 関数

dhcpmgr ツールは、com/sun/dhcpmgr/client/DSModule というインタフェースを提供します。このインタフェースは、パブリックモジュール管理ビーンが実装する必要のある API 関数を定義します。

DSModule インタフェースは、dhcpmgr.jar ファイルに含まれています。このインタフェースに対応したビーンをコンパイルするためには、/usr/sadm/admin/dhcpmgr/dhcpmgr.jar を javac のクラスパスに追加する必要があります。たとえば、myModule.java という名前のビーンには、次のように入力します。

```
javac -classpath /usr/sadm/admin/dhcpmgr/dhcpmgr.jar
myModule.java
```

getComponent()

形式

```
abstract java.awt.Component getComponent()
```

説明

DHCP 構成ウィザードのウィザードステップの 1 つとして表示されるコンポーネントを返します。返されるコンポーネントは、構成時にユーザーから提供されたデータ格納固有のデータの取得に使用される GUI コンポーネントが含まれているパネルです。構成データそのものは、getPath() および getAdditional() メソッドへの呼び出しの結果としてウィザードに返されます。詳細は、27 ページの「getPath()」と 27 ページの「getAdditional()」を参照してください。

getDescription()

形式

```
abstract java.lang.String getDescription()
```

説明

DHCP 構成ウィザードがデータ格納選択のリストにデータ格納を追加するときに使用する説明文を返します。たとえば、ds_SUNWfiles.so パブリックモジュールの管理ビーンは、説明として「Text files」を返します。

getPath()

形式

```
abstract java.lang.String getPath()
```

説明

データ格納によって使用されるパス/位置 (フレームワーク構成層の構成ファイル /etc/inet/dhcpsvc.conf 内の PATH の値)、または設定されていない場合は NULL を返します。パス/位置の値は、管理ビーンのコンポーネントとの対話の中でユーザーが提供する必要があります。24 ページの「データ格納構成データの受け渡し」を参照してください。

getAdditional()

形式

```
abstract java.lang.String getAdditional()
```

説明

フレームワーク構成層の構成ファイル /etc/inet/dhcpsvc.conf 内のその他のデータ格納固有の情報 (RESOURCE_CONFIG などの情報) を返します。このメソッドから返される値は、ほとんど、管理ビーンのコンポーネントとの対話の中でユーザーが提供するものです。24 ページの「データ格納構成データの受け渡し」を参照してください。

パブリックモジュール管理ビーンのパッケージングの要件

パブリックモジュールの管理ビーンは、次のパッケージングの用件に適合している必要があります。

- パブリックモジュールの管理ビーンは、JAR ファイルとしてアーカイブされなければなりません。JAR ファイルの名前は、パブリックモジュールの名前と .jar 接尾辞から構成されていなければなりません。たとえば、ds_SUNWfiles.so パブリックモジュール内のパブリックモジュール管理ビーンの名前は SUNWfiles.jar になります。
- JAR ファイルには、ビーンのクラスを識別する指標が含まれていなければなりません。たとえば、SUNWfiles.jar という JAR ファイルの指標には、次の行が含まれます。

ネーム com/sun/dhcpmgr/client/SUNWfiles/SUNWfiles.class

Java-Bean: True

com/sun/dhcpmgr/client/SUNWfiles/SUNWfiles.class クラスは、com/sun/dhcpmgr/client/DSModule インタフェースを実装する Java™ クラスです。

第 3 章

サービスプロバイダ層の API

この章では、パブリックモジュールによってエクスポートされ、フレームワーク構成層によって使用される API 関数をリストし、説明します。関数は目的によってセクションに分類されています。各セクション内では、関数は使用が予想される順にリストされています。

この章は、次の節から構成されます。

特定バージョンのサービスプロバイダ層の API に基づくすべての実装は、実装する API 関数についてのこの仕様に準拠しなければなりません。後のバージョンの API は、前のバージョンの API との後方互換性がなければなりません。したがって、新たに API 呼び出しを追加することはできますが、既存のものを変更したり、削除してはなりません。

関数についての詳細は、インクルードファイル `/usr/include/dhcp_svc_public.h` を参照してください。

一般的なデータ格納関数

この節では、一般的なデータ格納活動に関する関数について説明します。

configure()

目的

構成文字列をデータ格納に渡します。

形式

```
int configure(const char *configp);
```

説明

configure() 関数はオプション (省略可能) です。必要とされるパブリックモジュール管理ビーンとともにこの関数が提供されている場合 (25 ページの「データサービス構成ツールと DHCP 管理ツール」を参照)、フレームワーク構成層は、パブリックモジュールのロード時にこの関数を呼び出し、パブリックモジュール固有の構成文字列内に渡します。この文字列は、データ格納モジュールのために DHCP サーバー上のフレームワーク構成層によってキャッシュされます。

戻り値

DSVC_SUCCESS, DSVC_MODULE_CFG_ERR

configure() 関数は、モジュールがモジュールのロードをフレームワーク構成層に続けてもらいたければ DSVC_SUCCESS を返し、モジュールのロードを行なってもらいたくなければ DSVC_MODULE_CFG_ERR を返します。後者の例は、構成文字列の形式が適切でないためにモジュールの必要な構成を行えないような場合です。

mklocation()

目的

データ格納コンテナを置くディレクトリを作成します。

形式

```
int mklocation(const char *location);
```

説明

データ格納コンテナを置く場所として、location で指定されたディレクトリを作成します (ディレクトリが存在しない場合)。

戻り値

DSVC_SUCCESS, DSVC_ACCESS, DSVC_EXISTS, DSVC_BUSY, DSVC_INTERNAL, DSVC_UNSUPPORTED

status()

目的

データ格納の一般的な状態を取得します。

形式

```
int status(const char *location);
```

説明

status() 関数は、データ格納に対してその一般的な状態を返すよう指示します。もし location が NULL でなければ、そのコンテナが実際に存在し、アクセス可能であり、かつ形式がそのデータ格納タイプにとって正しいかどうかを判定することによって、データ格納コンテナのロケーションを検証します。データ格納は、必要な機能が利用できなかったり、準備されていない場合には、適切なエラーコードを返す必要があります。

戻り値

DSVC_SUCCESS, DSVC_ACCESS, DSVC_NO_LOCATION, DSVC_BUSY, DSVC_INTERNAL .

version()

目的

データ格納によって実装される API のバージョン番号を取得します。

形式

```
int version(int *versionp);
```

説明

このマニュアルに記載されているサービスプロバイダ層 API をサポートするデータ格納はバージョン 1 です。バージョンは、int 内で versionp で示して返されます。

戻り値

DSVC_SUCCESS, DSVC_INTERNAL, DSVC_MODULE_ERR

dhcptab 関数

この節に記載される API 関数は、dhcptab コンテナとともに使用されます。

list_dt()

目的

dhcptab コンテナの名前をリストします。

形式

```
int list_dt(const char *location, char ***listppp, uint_t *count)
;
```

説明

location で見つかった dhcptab コンテナオブジェクトの動的に割り当てたリストを生成し (listppp)、リストの項目数を count 内に格納します。dhcptab コンテナオブジェクトが全く存在しない場合は、DSVC_SUCCESS が返され、listppp は NULL に設定され、count は 0 に設定されます。

戻り値

DSVC_SUCCESS, DSVC_ACCESS, DSVC_NO_LOCATION

open_dt()

目的

dhcptab コンテナをオープンするか、または新たに作成します。

形式

```
int open_dt(void **handpp, const char *location, uint_t flags);
```

説明

既存の dhcptab コンテナをオープンするか、または新しいコンテナを location に作成し、そのインスタンスハンドルを指すように handpp を初期設定します。データ格納にとって必要な初期設定があれば、それを行います。新しい dhcptab を作成する場合には、所有者/アクセス権に呼び出し元の ID が使用されます。有効なフラグには、DSVC_CREATE、DSVC_READ、DSVC_WRITE、DSVC_NONBLOCK があります。dhcptab コンテナの読み取り専用 (DSVC_CREATE | DSVC_READ) としての作成は無効であることに注意してください。

戻り値

```
DSVC_SUCCESS, DSVC_EXISTS, DSVC_ACCESS, DSVC_NOENT,
DSVC_NO_LOCATION, DSVC_BUSY, DSVC_INTERNAL
```

lookup_dt()

目的

dhcptab コンテナ内のレコードに対するクエリー検索を行います。

形式

```
int lookup_dt(void *handp, boolean_t partial, uint_t query, int
count, const dt_rec_t *targetp, dt_rec_list_t **resultp, uint_t
*records);
```

説明

query と targetp の組み合わせで指定されたクエリーにマッチするインスタンスを dhcptab コンテナから検索します。partial 引数が B_TRUE の場合は、呼び出し元にとって部分的なクエリー結果も適用可能であることを意味します。したがって、

partial が B_TRUE の場合は、マッチするレコードが 1 つでもあれば、クエリーは有効と見なされます。partial が B_FALSE の場合は、クエリーがコンテナ全体に適用された場合だけ DSVC_SUCCESS が返されます。

クエリー引数は、長さがそれぞれ 16 ビットの 2 つのフィールドからなります。下位 16 ビットでは、targetp のどのフィールド {key, type} をクエリーの対象とするかを選択します。上位 16 ビットでは、下位 16 ビットで選択された特定のフィールド値がマッチするものを検索するのか (ビットセット)、マッチしないものを検索するのか (ビットクリア) を指定します。両方の 16 ビットフィールドとも、ビット 2 から 15 は現在のところ使用されておらず、0 に設定されていなければなりません。クエリーを構築するために有用なマクロを例 3–1 に示します。

count フィールドは、マッチするレコードを最大でいくつ返すかを指定します。count 値に -1 を指定すると、マッチするレコードがいくつあっても、すべて返すことを要求します。count 値に 0 を指定すると、lookup_dt はデータ無しでただちに返されます。

resultp は、返されるレコードのリストを指すよう設定されます。resultp に NULL が指定される場合、呼び出し元は単にクエリーにマッチするレコードがいくつあるかに関心があるということになります。これらのレコードは動的に割り当てられるため、呼び出し元でこれを解放する必要があることに注意してください。lookup_dt() は、records 引数内にマッチするレコードの数を返します。records の値 0 は、クエリーにマッチするレコードは全く無いことを示します。

次の例には、DHCP ネットワークおよび dhcptab のコンテナに対するクエリー検索の構築や操作に有用なマクロが含まれています。

例 **3–1** クエリー検索に有用なマクロ

```
/*
* Query macros - used for initializing query fields (lookup_d?)
*/
/* dhcp network container */
#define DN_QCID 0x0001
#define DN_QCIP 0x0002
#define DN_QSIP 0x0004
#define DN_QLEASE 0x0008
#define DN_QMACRO 0x0010
#define DN_QFDYNAMIC 0x0020
#define DN_QFAUTOMATIC 0x0040
#define DN_QFMANUAL 0x0080
#define DN_QFUNUSABLE 0x0100
#define DN_QFBOOTP_ONLY 0x0200
#define DN_QALL (DN_QCID | DN_QCIP | DN_QSIP | DN_QLEASE | \
DN_QMACRO | DN_QFDYNAMIC DN_QFAUTOMATIC |\
DN_QFMANUAL | DN_QFUNUSABLE | \
DN_QFBOOTP_ONLY)

/* dhcptab */
#define DT_DHCPTAB "dhcptab"  /* default name of container */
#define DT_QKEY 0x01
#define DT_QTYPE 0x02
```

例 **3–1** クエリー検索に有用なマクロ　*(続き)*

```
#define DT_QALL (DT_QKEY | DT_QTYPE)

/* general query macros */
#define DSVC_QINIT(q) ((q) = 0)
#define DSVC_QEQ(q, v) ((q) = ((q) | (v) | ((v) << 16)))
#define DSVC_QNEQ(q, v) ((q) = ((~(v << 16)) & (q)) | (v)))
#define DSVC_QISEQ(q, v) (((q) & (v)) && ((q) & ((v) << 16)))
#define DSVC_QISNEQ(q, v) (((q) & (v)) && (!((q) & ((v) << 16))))

/* Examples */
uint_t query;
/* search for dhcptab record with key value, but not flags value */
DSVC_QINIT(query);
DSVC_QEQ(query, DT_QKEY);
DSVC_QNEQ(query, DT_QTYPE);
/* search for dhcp network record that matches cid, client ip, server ip.
*/
DSVC_QINIT(query);
DSVC_QEQ(query, (DN_QCID | DN_QCIP | DN_QSIP));
```

戻り値

DSVC_SUCCESS, DSVC_ACCESS, DSVC_BUSY, DSVC_INTERNAL.

add_dt()

目的

dhcptab コンテナにレコードを追加します。

形式

```
int add_dt(void *handp, dt_rec_t *newp);
```

説明

handp で参照される dhcptab コンテナにレコード newp を追加します。newp に対応するシグニチャは、基盤となるパブリックモジュールによって更新されます。更新における衝突が発生する場合は、データ格納は更新されません。動的に割り当てられた引数は、呼び出し元で解放する必要があります。

戻り値

DSVC_SUCCESS, DSVC_ACCESS, DSVC_BUSY, DSVC_INTERNAL, DSVC_EXISTS
.

modify_dt()

目的

dhcptab コンテナ内のレコードを変更します。

形式

```
int modify_dt(void *handp, const dt_rec_t *origp, dt_rec_t *newp)
;
```

説明

handp で参照される dhcptab コンテナ内のレコード origp をレコード newp に変更します。newp に対応するシグニチャは、基盤となるパブリックモジュールによって更新されます。更新における衝突が発生する場合は、データ格納は更新されません。動的に割り当てられた引数は、呼び出し元で解放する必要があります。

戻り値

DSVC_SUCCESS, DSVC_ACCESS, DSVC_BUSY, DSVC_COLLISION,
DSVC_INTERNAL, DSVC_NOENT.

delete_dt()

目的

dhcptab コンテナからレコードを削除します。

形式

```
int delete_dt(void *handp, const dt_rec_t *dtp);
```

説明

ハンドル handp で参照される dhcptab コンテナから、dtp の key、type、dt_sig フィールドで指定されるレコードを削除します。更新における衝突が発生する場合には、マッチするレコードはデータ格納から削除されず、DSVC_COLLISION が返されます。動的に割り当てられた引数は、呼び出し元で解放する必要があります。

dtp シグニチャ (dt_sig) に 0 を指定すると、更新における衝突があるかどうかの検知は行なわれず、マッチするレコードが単純に削除されます。

戻り値

DSVC_SUCCESS, DSVC_ACCESS, DSVC_NOENT, DSVC_BUSY, DSVC_INTERNAL, DSVC_COLLISION.

close_dt()

目的

dhcptab コンテナをクローズします。

形式

```
int close_dt(void **handpp);
```

説明

インスタンスハンドルを解放し、インスタンスごとの状態を消去します。

戻り値

DSVC_SUCCESS, DSVC_ACCESS, DSVC_INTERNAL.

remove_dt()

目的

データ格納のロケーションから dhcptab コンテナを削除します。

形式

```
int remove_dt(const char *location);
```

説明

データ格納から location 内の dhcptab コンテナを削除します。

戻り値

DSVC_SUCCESS, DSVC_ACCESS, DSVC_NOENT, DSVC_NO_LOCATION, DSVC_BUSY, DSVC_INTERNAL.

DHCP ネットワークコンテナ関数

この節で説明する API 関数は、DHCP ネットワークコンテナやその中の IP アドレスレコードを処理するときに使用します。

list_dn()

目的

ネットワークコンテナのリストを返します。

形式

```
int list_dn(const char *location, char ***listppp, uint_t *count)
;
```

説明

location 内で見つかったネットワークコンテナオブジェクトの動的に割り当てられたリストを生成し (listppp) 、リストの項目数を count 内に格納します。ネットワークコンテナオブジェクトが全く存在しない場合は、DSVC_SUCCESS が返され、listppp は NULL に設定され、count は 0 に設定されます。

戻り値

DSVC_SUCCESS, DSVC_ACCESS, DSVC_NO_LOCATION

open_dn()

目的

ネットワークコンテナをオープンするか、または新たに作成します。

形式

```
int open_dn(void **handpp, const char *location, uint_t flags,
const struct in_addr *netp, const struct in_addr *maskp);
```

説明

既存の DHCP ネットワークコンテナをオープンするか、または netp および maskp (両方ともホスト順) で指定された新しいコンテナを location 内に作成し、そのインスタンスハンドルを指すように handpp を初期設定します。データ格納にとって必要な初期設定があれば、それを行います。新しい DHCP ネットワークコンテナを作成する場合には、所有者/アクセス権に呼び出し元の ID が使用されます。有効なフラグには、DSVC_CREATE、DSVC_READ、DSVC_WRITE、DSVC_NONBLOCK が含まれます。DHCP ネットワークコンテナの読み取り専用 (DSVC_CREATE | DSVC_READ) としての作成は無効であることに注意してください。

戻り値

DSVC_SUCCESS, DSVC_EXISTS, DSVC_ACCESS, DSVC_NOENT, DSVC_NO_LOCATION, DSVC_BUSY, DSVC_INTERNAL, DSVC_UNSUPPORTED.

lookup_dn()

目的

DHCP ネットワークコンテナ内のレコードに対するクエリー検索を行います。

形式

```
int lookup_dn(void *handp, boolean_t partial, uint_t query, int
count, const dn_rec_t *targetp, dn_rec_list_t **resultp, uint_t
*records);
```

説明

query と targetp の組み合わせで指定されたクエリーにマッチするインスタンスを DHCP ネットワークコンテナから検索します。partial 引数が B_TRUE の場合は、呼び出し元にとって部分的なクエリー結果も適用可能であることを意味します。したがって、partial が B_TRUE の場合は、マッチするレコードが 1 つでもあれば、クエリーは有効と見なされます。partial が B_FALSE の場合は、クエリーがコンテナ全体に適用された場合だけ DSVC_SUCCESS が返されます。

クエリー引数は、長さがそれぞれ 16 ビットの 2 つのフィールドからなります。下位 16 ビットでは、targetp のどのフィールド {クライアント id、フラグ、クライアント IP、サーバー IP、有効期限、マクロ、またはコメント} をクエリーの対象とするかを選択します。上位 16 ビットでは、下位 16 ビットで選択された特定のフィールド値がマッチするものを検索するのか (ビットセット)、マッチしないものを検索するのか (ビットクリア) を指定します。両方の 16 ビットフィールドとも、ビット 7 から 15 は現在のところ使用されておらず、0 に設定されていなければなりません。クエリーを構築するために有用なマクロを例 3–1 に示します。

count フィールドは、マッチするレコードを最大でいくつ返すかを指定します。count 値に -1 を指定すると、マッチするレコードがいくつあっても、すべて返すことを要求します。count 値に 0 を指定すると、lookup_dn データ無しでただちに返されます。

resultp は、返されるレコードのリストを指すよう設定されます。resultp に NULL が指定される場合、呼び出し元は単にクエリーにマッチするレコードがいくつあるかに関心があるということになります。これらのレコードは動的に割り当てられるため、呼び出し元でこれを解放する必要があることに注意してください。lookup_dn() は、records 引数内にマッチするレコードの数を返します。records の値 0 は、クエリーにマッチするレコードは全く無いことを示します。

戻り値

DSVC_SUCCESS, DSVC_ACCESS, DSVC_BUSY, DSVC_INTERNAL.

add_dn()

目的

DHCP ネットワークコンテナにレコードを追加します。

形式

```
int add_dn(void *handp, dn_rec_t *newp);
```

説明

ハンドル handp で参照される DHCP ネットワークコンテナにレコード newp を追加します。newp に対応するシグニチャは、基盤となるパブリックモジュールによって更新されます。更新における衝突が発生する場合は、データ格納は更新されません。

戻り値

DSVC_SUCCESS, DSVC_ACCESS, DSVC_BUSY, DSVC_INTERNAL, DSVC_EXISTS
.

modify_dn()

目的

DHCP ネットワークコンテナ内のレコードを変更します。

形式

```
int modify_dn(void *handp, const dn_rec_t *origp, dn_rec_t *newp)
;
```

説明

ハンドル handp で参照される DHCP ネットワークコンテナ内のレコード origp をレコード newp に変更します。newp に対応するシグニチャは、基盤となるパブリックモジュールによって更新されます。更新における衝突が発生する場合には、データ格納は更新されません。

戻り値

DSVC_SUCCESS, DSVC_ACCESS, DSVC_BUSY, DSVC_COLLISION,
DSVC_INTERNAL, DSVC_NOENT.

delete_dn()

目的

DHCP ネットワークコンテナからレコードを削除します。

形式

```
int delete_dn(void *handp, const dn_rec_t *pnp);
```

説明

ハンドル handp で参照される DHCP ネットワークコンテナから、pnp の dn_cip および dn_sig 要素で指定されるレコードを削除します。更新における衝突が発生する場合には、マッチするレコードはデータ格納から削除されず、DSVC_COLLISION が返されます。

pnp の dn_sig シグニチャに 0 を指定すると、更新における衝突があるかどうかの検知は行なわず、マッチするレコードが単純に削除されます。

戻り値

DSVC_SUCCESS, DSVC_ACCESS, DSVC_NOENT, DSVC_BUSY, DSVC_INTERNAL, DSVC_COLLISION.

close_dn()

目的

ネットワークコンテナをクローズします。

形式

```
int close_dn(void **handpp);
```

説明

インスタンスハンドルを解放し、インスタンスごとの状態を消去します。

戻り値

DSVC_SUCCESS, DSVC_ACCESS, DSVC_INTERNAL.

remove_dn()

目的

データ格納のロケーションから DHCP ネットワークコンテナを削除します。

形式

```
int remove_dn(const char *location, const struct in_addr *netp);
```

説明

location 内の DHCP ネットワークコンテナ netp (ホスト順) を削除します。

戻り値

DSVC_SUCCESS, DSVC_ACCESS, DSVC_NOENT, DSVC_NO_LOCATION, DSVC_BUSY, DSVC_INTERNAL.

一般的なエラーコード

フレームワーク構成層とサービスプロバイダ層の API 関数は、エラーとして次の整数値を返します。ファイル /usr/include/dhcp_svc_public.h は、これらのコードの完全なソースとなります。

```
 * Standard interface errors
*/
#define DSVC_SUCCESS           0   /* success */
#define DSVC_EXISTS            1   /* object already exists */
#define DSVC_ACCESS            2   /* access denied */
#define DSVC_NO_CRED           3   /* No underlying credential */
#define DSVC_NOENT             4   /* object doesn't exist */
#define DSVC_BUSY              5   /* object temporarily busy (again) */
#define DSVC_INVAL             6   /* invalid argument(s) */
#define DSVC_INTERNAL          7   /* internal data store error */
#define DSVC_UNAVAILABLE       8   /* underlying service required by */
                                   /* public module unavailable */
#define DSVC_COLLISION         9   /* update collision */
#define DSVC_UNSUPPORTED       10  /* operation not supported */
#define DSVC_NO_MEMORY         11  /* operation ran out of memory */
#define DSVC_NO_RESOURCES      12  /* non-memory resources unavailable */
#define DSVC_BAD_RESOURCE      13  /* malformed/missing RESOURCE setting */
#define DSVC_BAD_PATH          14  /* malformed/missing PATH setting */
#define DSVC_MODULE_VERSION    15  /* public module version mismatch */
#define DSVC_MODULE_ERR        16  /* internal public module error */
#define DSVC_MODULE_LOAD_ERR   17  /* error loading public module */
#define DSVC_MODULE_UNLOAD_ERR 18  /* error unloading public module */
#define DSVC_MODULE_CFG_ERR    19  /* module configuration failure */
#define DSVC_SYNCH_ERR         20  /* error in synchronization protocol */
#define DSVC_NO_LOCKMGR        21  /* cannot contact lock manager */
#define DSVC_NO_LOCATION       22  /* location nonexistent */
```

```
#define DSVC_BAD_CONVER         23  /* malformed/missing CONVER setting */
#define DSVC_NO_TABLE           24  /* table does not exist */
#define DSVC_TABLE_EXISTS       25  /* table already exists */
#define DSVC_NERR                   (DSVC_TABLE_EXISTS + 1)
```

第 4 章

コード例とテスト

この章では、API 関数の適切な使用方法を示すコード例を記載します。

この章は、次の節から構成されます。

コードのテンプレート

この節では、API 関数の「一般的な」使用方法を示すテンプレートを記載します。

注 – Sun の Web サイトにある developer のページ (`http://www.sun.com/developer`) から Sun の ASCII ファイルデータ格納 (`ds_SUNWfiles`) 用のソースコードをダウンロードしてください。このモジュールのためのソースコードは、ユーザーが独自のモジュールを作成する上で役立ちます。

一般的な API 関数

このテンプレートでは、一般的な API 関数である `status()`、`version()`、`mklocation()` が使用されています。

例 **4–1** `general.c`

```
* Copyright (c) 2000 by Sun Microsystems, Inc. /*
 * Copyright (c) 2000 by Sun Microsystems, Inc.
 * All rights reserved.
```

例 **4–1** general.c　(続き)

```
 */

#pragma    ident    "@(#)general.c    1.15    00/08/16 SMI"

/*
 * This module contains the public APIs for status, version, and mklocation.
 */

#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <dhcp_svc_public.h>

/*
 * This API function instructs the underlying datastore to return its
 * general status. If the "location" argument is non-NULL, the function
 * validates the location for the data store containers (is it formed
 * correctly for the data store, and does it exist).
 */
int
status(const char *location)
{
    return (DSVC_UNSUPPORTED);
}

/*
 * Return the data store API version supported by this module.  This version
 * was implemented to support version 1 of the API.
 */
int
version(int *vp)
{
    *vp = DSVC_PUBLIC_VERSION;
    return (DSVC_SUCCESS);
}

/*
 * Create the datastore-specific "location" if it doesn't already exist.
 * Containers will ultimately be created there.
 */
int
mklocation(const char *location)
{
    return (DSVC_UNSUPPORTED);
}
```

dhcptab API 関数

このテンプレートでは、dhcptab コンテナとともに使用される関数が例示されています。

例 **4–2** dhcptab.c

```
/*
 * Copyright (c) 1998-2000 by Sun Microsystems, Inc.
 * All rights reserved.
 */

#pragma    ident    "@(#)dhcptab.c    1.12    00/08/16 SMI"

/*
 * This module contains the public API functions for managing the dhcptab
 * container.
 */

#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <dhcp_svc_public.h>

/*
 * List the current number of dhcptab container objects located at
 * "location" in "listppp". Return number of list elements in "count".
 * If no objects exist, then "count" is set to 0 and DSVC_SUCCESS is
 * returned.
 *
 * This function will block waiting for a result, if the underlying
 * data store is busy.
 */
int
list_dt(const char *location, char ***listppp, uint32_t *count)
{
    return (DSVC_UNSUPPORTED);
}

/*
 * Creates or opens the dhcptab container in "location" and initializes
 * "handlep" to point to the instance handle. When creating a new dhcptab,
 * the caller's identity is used for owner/permissions. Performs any
 * initialization needed by data store.
 */
int
open_dt(void **handlep, const char *location, uint32_t flags)
{
    return (DSVC_UNSUPPORTED);
}

/*
 * Frees instance handle, cleans up per instance state.
 */
int
close_dt(void **handlep)
{
    return (DSVC_UNSUPPORTED);
}
```

例 **4–2** dhcptab.c　(続き)

```
/*
 * Remove dhcptab container in "location" from data store. If the underlying
 * data store is busy, this function will block.
 */
int
remove_dt(const char *location)
{
    return (DSVC_UNSUPPORTED);
}

/*
 * Searches the dhcptab container for instances that match the query
 * described by the combination of query and targetp.  If the partial
 * argument is true, then lookup operations that are unable to
 * complete entirely are allowed (and considered successful).  The
 * query argument consists of 2 fields, each 16 bits long.  The lower
 * 16 bits selects which fields {key, flags} of targetp are to be
 * considered in the query.  The upper 16 bits identifies whether a
 * particular field value must match (bit set) or not match (bit
 * clear).  Bits 2-15 in both 16 bit fields are currently unused, and
 * must be set to 0.  The count field specifies the maximum number of
 * matching records to return, or -1 if any number of records may be
 * returned.  The recordsp argument is set to point to the resulting
 * list of records; if recordsp is passed in as NULL then no records
 * are actually returned. Note that these records are dynamically
 * allocated, thus the caller is responsible for freeing them.  The
 * number of records found is returned in nrecordsp; a value of 0
 * means that no records matched the query.
 */
int
lookup_dt(void *handle, boolean_t partial, uint32_t query, int32_t count,
    const dt_rec_t *targetp, dt_rec_list_t **recordsp, uint32_t *nrecordsp)
{
    return (DSVC_UNSUPPORTED);
}

/*
 * Add the record pointed to by "addp" to from the dhcptab container
 * referred to by the handle. The underlying public module will set
 * "addp's" signature as part of the data store operation.
 */
int
add_dt(void *handle, dt_rec_t *addp)
{
    return (DSVC_UNSUPPORTED);
}

/*
 * Atomically modify the record "origp" with the record "newp" in the
 * dhcptab container referred to by the handle.  "newp's" signature will
 * be set by the underlying public module.  If an update collision
 * occurs, either because "origp's" signature in the data store has changed
```

例 **4–2** dhcptab.c　*(続き)*

```
 * or "newp" would overwrite an existing record, DSVC_COLLISION is
 * returned and no update of the data store occurs.
 */
int
modify_dt(void *handle, const dt_rec_t *origp, dt_rec_t *newp)
{
    return (DSVC_UNSUPPORTED);
}

/*
 * Delete the record referred to by dtp from the dhcptab container
 * referred to by the handle. If "dtp's" signature is zero, the
 * caller is not interested in checking for collisions, and the record
 * should simply be deleted if it exists. If the signature is non-zero,
 * and the signature of the data store version of this record do not match,
 * an update collision occurs, no deletion of matching record in data store
 * is done, and DSVC_COLLISION is returned.
 */
int
delete_dt(void *handle, const dt_rec_t *dtp)
{
    return (DSVC_UNSUPPORTED);
}
```

DHCP ネットワークコンテナの API 関数

このテンプレートでは、DHCP ネットワークコンテナとともに使用される関数が例示されています。

例 **4–3** dhcp_network.c

```
/*
 * Copyright (c) 1998-2000 by Sun Microsystems, Inc.
 * All rights reserved.
 */

#pragma    ident    "@(#)dhcp_network.c    1.12    00/08/16 SMI"

/*
 * This module contains public API functions for managing dhcp network
 * containers.
 */

#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <dhcp_svc_public.h>

/*
```

例 **4–3** dhcp_network.c　(続き)

```
 * List the current number of dhcp network container objects located at
 * "location" in "listppp". Return number of list elements in "count".
 * If no objects exist, then "count" is set to 0 and DSVC_SUCCESS is
 * returned.
 *
 * This function will block if the underlying data service is busy or is
 * otherwise unvailable.
 */
int
list_dn(const char *location, char ***listppp, uint32_t *count)
{
    return (DSVC_UNSUPPORTED);
}

/*
 * Creates or opens the dhcp network container "netp" (host order) in
 * "location" and initializes "handlep" to point to the instance handle.
 * Performs any initialization needed by data store. New containers are
 * created with the identity of the caller.
 */
int
open_dn(void **handlep, const char *location, uint32_t flags,
    const struct in_addr *netp)
{
    return (DSVC_UNSUPPORTED);
}

/*
 * Frees instance handle, cleans up per instance state.
 */
int
close_dn(void **handlep)
{
    return (DSVC_UNSUPPORTED);
}

/*
 * Remove DHCP network container "netp" (host order) in location.
 * This function will block if the underlying data service is busy or
 * otherwise unavailable.
 */
int
remove_dn(const char *location, const struct in_addr *netp)
{
    return (DSVC_UNSUPPORTED);
}

/*
 * Searches DHCP network container for instances that match the query
 * described by the combination of query and targetp.  If the partial
 * argument is true, then lookup operations that are unable to
 * complete entirely are allowed (and considered successful).  The
 * query argument consists of 2 fields, each 16 bits long.  The lower
```

例 **4–3** dhcp_network.c　(続き)

```
 * 16 bits selects which fields {client_id, flags, client_ip,
 * server_ip, expiration, macro, or comment} of targetp are to be
 * considered in the query.  The upper 16 bits identifies whether a
 * particular field value must match (bit set) or not match (bit
 * clear).  Bits 7-15 in both 16 bit fields are currently unused, and
 * must be set to 0.  The count field specifies the maximum number of
 * matching records to return, or -1 if any number of records may be
 * returned.  The recordsp argument is set to point to the resulting
 * list of records; if recordsp is passed in as NULL then no records
 * are actually returned. Note that these records are dynamically
 * allocated, thus the caller is responsible for freeing them.  The
 * number of records found is returned in nrecordsp; a value of 0 means
 * that no records matched the query.
 */
int
lookup_dn(void *handle, boolean_t partial, uint32_t query, int32_t count,
    const dn_rec_t *targetp, dn_rec_list_t **recordsp, uint32_t *nrecordsp)
{
    return (DSVC_UNSUPPORTED);
}

/*
 * Add the record pointed to by "addp" to from the dhcp network container
 * referred to by the handle.  The underlying public module will set
 * "addp's" signature as part of the data store operation.
 */
int
add_dn(void *handle, dn_rec_t *addp)
{
    return (DSVC_UNSUPPORTED);
}

/*
 * Atomically modify the record "origp" with the record "newp" in the dhcp
 * network container referred to by the handle.  "newp's" signature will
 * be set by the underlying public module.  If an update collision
 * occurs, either because "origp's" signature in the data store has changed
 * or "newp" would overwrite an preexisting record, DSVC_COLLISION is
 * returned and no update of the data store occurs.
 */
int
modify_dn(void *handle, const dn_rec_t *origp, dn_rec_t *newp)
{
    return (DSVC_UNSUPPORTED);
}

/*
 * Delete the record pointed to by "pnp" from the dhcp network container
 * referred to by the handle. If "pnp's" signature is zero, the caller
 * is not interested in checking for collisions, and the record should
 * simply be deleted if it exists. If the signature is non-zero, and the
 * signature of the data store version of this record do not match, an
 * update collision occurs, no deletion of any record is done, and
```

例 **4–3** dhcp_network.c　(続き)

```
 * DSVC_COLLISION is returned.
 */
int
delete_dn(void *handle, const dn_rec_t *pnp)
{
    return (DSVC_UNSUPPORTED);
}
```

パブリックモジュールのテスト

パブリックモジュールをテストする上で助けになるテストセットのダウンロードについては、http://www.sun.com/developer を参照してください。

索引

な

ね

は

ふ

も

れ