Solaris 8 のソフトウェア開発 (追補)

Sun Microsystems, Inc.
901 San Antonio Road
Palo Alto, CA 94303
U.S.A. 650-960-1300

Part Number 816-0112-10
2001 年 5 月

原典: *Solaris 8 Software Developer Supplement*

Part No: 806-7503-10

Revision A

目次

はじめに

このマニュアルでは、Solaris™ 8 Update リリースの新機能について説明します。ここでの説明は、すでにリリースされている Solaris 8 のマニュアルセットの内容を補足または変更するものです。Solaris のマニュアルは、Solaris 8 DOCUMENTATION CD に含まれています。

注 - Solaris オペレーティング環境は、2 種類のハードウェア (プラットフォーム) 上で動作します。つまり、SPARC™ と IA (Intel アーキテクチャ) です。Solaris オペレーティング環境は、64 ビットと 32 ビットの両方のアドレス空間で動作し、IA では 32 ビットのアドレス空間でのみ動作します。このマニュアルで説明する情報は、章、節、注、箇条書き、図、表、例、またはコード例において特に明記しない限り、両方のプラットフォームおよびアドレス空間に該当します。

Sun のマニュアルの注文方法

専門書を扱うインターネットの書店 Fatbrain.com から、米国 Sun Microsystems™, Inc. (以降、Sun™ とします) のマニュアルをご注文いただけます。

マニュアルのリストと注文方法については、`http://www1.fatbrain.com/documentation/sun` の Sun Documentation Center をご覧ください。

Sun のオンラインマニュアル

http://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。

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。

表 P–1 表記上の規則 続く

字体または記号	意味	例
「 」	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'`

ただし AnswerBook2™ では、ユーザーが入力する文字と画面上のコンピュータ出力は区別して表示されません。

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、「IA」という用語は、Intel 32 ビットのプロセッサアーキテクチャを意味します。これには、Pentium、Pentium Pro、Pentium II、Pentium II Xeon、Celeron、Pentium III、Pentium III Xeon の各プロセッサ、および AMD、Cyrix が提供する互換マイクロプロセッサチップが含まれます。

第 1 章

新規機能の概要

この章では、Solaris 8 Update リリースに追加された新機能について説明します。

注 - 最新のマニュアルページを参照するには、man コマンドを使用してください。Solaris 8 Update リリースのマニュアルページには、「*Solaris 8 Reference Manual Collection*」には記載されていない新しい情報も提供されています。

表 1–1 Solaris 8 の新規機能

機能	更新リリース
デバイスドライバの作成	
SPARC: ドライバ強化テストハーネスは、Solaris デバイスドライバの開発ツールです。開発中のドライバがハードウェアにアクセスするときに、テストハーネスによってハードウェア障害のさまざまなシミュレーションが投入されます。この障害投入テストハーネスは、SPARC ベースのデバイスドライバの耐性をテストします。 詳細は、第 4 章を参照してください。	4/01
「高可用性ドライバ」の節では、ドライバの強化およびサービス利用可能性の確保によって高可用性をサポートするドライバの設計方法について、詳しく説明しています。これは、Solaris 8 の『*Writing Device Drivers*』の内容を補足するものです。 詳細は、第 3 章を参照してください。	10/00

表 1–1 Solaris 8 の新規機能　続く

機能	更新リリース
Generic LAN ドライバ (GLD) を使用すると、Solaris ネットワークドライバに必要な STREAMS および Data Link Provider Interface (DLPI) 機能の大部分を実装できます。Solaris 8 10/00 より前のリリースでは、GLD モジュールを利用できるのは、Solaris の Intel 版ネットワークドライバに限定されていました。Solaris 8 10/00 からは、Solaris の SPARC 版ネットワークドライバでも GLD を利用できます。Solaris 8 4/01 リリースで、GLD はバグ修正により更新されました。 詳細は、第 5 章を参照してください。	10/00。4/01 で更新。
言語サポート	
File System Safe Universal Transformation Format (UTF-8) は、X/Open によって Unicode の複数バイト表現として定義されているエンコード方式です。UTF-8 は、ヨーロッパおよびアジア言語に対応する、従来の Solaris のシングルバイトおよび複数バイトロケールで使用できた文字をほぼ網羅します。Solaris 8 10/00 リリースでは、ロシア語およびポーランド語対応の UTF-8 ロケールと、カタロニア語対応の 2 つの新しいロケールが追加されました。Solaris 8 4/01 リリースでは、トルコ語の UTF-8 コードセットとロシア語の UTF-8 コードセットが、既存の東ヨーロッパのロケールの表に追加されました。 詳細は、75ページの「ヨーロッパ言語版 Solaris ソフトウェア用に追加された部分ロケール」を参照してください。	10/00。4/01 で更新。
`mp` プログラムは、さまざまな Solaris ロケールの国際テキストファイルを受け付け、指定されたロケールに合った出力を行います。出力には、双方向のテキストレンダリングなどの正確なテキストレイアウトも含まれます。また、`mp` では複合テキストレイアウト (CTL) としてのレイアウトもサポートされています。各ロケールの `mp` に対するシステムフォント構成によっては、PostScript™ の出力ファイルに、Solaris システム上のスケーラブルフォントやビットマップフォントのグリフイメージを組み込めます。 詳細は、第 8 章を参照してください。	4/01
開発ツール	
`appcert` ユーティリティは、オブジェクトファイルの Solaris ABI への準拠を検証します。Solaris ABI に準拠していると、今後の Solaris ソフトウェアのリリースに対するアプリケーションの互換性が大幅に向上します。 詳細は、第 10 章を参照してください。	4/01

表 1–1 Solaris 8 の新規機能　続く

機能	更新リリース
Web-Based Enterprise Management (WBEM) では、複数のプラットフォーム上のシステム、ネットワーク、およびデバイスを Web ベースで管理するための標準が提供されています。Sun WBEM Software Developer's Toolkit (SDK) では、ソフトウェア開発者は、Solaris オペレーティング環境におけるリソースを管理する、標準ベースのアプリケーションの作成が可能になります。また、開発者はこのツールキットを使用して、データにアクセスするときに管理対象のリソースと通信するプログラムである、プロバイダを作成することもできます。Sun WBEM SDK には、Common Information Model (CIM) のリソースの記述や管理を行うためのクライアントアプリケーションプログラミングインタフェース (API: Client Application Programming Interface)、および管理対象リソースの動的データの取得や設定を行うためのプロバイダ API が含まれています。また、Sun WBEM SDK では、システム上で管理対象リソースの作成や表示を行うための Java アプリケーションである CIM WorkShop、および WBEM のクライアントプログラムとプロバイダプログラムのサンプル集も提供しています。 詳細は、『*Sun WBEM SDK* 開発ガイド』を参照してください。	4/01
SPARC:『マルチスレッドのプログラミング』が更新され、バグ ID 4308968、4356675、4356690 が修正されました。 このマニュアルは、『マルチスレッドのプログラミング』を参照してください。	1/01
『リンカーとライブラリ』が更新され、新しい機能が追加されました。 詳細は、113ページの「『リンカーとライブラリ』の変更点」を参照してください。	1/01 および 10/00
システムインタフェースツール	
『システムインタフェース』が更新され、バグが修正されました。このリリースでは、テキストやソースコード例のタイプミスがいくつか修正されています。 このマニュアルは、『システムインタフェース』を参照してください。	6/00
Java™ のリリース	

表 1–1 Solaris 8 の新規機能　続く

機能	更新リリース
J2SE™ 1.3.0 としても知られている Java 2 SDK Standard Edition バージョン 1.3.0 は、Java 2 SDK のアップグレードリリースです。J2SE リリースには、次の新機能や拡張機能が組み込まれています。 ■ パフォーマンスの向上 Java HotSpot™ 技術と最適化された実行時ライブラリにより、J2SE 1.3.0 は今までで最も高速な Java プラットフォームとなっています。 ■ より簡単な Web への展開 アプレットのキャッシュやオプションパッケージの自動インストールなど、J2SE 1.3.0 の Java Plug-In コンポーネントによる機能が新たに追加されたため、プログラムを Web 上で表示するときの速度や柔軟性が向上しました。 ■ 企業レベルでの相互運用性 J2SE 1.3.0 に RMI/IIOP と Java Naming and Directory Interface™ が追加されたため、Java 2 プラットフォームの相互運用性が向上しました。 ■ セキュリティの強化 RSA 電子署名、動的信用管理、X.509 証明書、Netscape™ 署名ファイルの検証が新たにサポートされたため、開発者はさまざまな手段で電子データを保護できるようになりました。 ■ Java サウンド J2SE 1.3.0 には強力なサウンド API が新たに提供されています。これまでのプラットフォームのリリースでは、オーディオのサポートがオーディオクリップの基本的な再生に限定されていました。このリリースでは、低レベルのオーディオサポートに対する標準のクラスやインタフェースが、Java 2 プラットフォームによって定義されました。 ■ 拡張された API と開発しやすさの向上 開発側からの要望に応えて、J2SE 1.3.0 は Java 2 プラットフォームに多彩な機能を追加しています。追加された機能によって、プラットフォームの機能性がさらに向上し、一層強力なアプリケーションが開発できるようになりました。また、新規の機能の多くは、開発工程の短縮や効率化を実現する機能です。 J2SE の改善点の詳細は、125ページの「Java 2 SDK Standard Edition バージョン 1.3.0」を参照してください。	4/01
J2SE 1.2.2_07a では、J2SE 1.2.2 シリーズの前回のリリースで見つかったバグが修正されています。J2SE 1.2.2_07a での重要なバグ修正に、J2SE 1.2.2_05 で発生したパフォーマンスの低下にに対する修正があります。J2SE 1.2.2_07a におけるバグ修正の詳細は、`http://java.sun.com/j2se/1.2/ReleaseNotes.html` を参照してください。	4/01
Java 2 SDK 1.2.2_06 および JDK™ 1.1.8_12 が、前回リリースのバグ修正により改良されました。	1/01

表 1–1 Solaris 8 の新規機能　続く

機能	更新リリース
The Java 2 SDK 1.2.2_05a には、次の新機能が含まれます。 ■ スケーラビリティの向上 (20 以上の CPU に対応) ■ JIT (just-in time) コンパイラの最適化の向上 ■ テキストの描画性能の向上 ■ `poller` クラスのデモパッケージ ■ Swing の向上 詳細は、表 18–1 を参照してください。	10/00
32 ビット: `mod_jserv` モジュールおよび関連ファイルの追加によって、Apache Web サーバーで Java Servlet がサポートされるようになりました。 詳細は、138ページの「Apache Web サーバーにおける Java Servlet のサポート」を参照してください。	10/00
アーリーアクセス	
このリリースでは、アーリーアクセス (EA) ディレクトリにアーリーアクセスソフトウェアが含まれています。詳細は、Solaris 8 のリリースの SOFTWARE 2 of 2 CD に含まれる各アーリーアクセスソフトウェアの `README` を参照してください。	

第 2 章

デバイスドライバの作成についてのトピック

このトピックでは、Solaris 環境におけるデバイスドライバの作成方法について記述します。次の章で構成されています。

第 3 章	デバイスの障害から発生するドライバの障害を防止するための、ドライバ強化についての情報を記述します。
第 4 章	テストハーネスの構成方法、エラー投入の仕様 (*errdefs*) の作成方法、デバイスドライバでのテストの実行方法について説明します。
第 5 章	Generic LAN ドライバ (GLD) を使用して、Solaris ネットワークドライバに STREAMS および Data Link Provider Interface (DLPI) の機能の大部分を実装するための情報を記述します。

高可用性ドライバ

Solaris 8 10/00 リリースでドライバの強化が新規に追加されました。Solaris デバイスドライバの作成方法についての詳細は、『*Writing Device Drivers*』を参照してください。

可用性 (availability) は、障害の発生率と修復速度の両方により決定されます。多くの場合、個々のデバイスで障害が発生しても、トータルなシステム障害に至るわけではありません。冗長構成のハードウェアコンポーネントを高可用性をサポートする設計のドライバと組み合わせることで、個々のコンポーネントで障害が発生しても、システム動作を継続することができます。このようなドライバを使用している場合は一般に、サービスを継続したまま、システムを修復できます。

デバイス障害がもたらすドライバ障害をプログラミングによって排除することを、ドライバの強化 (driver hardening) といいます。強化されたドライバは、故障したデバイスからシステムの他の部分にエラーが拡がらないように、エラーに耐え、他を守ることができます。

障害の特定および回復/修理所要時間の短縮に役立つ、ドライバ内部の機能により、システムのサービス利用可能性が向上します。その結果、修復所要時間が短縮され、可用性が向上します。

注 - 最新のマニュアルページを参照するには、`man` コマンドを使用してください。Solaris 8 Update リリースのマニュアルページには、「*Solaris 8 Reference Manual Collection*」には記載されていない新しい情報も提供されています。

ドライバの強化

強化 (hardening) とは、ドライバが制御する入出力デバイスで障害が発生しても、またはシステムコアの外部に起因するその他の障害が発生しても、そのドライバが正常に動作するようにするプロセスです。強化されたドライバは、パニックに陥ったり、システムをハングさせたり、またこのような障害の結果として壊れたデータを無秩序に拡散させたりしてはなりません。

ドライバの開発者は、次の役割を担います。

- DDI 関数を正しく使用する
- デバイス入出力の破壊を検出して報告する
- 異質な割り込みロジックのデバイスを処理する

すべての Solaris ドライバは強化されるべきです。強化されたドライバは、次の規則を守ります。

- ハードウェアの各部分をそれぞれ別個のデバイスドライバインスタンスで制御する必要があります。
- プログラムされた入出力 (PIO) は、必ず、適切なデータアクセスハンドルを使用して、DDI アクセス関数からのみ実行されなければなりません。
- デバイスドライバは、デバイスから受け取ったデータが壊れている可能性があることを想定しなければなりません。ドライバは、データを使用する前に、データの整合性をチェックしなければなりません。
- ドライバは、検出した障害の影響を制御しなければなりません。既知の不良データをシステムの他の部分に流してはなりません。
- ドライバは、デバイスによる DMA バッファ (`DDI_DMA_READ`) へのすべての書き込みが、ドライバが全面的に制御するメモリーページ内に収まるようにしなければなりません。これにより、DMA の障害が原因でシステムメインメモリーの部分が破壊される事態を防止します。
- デバイスドライバは、デバイスがロックアップした場合に、システムリソースを無制限に割り当ててはなりません。デバイスが連続使用を要求した場合には、ドライバはタイムアウトすべきです。ドライバはさらに、有害な stuck 割り込み要求を検出し、適切な処置を取らなければなりません。
- ドライバは、障害発生後にリソースを解放しなければなりません。たとえば、ハードウェア障害の後でも、システムはすべてのマイナーデバイスをクローズし、ドライバインスタンスを切り離すことができなければなりません。

デバイスドライバのインスタンス

Solaris カーネルは、複数のドライバのインスタンスの使用を認めます。インスタンスごとに専用のデータ領域を持ちますが、テキストおよび一部のグローバルデータは、他のインスタンスとの共有になります。デバイスはインスタンス単位で管理されます。強化されたドライバは、ドライバがフェイルオーバーを内部的に処理する設計になっていない限り、ハードウェアの部分ごとに別々のインスタンスを使用すべきです。たとえば、多機能カードなど、スロットごとに複数のドライバインスタンスが存在することになりますが、これは Solaris デバイスドライバの標準の動作です。

DDI アクセスハンドルの排他的使用

強化されたドライバによるすべてのプログラムされた入出力 (PIO) アクセスでは、必ず、`ddi_get`*X*、`ddi_put`*X*、`ddi_rep_get`*X*、および `ddi_rep_put`*X* ルーチンファミリーに含まれる Solaris DDI アクセス関数を使用しなければなりません。ドライバが、`ddi_regs_map_setup(9F)` から返されたアドレスを使用して、対応付けされたレジスタに直接アクセスしてはなりません。アクセスハンドルを使用することによって、入出力エラーが確実に制御され、その影響が戻り値に限定されるので、マシン状態の他の部分を壊す可能性が減少します (アクセスハンドルを使用しないため、`ddi_peek(9F)` および `ddi_poke(9F)` ルーチンは避けてください)。

DDI アクセスメカニズムは、カーネルにデータを読み込む方法を制御する機会を提供するので重要です。DDI アクセスルーチンは、バスタイムアウトトラップの影響を限定することによって、保護能力を提供します。

破壊されたデータの検出

以下の節では、どこでデータ破壊が発生する可能性があるか、およびそれらを検出するにはどのような手順が必要かについて説明します。

デバイス管理および管理データの破壊

ドライバは、PIO によるか DMA によるかを問わず、デバイスから取得したデータは壊れている可能性があるということを想定しなければなりません。特に、デバイスから得たデータから読み取った、または計算されたポインタ、メモリーオフセット、配列インデックスは、慎重に扱ってください。この種の値は有害な場合があり、参照に利用される場合、カーネルパニックを引き起こす可能性があります。この種の値はすべて、範囲および整列 (必要な場合) をチェックしてから使用してください。

ポインタが有害でなくても、誤った結果を導く可能性もあります。たとえば、オブジェクトの有効なインスタンスを指し示していても、該当する正しいインスタンスではないことがあります。可能であれば、ドライバはポインタと指示先オブジェクトをクロスチェックするか、またはポインタによって取得したデータの妥当性を検証すべきです。

他のタイプのデータ (パケット長、ステータスワード、チャネル ID など) も誤った結果を導く可能性があります。データの各タイプを可能な範囲でチェックすべきです。たとえば、パケット長の場合は、範囲をチェックして、負ではないか、収容バッファより大きくないかどうかを確認できます。また、ステータスワードの場合は、「不可能」ビットの有無をチェックできます。チャネル ID は有効 ID のリストと突き合わせます。

値を使用して Stream を識別する場合、ドライバは Stream が実際に存在しているかどうかを確認しなければなりません。STREAMS 処理は非同期の性質があるので、デバイスの割り込みがまだ処理されないうちに、Stream が取り除かれてしまうことがあります。

ドライバはデバイスからデータを再読み込みしてはなりません。データは 1 回だけ読み取って、妥当性を検証し、ドライバのローカル状態で格納します。こうすることによって、最初に読み取って検証したときには正しかったが、後で再読み取りしたときには正しくないというデータによる危険性を回避することができます。

さらに、連続して `BUSY` 状態を返すデバイス、または別のバッファ処理を要求するデバイスがシステム全体をロックアップすることがないように、ドライバで、すべてのループがバインドされていることも確認すべきです。

受信したデータの破壊

デバイスエラーによって、受信バッファに壊れたデータが格納されてしまうことがあります。このような破壊を、デバイスドメインの外 (たとえば、ネットワーク内など) で発生した破壊と区別することはできません。通常は、プロトコルスタックのトランスポート層やデバイスを使用するアプリケーション内部での整合性チェックなど、この種の破壊に対処する既存のソフトウェアが既に配備されています。

ディスクドライバなどのように、受信したデータの整合性を上位層でチェックしない場合は、ドライバ内部で整合性をチェックすることができます。受信したデータの破壊を検出する方法は、一般にデバイス固有です (チェックサム、CRC など)。

障害の検出

従来のデバイスドライバでは、障害を検出した場合に、デバイスまでのデータパスを無効にできます。PIO アクセスが無効にされている場合にそのデバイスから読み取りを行おうとすると、undefined 値が返り、書き込みが無視されます。DMA アクセスが無効にされている場合は、デバイスはメモリーへのアクセスが禁止される可能性があり、または読み取り時に未定義データを受け取り、書き込みが廃棄される可能性があります。

デバイスドライバは次の DDI ルーチンを使用することによって、データパスが無効にされているかどうかを検出することができます。

- ddi_check_acc_handle(9F)
- ddi_check_dma_handle(9F)

各関数は、指定のハンドルによって示されたデータパスに影響を与えるような障害が検出されているかどうかを調べます。いずれかの関数が DDI_FAILURE を返した場合、そのデータパスに障害があります。ドライバは ddi_dev_report_fault(9F) を使用してその障害を報告し、必要なクリーンアップを実行し、さらに可能であれば、呼出側に適切なエラーを返す必要があります。

障害の封じ込め

システムの整合性を維持するには、システム状態が変更されないうちに障害を検出する必要があります。したがって、ドライバは、システムが使用する予定のデータがデバイスから返されるたびに、障害の有無をテストしなければなりません。

- 上位層にデータブロックを渡す直前など、重要な接合部分で、ddi_check_acc_handle(9F) および ddi_check_dma_handle(9F) を呼び出す必要があります。
- デバイスで障害が発生した場合は、ドライバから外部へデータを転送してはなりません。
- ドライバでは、障害がシステムの整合性に与える影響が他に無いかどうかをチェックしなければなりません。ドライバは、データを転送できない場合には、メモリーなどのカーネルリソースが永久に失われていないかどうかを確認しなければなりません。発生することのない信号を待機して、スレッドがブロックのままになっていてはなりません。

- ドライバは、障害状態時の処理を制限する必要があります (たとえば、wput ルーチンでメッセージを解放する、障害のあるボードからの割り込みを永久的に無効にするようにする、など)。

DMA の切り離し

障害のあるデバイスは、バス上で不適切な DMA 転送を開始する可能性があります。このデータ転送によって、配信済みの正常データが破壊されることがあります。障害のあるデバイスは、そのデバイスのドライバに属さないメモリーにまで悪影響をおよぼすような破壊されたアドレスを生成しかねません。

IOMMU が備わったシステムでは、デバイスは DMA 用に書き込み可能として対応付けされたページに限って書き込むことができます。したがって、DMA 書き込みのターゲットとなるページは、1 つのドライバインスタンスが単独で所有し、他のカーネル構造体と共有することはありません。該当するページが DMA 用に書き込み可能として対応付けされている場合、ドライバはそのページのデータを疑ってみるべきです。ページをドライバの外部に渡す前に、またはデータを検証する前に、ページと IOMMU の対応付けを解除しなければなりません。

ddi_umem_alloc(9F) を使用して、整合ページ全体が割り当てられる、または複数のページを割り当てて最初のページ境界より下のメモリーを無視することができます。ddi_ptob(9F) を使用すると、IOMMU ページのサイズを調べることができます。

あるいは、ドライバは、メモリーの安全な部分にデータをコピーしてから、そのデータを処理することもできます。この場合、最初に ddi_dma_sync(9F) を使用してデータを同期させる必要があります。

ddi_dma_sync(9F) を呼び出すときには、DMA を使用してデバイスにデータを転送する前に SYNC_FOR_DEV を指定し、デバイスからメモリーへ DMA を使用してデータを転送した後で SYNC_FOR_CPU を指定する必要があります。

IOMMU が備わった一部の PCI ベースのシステムでは、デバイスは PCI デュアルアドレスサイクル (64 ビットアドレス) を使用して、IOMMU をバイパスすることができます。その結果、デバイスにおいてメインメモリーの領域が破壊されてしまう潜在的可能性が生じます。強化されたデバイスドライバでは、このようなモードを使用しようとしてはならず、使用不可にしておくべきです。

stuck 割り込みの処理

割り込みが絶えないと、システムパフォーマンスが大幅に低下し、シングルプロセッサマシンの場合はほぼ確実に作業が進まなくなってしまうので、ドライバは stuck 割り込みを識別する必要があります。

時にはドライバにとって、特定の割り込みを偽として識別するのが困難な場合があります。ネットワークドライバの場合、受信した割り込みが指示されても、新しいバッファが利用できなければ作業は不要です。これが単独で発生した場合は問題ありません。実際の作業は別のルーチン (たとえば読み取りサービスなど) によってすでに完了している可能性があるからです。

一方、ドライバが処理する作業を伴わない割り込みが連続した場合、stuck 割り込みラインを示していることがあります。そのため、防御手段を講じる前に、すべてのプラットフォームで多数の明らかに偽の割り込みの発生を許してしまうことになります。

処理すべき作業がありそうなのにハングしてしまったデバイスは、対応するバッファ記述子を更新できなかった可能性があります。ドライバは、このような繰り返しの要求に対して防御しなければなりません。

場合によっては、プラットフォーム固有のバスドライバに請求によらない持続的な割り込みを識別する能力があり、攻撃しているデバイスを使用禁止にできます。しかし、これは有効な割り込みを識別して適切な値を返すことができるという、ドライバの能力に依存します。ドライバはしたがって、デバイスが正当な割り込みをかけた (すなわちデバイスがドライバになんらかの有用な作業を行うことを実際に要求している) ことを検出した場合以外は、`DDI_INTR_UNCLAIMED` を返さなければなりません。

偶発性の高い他の割り込みの正当性を確認することは、さらに困難です。この目的のためには、割り込みが有効かどうかを評価する手段として、割り込み想定フラグが役立ちます。デバイスの記述子が全部すでに割り当てられている場合に生成できる、descriptor free などの割り込みを例として仮定してください。ドライバはカードの最後の記述子を使用してしまったことを検出すると、割り込み想定フラグを設定できます。対応する割り込みが伝達された時にこのフラグが設定されていないと、疑わしいといえます。

メディアが切断された、またはフレーム同期が失われたといったことを伝える、情報通知目的の割り込みには、予測できないものがあります。このような割り込みが stuck かどうかを検出する最も簡単な方法は、最初のイベントでその送信元を次のポーリングサイクルまでマスクすることです。

禁止されている間に割り込みが再び発生した場合は、偽の割り込みとみなすべきです。デバイスによっては、マスクレジスタが対応する送信元を無効にし、割り込みを発生させない場合でも読み取ることのできる、割り込み状態ビットがあります。ドライバの設計者は、それぞれのデバイスに合わせて、より適切なアルゴリズムを工夫することができます。

割り込み状態ビットが無限ループに陥らないようにしてください。パスの最初に設定された状態ビットセットがいずれも実際の作業を必要としない場合は、このようなループを切断してください。

ドライバの強化に関するその他の考慮事項

すでに説明した要件のほかに、ドライバの開発者は、次の問題も考慮する必要があります。

- スレッドのインタラクション
- トップダウン式要求における危険
- 代替の対応策

スレッドのインタラクション

デバイスドライバのカーネルパニックは、デバイス障害の発生後に生じた予想外のカーネルスレッドのインタラクションによって引き起こされることがよくあります。デバイスで障害が発生すると、設計者が予想しなかった形でスレッドのインタラクションが起きることがあります。

たとえば、処理ルーチンが正常に完了しないうちに打ち切られた場合、条件変数を待機している他のスレッドに知らせることができないことがあります。他のモジュールに障害を知らせようとしたり、または予想外のコールバックを処理しようとしたりすると、望ましくないスレッドのインタラクションが発生する可能性があります。デバイス障害時に発生する可能性のある、相互排他ロックの取得と放棄の順序を検証してください。

アップストリームの STREAMS モジュールを起点とするスレッドは、予想外にそのモジュールをコールバックするために使用された場合、望ましくないパラドックスに陥る可能性があります。代替スレッドを使用して、例外メッセージを処理してください。たとえば、wput 手順では、読み取り側の putnext で直接通信するのではなく、読み取り側のサービスルーチンを使用して M_ERROR と通信することができます。

障害の発生したSTREAMSデバイスが、クローズ時に(障害が原因で)静止できなかった場合、Streamを取り除いた後に割り込みが発生する可能性があります。割り込みハンドラは、古いStreamポインタを使用して、メッセージを処理しようとしてはなりません。

トップダウン式要求における危険

ドライバの設計者は、ハードウェアの故障からシステムを保護する一方で、ドライバの誤用に対しても防御する必要があります。ドライバは、カーネル基盤は常に正しい(信頼できるコア)ということを前提にできますが、ドライバに渡されるユーザー要求は潜在的な破壊性を有している可能性があります。

たとえば、ユーザーが提供したデータブロック(M_IOCTL)に対してアクションを実行することをユーザーが要求し、そのデータブロックがメッセージの制御部で指示されたサイズより小さいという場合があります。ドライバはユーザーアプリケーションを信用してはなりません。

設計時には、害を引き起こす可能性があるという観点から、受け取ることのできる各タイプのioctlの構造を検討すべきです。ドライバは、異常なioctlを処理しないように、チェックしなければなりません。

代替の対応策

ドライバは障害の起きたハードウェアを引き続き使用してサービスを提供し、代替のデバイスアクセス方法を使用することによって、特定された問題への対処を試みることができます。ハードウェアの故障が予測不能で、設計が複雑になるリスクを伴うことを考えると、代替の対応策をとることが常に賢明な選択というわけではありません。せいぜい、周期的な割り込みポーリングと再試行に限定すべきです。周期的にデバイスを再試行することによって、ドライバはデバイスが復旧したかどうかを知ることができます。周期的ポーリングを使用すると、ドライバが割り込みを強制的に禁止された後でも、割り込みメカニズムを制御することができます。

重要なシステムサービスを提供できるように、システムが常に代替デバイスを備えているのが理想です。カーネルまたはユーザースペースのサービスマルチプレクサは、デバイス障害時にシステムのサービスを維持する最良の手段です。しかし、この章ではこの種の方式については記載していません。

サービス利用可能性

サービス利用可能性を確保するには、ドライバが下記を実行できるようにしなければなりません。

- 障害デバイスを検出して障害を報告する
- デバイスを取り除く (Solaris のホットプラグモデルでサポートされる)
- 新しいデバイスを追加する (Solaris のホットプラグモデルでサポートされる)
- 潜在的な障害を検出できるように定期的な健全性チェックを実行する

現在のデバイス状態のチェック

ドライバはリソースの無駄なコミットを避けるために、適切なポイントでデバイス状態をチェックしなければなりません。`ddi_get_devstate(9F)` 関数を使用すると、ドライバでフレームワークによって維持されているデバイスの現在の状態を判別できます。

```
ddi_devstate_t ddi_get_devstate(dev_info_t *dip);
```

ドライバは通常、`OFFLINE` のデバイスを扱うときには呼び出されません。デバイス状態は一般に、再構成アクティビティによって変更されたというような、以前のデバイス障害レポートを反映します。

デバイス障害時の適切な動作

システムは、デバイスのサービス利用可能性に対する影響という観点から、障害を報告しなければなりません。一般に次の場合に、サービス損失が予期されます。

- PIO または DMA エラーが検出された。
- データ破壊が検出された。
- デバイスがロックされている、またはハングしている (たとえば、コマンドが完了しない場合など)。
- ドライバの設計時にあり得ないと見なされたために、ドライバが対処しないことになっている状態が発生した。

ddi_get_decstate(9F) から返されたデバイス状態が、デバイスが使用不能であることを示している場合、ドライバは新規または未処理の入出力要求をすべて拒否し、(可能であれば) 適切なエラーコード (EIO など) を返す必要があります。STREAMS ドライバでは、M_ERROR または M_HANGUP をアップストリームに送り、ドライバが使用不能であることを示さなければなりません。

各メジャーエントリポイントで、またオプションで操作のためにリソースをコミットする前、および障害の報告後に、デバイスの状態をチェックする必要があります。いずれかの段階で、デバイスが使用不能であることが判明した場合は、ドライバで必要なクリーンアップ処理 (リソースの解放など) を実行し、タイミングよく戻る必要があります。再試行または回復処理を試行してはなりません。障害の報告も不要です。状態は障害ではなく、状態はすでにフレームワークと管理エージェントに認識されています。状態は、現在の要求および他の未処理すなわちキュー内の要求に完了マークを付け、可能であれば再びエラー標識を出します。

ioctl() エントリポイントは、デバイスが使用不能で、他 (エラー状態の回復など) が引き続き動作中である場合、所定の ioctl 操作における、デバイスに対する入出力 (たとえば、ディスクのフォーマットなど) 障害を意味する問題を提示します。したがって、状態チェックはコマンド単位で行う必要があります。または、別のエントリポイントまたはマイナーデバイスモードを使用して、あらゆる状態でこれらの操作を実装できますが、この方法は既存アプリケーションとの互換性の点から制約を受ける可能性があります。

close() は、デバイスが使用不能な場合であっても、常に正常に完了しなければなりません。デバイスが使用不能な場合は、割り込みハンドラがその後のすべての割り込みに DDI_INTR_UNCLAIMED を返す必要があります。割り込みが引き続き発生する場合は、最終的に割り込みが禁止されることがあります。

障害の報告

次の関数は、ドライバがデバイス障害を検出したことをシステムに通知します。

```
void ddi_dev_report_fault(dev_info_t *dip, ddi_fault_impact_t impact,
            ddi_fault_location_t location, const char *message);
```

impact パラメータは、デバイスの通常のサービス利用可能性に障害が与える影響を示します。このパラメータは、障害に対する適切な処置を決定するために、システムの障害管理コンポーネントが使用します。この処置によってデバイス状態が変化することがあります。サービスが損われる障害が発生すると、デバイス状態が

DOWN に変わります。サービスが低下する障害の場合は、デバイス状態が DEGRADED に変わります。

デバイスは次の場合に、障害として報告されなければなりません。

- PIO エラーが検出された。
- 破壊されたデータが検出された。
- デバイスがロックアップされた。

できるだけ、ドライバが同じ障害を繰り返し報告しないようにします。特に、デバイスがすでに使用不能状態になっている場合に、ドライバがエラーを報告しても無駄です (望ましくありません) (ddi_get_devstate(9F) のマニュアルページを参照)。

接続プロセスでハードウェア障害が検出された場合、ドライバは ddi_dev_report_fault(9F) のを使用して障害を報告するとともに、DDI_FAILURE を返さなければなりません。

定期的な健全性チェック

潜在的な障害とは、他のアクションが発生するまで明らかにならない障害のことです。たとえば、コールドスタンバイモードのデバイスで発生したハードウェア障害は、マスターデバイスで障害が発生するまで検出されない可能性があります。マスターデバイスで障害が発生した時点で、システムに 2 つの障害デバイスがあり、動作を継続できない可能性があることが判明します。

一般論として、未検出のままの潜在的障害は、最終的にシステム障害につながります。潜在的な障害を検査しなかった場合は、冗長型システム全体の可用性が損なわれる危険があります。これを回避するには、デバイスドライバで潜在的な障害を検出し、他の障害と同様に報告しなければなりません。

ドライバには、デバイスに対する定期的な健全性チェックを行うメカニズムが必要です。デバイスを第 2 デバイスすなわちフェイルオーバーデバイスにできる、障害耐久 (フォールトトレラント) の状況では、故障した第 2 デバイスを早い段階で検出することが重要です。そうすれば、第 1 デバイスで障害が発生しないうちに修理または交換することができます。

定期的な健全性チェックには、次のものがあります。

- ボードに対してクイックアクセスチェック (書き込み、読み取り) を実行し、さらに ddi_check_acc_handle(9F) ルーチンを使用してデバイスをチェックする。
- 前回のポーリング以後、値がドライバの予期値と決定的に異なっているデバイスについて、レジスタまたはメモリーロケーションをチェックする。

一般に決定的な動作を示すデバイス機能には、ハートビートセマフォ、デバイスタイマー (たとえば、ダウンロードで使用されるローカル `lbolt` など)、イベントカウンタがあります。デバイスから更新済みの予想可能値を読み取ると、進行が順調であるかどうか、ある程度確かなことがわかります。

- ドライバによる発行時に、発信要求 (送信ブロックまたはコマンド) にタイムスタンプを設定する。

  定期的な健全性チェックで、期限を過ぎていながら完了していない要求を検索できます。

- 予定された次回のチェックまでに完了すべき、デバイスに対するアクションを開始する。

  このアクションを割り込みにすると、デバイスの回路が割り込みのデリバリにまだ対応できるかどうかを確認するうえで理想的です。

第 4 章

SPARC: ドライバ強化のテストハーネス

Solaris 8 4/01 リリースの SPARC 版で、ドライバ強化のテストハーネスが新規に追加されました。Solaris デバイスドライバの作成方法についての詳細は、『*Writing Device Drivers*』を参照してください。

ドライバ強化テストハーネスは、Solaris デバイスドライバの開発ツールです。開発中のドライバがハードウェアにアクセスする際に、テストハーネスは模擬的なハードウェア障害 (シミュレーション) を投入します。この章では、テストハーネスの構成方法、エラー投入仕様 (*errdefs*) の作成方法、デバイスドライバでのテストの実行方法について説明します。

注 - 最新のマニュアルページを参照するには、`man` コマンドを使用してください。Solaris 8 Update リリースのマニュアルページには、「*Solaris 8 Reference Manual Collection*」には記載されていない新しい情報も提供されています。

テストハーネスについて

強化されたデバイスドライバは、可能性として考えられるハードウェアの障害に対し耐性があります。デバイスドライバの耐性は、ドライバの開発工程の一環としてテストしておく必要があります。こうしたテストでは、ハードウェアによくあるさまざまな障害に対し、ドライバはきちんと管理された方法で繰り返し対処できることを確認します。ドライバ強化テストハーネスを使用すると、ドライバの開発者はソフトウェアでハードウェア障害のシミュレーション (模擬的なテスト) が行えます。

テストハーネスはさまざまな DDI ルーチンに対するドライバからの呼び出しを横取りし、あたかもハードウェアが原因であるかのように、呼び出しの結果を破壊します。ハーネスでは、特定のレジスタへのアクセスに対する破壊を定義することができますし、よりランダムなタイプの破壊を定義することもできます。

注 - Solaris DDI/DKI に準拠するためには、ドライバは DDI ルーチンを使用して、すべての I/O アクセスを実行する必要があります。

テストハーネスでは、指定された作業負荷の実行時におけるすべてのレジスタアクセスとダイレクトメモリーアクセス (DMA)、割り込みの使用状況を追跡することによって、テストスクリプトが自動的に生成できます。生成されるスクリプトでは、その作業負荷が再実行されると同時に、アクセスのたびに一連の障害が投入されます。

ドライバのテスト担当者はさらにテストケースを追加して、一層不明確な障害経路を発生させなければなりません。また、生成されたスクリプトから重複しているテストケースを削除する必要もあります。

テストハーネスは `bofi` というデバイスドライバと、ユーザーレベルの 2 つのユーティリティである `th_define`(1M) と `th_manage`(1M) として実装されています。`bofi` は、`bus_ops` 障害投入 (fault injection) を意味しています。

テストハーネスは次の作業を行います。

- Solaris DDI サービスが標準に準拠して使用されていることを確認する
- プログラムされた I/O (PIO) と DMA 要求の破壊と割り込みに対する干渉を管理のもとで行い、ドライバが管理するハードウェアに発生する障害をシミュレートする
- 親の連結 (nexus) ドライバから通知される、CPU とデバイス間のデータ経路において、障害のシミュレーションを行う
- 指定された作業負荷の間、ドライバのアクセスを監視し、障害投入スクリプトを生成する

障害投入

ドライバ強化テストハーネスは、ドライバがハードウェアにアクセスするたびにそれを横取りし、必要に応じてアクセスを破壊します。この節では、ドライバの耐性テストで障害を作成するとき、理解しておく必要のある事柄について説明します。

データアクセス関数

Solaris デバイスはデバイス (devinfo) ツリーというツリー状の構造内で管理されます。devinfo ツリーの各ノードには、システム内にあるデバイスの特定のインスタンスに関する情報が保存されています。各末端ノードはデバイスドライバに対応しています。それ以外のノードは連結 (nexus) ノードといいます。通常は、1 つの連結 (nexus) が 1 つのバスを表します。バスノードは末端ドライバをバスへの依存関係から分離し、その結果、アーキテクチャ上独立したドライバが作成できます。

DDI 関数の多く (特にデータアクセス関数 (DAF)) は、結果としてバス連結ドライバに対する上位呼び出しになります。末端ドライバはハードウェアにアクセスすると、アクセスルーチンにハンドルを渡します。バス連結はハンドルの操作方法を知っており、要求に応じます。DDI 準拠のドライバはこれらの DDI アクセスルーチンだけを使用して、ハードウェアにアクセスします。テストハーネスはこれらの上位呼び出しが指定されたバス連結に到達する前に横取りします。データアクセスがドライバのテスト担当者で指定されている基準に一致すると、そのアクセスは破壊されます。データアクセスが基準に一致しなかった場合は、バス連結に渡され、通常どおりに処理されます。

ドライバは `ddi_map_regs_setup` (*dip*、*rset*、*ma*、*offset*、*size*、*handle*) 関数を使用して、アクセスハンドルを取得します。引数では、対応付け (マッピング) 対象の「オフボード」メモリーを指定します。ハンドルはドライバをバス階層構造の詳細から分離するためのものなので、対応付けされた I/O アドレスをドライバが参照するときは、返されたハンドルを使用する必要があります。したがって、返される対応付けされたアドレスである *ma* を直接使用しないようにしてください。対応付けされたアドレスを直接使用すると、DAF の機構を使用しないことになります。

プログラムされた I/O では、DAF の組み合わせは次のようになります。

- I/O からホストへ: `ddi_get`*X*(*handle*, *ma*) および `ddi_rep_get`*X*(*handle*, *buf*, *ma*, *repcnt*, *flag*)
- ホストから I/O へ: `ddi_put`*X*(*handle*, *ma*, *value*) および `ddi_rep_put`*X*()

X と *repcnt* は、転送されるバイト数です。*X* は 8 バイト、16 バイト、32 バイト、または 64 バイトのバス転送サイズです。

DMA には類似した、さらに充実した一群の DAF があります。

テストハーネスの設定

ドライバ強化テストハーネスは Solaris 開発者クラスタや全体ディストリビューションクラスタに含まれています。どちらの Solaris クラスタもインストールしていない場合は、プラットフォームに合ったテストハーネスパッケージを手作業でインストールする必要があります。

テストハーネスのインストール

テストハーネスパッケージ (SUNWftduu、SUNWftdur、および SUNWftdux) をインストールするときは、pkgadd(1M) を使用します。

パッケージの存在するディレクトリに移動し、スーパーユーザーとして次のように実行します。

```
# pkgadd -d . SUNWftduu SUNWftdur SUNWftdux
```

テストハーネスの構成

テストハーネスをインストールしたら、/kernel/drv/bofi.conf ファイルを編集して、ドライバに作用できるようにハーネスを構成します。テストハーネスの属性については、次の節の説明を参照してください。

ハーネスの構成が完了したら、システムを再起動してハーネスドライバを読み込みます。

テストハーネスの属性

テストハーネスの動作は、/kernel/drv/bofi.conf 構成ファイルに設定されている起動時の属性によって制御されます。

ハーネスを最初にインストールしたときに、次の属性を設定して、ドライバへの DDI アクセスをハーネスが横取りするようにします。

bofi-nexus	PCI バスなどのバス連結の種類
bofi-to-test	テスト中のドライバの名前

たとえば xyznetdrv という PCI バスネットワークドライバをテストする場合は、次の属性値を設定します。

```
bofi-nexus="pci"
bofi-to-test="xyznetdrv"
```

上記以外の他の属性は、PIO を使用している周辺装置からの読み書き、および DMA を使用している周辺装置とデータのやりとりをする、Solaris DDI データアクセスメカニズムの使用状況やハーネスの検査に関する属性です。

`bofi-range-check` この属性を設定すると、テストハーネスは PIO DAF に渡される引数の整合性が検査されます。

`bofi-ddi-check` この属性を設定すると、`ddi_map_regs_setup()` によって返される対応付けされたアドレスが DAF のコンテキスト以外では使用されていないことを、テストハーネスが確認します。

`bofi-sync-check` この属性を設定すると、テストハーネスは DMA 関数の使い方が正しいことを検証し、ドライバが標準に準拠して `ddi_dma_sync()` を使用していることを確認します。

ドライバのテスト

この節では、`th_define`(1M) コマンドと `th_manage`(1M) コマンドを使用して、障害を作成し投入する方法を説明します。

障害の作成

`th_define`(1M) ユーティリティは、errdef を定義するための、`bofi` デバイスドライバへのインタフェースを提供します。1 つの errdef が、デバイスドライバのハードウェアへのアクセスを破壊する方法に関する 1 つの仕様に対応しています。`th_define` のコマンド行引数によって、投入する障害の正確な種類を指定します。指定した引数によって矛盾のない errdef が定義される場合は、`th_define` プロセスは `bofi` ドライバに errdef を格納します。errdef で定められている基準に達

するまで、プロセスは停止状態になります。実際には、アクセスカウントがゼロ (0) になると、停止状態が終了します。

障害の投入

テストハーネスはデータアクセスのレベルで動作します。データアクセスには次のような特性があります。

- アクセスしているハードウェアの種類 (ドライバ名)
- アクセスしているハードウェアのインスタンス (ドライバインスタンス)
- テストを行っているレジスタセット
- 目標となるレジスタセットのサブセット
- 転送の方向 (読み取りまたは書き込み)
- アクセスの種類 (PIO または DMA)

テストハーネスはデータのアクセスを横取りし、該当する障害をドライバに投入します。 `th_define`(1M)コマンドで指定された errdef によって、次の情報がエンコードされます。

- テストを行っているドライバインスタンスとレジスタセット(`–n` *name*、 `–i` *instance*、および `–r` *reg_number*)
- 破壊の対象となるレジスタセットのサブセット。レジスタセット内のオフセットとそのオフセットからの長さを指定することで、このサブセットを指定します (`–l` *offset* [*len*])
- 横取りするアクセスの種類:
  `log`、`pio`、`dma`、`pio_r`、`pio_w`、`dma_r`、`dma_w`、`intr` (`–a` *acc_types*)
- 障害が発生するアクセスの数 (`–c` *count* [*failcount*])
- 該当するアクセスに適用する破壊の種類 (`–o` *operator* [*operand*]):
  - データを固定値に置き換える (`EQUAL`)
  - データをビットごとに操作する (`AND`、`OR`、`XOR`)
  - 転送を無視する (ホストから I/O へのアクセス `NO_TRANSFER`)
  - 割り込みを失う、遅らせる、または偽の割り込みを投入する (`LOSE`、`DELAY`、`EXTRA`)

`–a` *acc_chk* オプションを使用すると、errdef でフレームワーク障害をシミュレートできます。

障害投入プロセス

障害を投入するプロセスには、次の 2 つの段階があります。

1. th_define コマンドを使用して **errdef** を作成します。

   テストの定義を bofi ドライバに渡して、errdef を作成します。bofi ドライバは定義を保持するので、 th_manage(1M) を使用して定義にアクセスできます。

2. 作業負荷を作成し、th_manage を使用して **errdef** を起動し管理します。

   th_manage(1M) コマンドは、bofi ハーネスドライバが認知するさまざまな ioctl に対するユーザーインタフェースです。th_manage はドライバ名とインスタンスのレベルで動作します。th_manage には、アクセスハンドルを一覧表示する get_handles、errdef を起動する start、errdef を中止する stop などのコマンドがあります。

   errdef を起動すると、対象となるデータのアクセスに障害が発生します。th_manage ユーティリティでは、errdef の現在の状態を表示する broadcast や errdef をクリアする clear_errors などのコマンドがサポートされています。

   詳細は、 th_define(1M) および th_manage(1M) のマニュアルページを参照してください。

テストハーネスの警告

テストハーネスは、警告メッセージを次の方法で処理するように設定できます。

- 警告メッセージをコンソールに書き込む
- 警告メッセージをコンソールに書き込んでから、システムをパニック状態にする

2 つめの方法を取ると、問題の根本原因を突き止めやすくなります。

bofi-range-check 属性値を warn に設定すると、ドライバによる DDI 関数の範囲違反をハーネスが検出したときに、次のメッセージが出力されます(panic に設定されている場合はパニック状態になります)。

```
ddi_getX() out of range addr %x not in %x
ddi_putX() out of range addr %x not in %x
ddi_rep_getX() out of range addr %x not in %x
ddi_rep_putX() out of range addr %x not in %x
```

X は 8、16、32、または 64 です。

ハーネスが 1000 を超える割り込みを挿入するように要求された場合には、ドライバが割り込みジャバーを検出しなかった場合に、次のメッセージが出力されます。

```
undetected interrupt jabber - %s %d
```

スクリプトによるテストプロセスの自動化

th_define ユーティリティでアクセスログタイプを指定して、障害投入テストスクリプトが作成できます。

```
# th_define –n name –i instance –a log [–e fixup_script]
```

th_define はインスタンスをオフラインにし、またオンラインに戻します。次に、修正スクリプト (*fixup_script*) が記述している作業負荷を実行し、ドライバインスタンスが行う I/O アクセスをログに記録します。

修正スクリプトは省略可能ないくつかの引数を指定されて 2 度呼び出されます (インスタンスがオフラインにされる直前と、オンラインに戻された後の 2 回です)。次の変数が、呼び出された実行可能ファイルの環境に渡されます。

DRIVER_PATH	インスタンスのデバイスパス
DRIVER_INSTANCE	ドライバのインスタンス番号
DRIVER_UNCONFIGURE	インスタンスがこれからオフラインになるときは、1 に設定
DRIVER_CONFIGURE	インスタンスがオンラインに戻されたときは、1 に設定

通常、修正スクリプトはテスト中のデバイスがオフラインとなってよい状態 (未構成)にあるか、あるいはエラー投入が適切な状態 (構成済み、エラーなし、および作業負荷を処理中) であることを確認します。次に、ネットワークドライバ用の最小限のスクリプトの例を示します。

```
#!/bin/ksh
driver=xyznetdrv
ifnum=$driver$DRIVER_INSTANCE
```

(続く)

```
if [[ $DRIVER_CONFIGURE = 1 ]]; then
   ifconfig $ifnum plumb
   ifconfig $ifnum ...
   ifworkload start $ifnum
elif [[ $DRIVER_UNCONFIGURE = 1 ]]; then
   ifworkload stop $ifnum
   ifconfig $ifnum down
   ifconfig $ifnum unplumb
fi
exit $?
```

注 - ifworkload は作業負荷をバックグラウンドタスクとして開始する必要があります。障害の投入は、修正スクリプトがテスト中のドライバを構成してオンラインに戻した後で行われます (DRIVER_CONFIGURE は 1 に設定されます)。

–e *fixup_script* オプションを指定する場合は、コマンド行の最後に指定しなければなりません。指定しないと、デフォルトのスクリプトが使用されます。デフォルトのスクリプトは、テスト中のデバイスのオフラインとオンラインを繰り返し試行します。従って、作業負荷は、ドライバのアタッチとデタッチの処理となります。

出来上がったログは、自動障害投入テストに適した、いくつかの実行可能なスクリプトに変換されます。これらのスクリプトは、*driver*.test.*id* という名前で、現在のディレクトリの下のサブディレクトリに作成されます。スクリプトはドライバに障害を 1 度に 1 つずつ投入します。同時に、修正スクリプトが記述している作業負荷を実行します。

ドライバのテスト担当者は、テスト自動化プロセスで生成された errdef を大部分制御できます。詳細は、 th_define(1M) のマニュアルページを参照してください。

テスト担当者がテストスクリプトに適した作業負荷の範囲を選択すれば、ハーネスはドライバの強化の多くの局面をテストできます。しかし、すべての局面を網羅するためには、テスト担当者は別のテストケースを手作業で作成しなければならなくなる場合があります。これらのケースは、テストスクリプトに追加します。テストを適切な時間内に完了させるには、テスト担当者は重複しているテストケースを手作業で削除する必要がある場合があります。

自動テストプロセス

自動テストのプロセスは、次のようになります。

1. ドライバの何をテストするかを確認します。

   次のようにドライバがハードウェアに作用する部分は、すべてテストします。

   - アタッチとデタッチ
   - スタック下の接続と切り離し
   - 通常のデータ転送
   - 文書化されたデバッグモード

   使用モードごとに、作業負荷スクリプト (*fixup_script*) を生成する必要があります。

2. 使用モードごとに、デバイスの構成と構成の解除を行い、作業負荷を作成し終了する、実行可能なプログラム (***fixup_script***) を作成します。

3. **errdef** によってアクセスの種類を `-a log` と指定し、`th_define` を実行します。

4. ログがいっぱいになるまで待ちます。

   ログには、`bofi` ドライバの内部バッファのダンプが入っています。このデータはスクリプトの最初に記載されます。

   ログの作成には数秒から数分かかるので、`th_manage broadcast` コマンドを使用して進行状況を検査します。

5. 作成されたテストディレクトリに移動し、マスターテストスクリプトを実行します。

   マススクリプトは、生成されたテストスクリプトを順次実行します。レジスタセットごとに、テストスクリプトが生成されます。

6. 分析結果を保存します。

   `success (corruption reported)` や `success (corruption undetected)` などの正常に終了したテスト結果は、テスト中のドライバが正常に動作していることを示しています。

   出力に `test not triggered` という失敗が 2、3 含まれても問題ではありませんが、そうした失敗がそれ以上になった場合には、テストがうまくいっていないことを示します。テストスクリプトが生成されたときと同じレジスタにドライバがアクセスしていないときは、こうした失敗が発生することがあります。

7. ドライバの複数のインスタンスに対して同時にテストを実行し、エラーパスのマルチスレッド化をテストします。

   たとえば、th_define コマンドごとに、テストスクリプトとマスタースクリプトが入ったディレクトリを作成します。

```
# th_define -n xyznetdrv -i 0 -a log -e script
# th_define -n xyznetdrv -i 1 -a log -e script
```

   マスタースクリプトを作成したら、それらを同時に実行します。

注 - 生成されたスクリプトでは、ログ対象の errdef がアクティブであった間に記録されたログ内容に基づいた障害投入のシミュレーションだけが生成されます。作業負荷を定義するときは、必要な結果がログに記録されることを確認し、出来上がったログや障害投入の仕様も分析します。出来上がったテストスクリプトが行ったハードウェアのアクセスの範囲が、要望どおりであることを確認する必要があります。

第 5 章

ネットワークデバイス用のドライバ

Solaris 8 10/00 リリースで、Generic LAN ドライバが新規に追加されました。

Generic LAN ドライバ (GLD) により、Solaris ネットワークドライバに STREAMS および Data Link Provider Interface (DLPI) 機能の大部分が実装されます。

Solaris 8 10/00 より前のリリースでは、GLD モジュールを利用できるのは、Solaris Intel 版のネットワークドライバに限定されていました。Solaris 8 10/00 からは、Solaris SPARC 版のネットワークドライバでも GLD を利用できます。

詳細は、次のマニュアルページを参照してください。`gld(7D)`、 `dlpi(7P)`、`gld(9E)`、 `gld(9F)`、 `gld_mac_info(9S)`、 `gld_stats(9S)`

注 - 最新のマニュアルページを参照するには、`man` コマンドを使用してください。Solaris 8 Update リリースのマニュアルページには、「*Solaris 8 Reference Manual Collection*」には記載されていない新しい情報も提供されています。

Generic LAN ドライバ (GLD) の概要

GLD は、Solaris ローカルエリアネットワークデバイスドライバをサポートする、クローン化可能でロード可能なマルチスレッド化カーネルモジュールです。Solaris のローカルエリアネットワーク (LAN) デバイスドライバは、DLPI を使用してネットワークプロトコルスタックと通信する、STREAMS ベースのドライバです。これらのプロトコルスタックは、ネットワークドライバを使用して、ローカルエリアネットワーク上でパケットを送受信します。ネットワークデバイスドライバは、

DDI/DKI 仕様、STREAMS 仕様、DLPI 仕様、およびデバイスそのもののプログラム式インタフェースの要件を満たすように実装する必要があります。

GLD には、Solaris LAN ドライバに必要な STREAMS および DLPI 機能の大部分が実装されています。いくつかの Solaris ネットワークドライバは GLD を使用して実装されています。

GLD を使用して実装された Solaris ネットワークドライバは、2 種類の部分で構成されます。STREAMS および DLPI インタフェースを扱う汎用コンポーネント、および特定のハードウェアデバイスを扱うデバイス固有のコンポーネントです。デバイス固有のモジュールは、GLD モジュール (/kernel/misc/gld に存在する) への依存を意味し、ドライバの attach(9E) 関数内から GLD に自身を登録します。正常に読み込まれると、ドライバは DLPI に準拠することになります。ドライバのデバイス固有の部分は、データを受信するとき、または GLD のサービスが必要になったときに、gld(9F) 関数を呼び出します。GLD は、デバイス固有のドライバが GLD に登録するときに提供したポインタを介して、デバイス固有のドライバの gld(9E) エントリポイントを呼び出します。gld_mac_info(9S) 構造体が GLD とデバイス固有のドライバ間のメインデータインタフェースです。

GLD 機能が現在サポートしているデバイスのタイプは、DL_ETHER、DL_TPR、および DL_FDDI です。GLD ドライバは、フルフォームの MAC 層パケットを処理することが想定されており、論理リンク制御 (LLC) の処理は想定されていません。

状況によっては、GLD 機能を使用しないで、DLPI に完全に準拠したドライバを実装する場合もあるでしょう。これには、ISO 8802 スタイル (IEEE 802) ではない LAN デバイスの場合、または GLD ではサポートされないデバイスタイプまたは DLPI サービスが必要な場合、などがあります。

タイプ DL_ETHER：Ethernet V2 および ISO 8802-3 (IEEE 802.3)

タイプとして DL_ETHER が指定されたデバイスについては、GLD は Ethernet V2 と ISO 8802-3 (IEEE 802.3) の両方のパケット処理をサポートします。Ethernet V2 によって、データリンクサービスのユーザーは、プロバイダのプロトコルを特に知らなくても、さまざまな適合するデータリンクサービスプロバイダにアクセスして利用することができます。サービスアクセスポイント (SAP) は、ユーザーがサービスプロバイダと通信するときに通過するポイントです。

0 ～ 255 の範囲の SAP 値と結合された Stream は、ユーザーが 8802-3 モードの使用を求めていることを示します。DL_BIND_REQ の SAP フィールド値がこの範囲内

にある場合、GLD はその Stream の各後続の DL_UNITDATA_REQ メッセージ長 (14 バイトのメディアアクセス制御 (MAC) ヘッダーは含まない) を計算し、MAC フレームのヘッダーの type フィールドにその長さの 8802-3 フレームを送信します。この長さは 1500 を超えてはなりません。

0 ～ 1500 の範囲の type フィールドを持つメディアから受信されたフレームはすべて、8802-3 フレームとみなされ、8802-3 モードのオープンなすべての Stream (0 ～ 255 の範囲の SAP 値に結合された Stream) にルーティングされます。8802-3 モードの Stream が複数ある場合、着信フレームがコピーされて、該当するそれぞれの Stream にルーティングされます。

1500 より大きい SAP 値 (Ethernet V2 モード) と結合された Stream は、Ethernet MAC ヘッダーの type 値が、Stream と結合された SAP 値と正確に一致する着信パケットを受信します。

タイプ DL_TPR および DL_FDDI：SNAP 処理

メディアタイプ DL_TPR および DL_FDDI については、GLD は 255 より大きい SAP 値と結合された Stream に、最小限の SNAP (Sub-Net Access Protocol) 処理を実装します。0 ～ 255 の範囲の SAP 値は LLC SAP 値であり、メディアパケットフォーマットによって通常どおりに伝送されます。255 より大きい SAP 値には、LLC ヘッダーに従属し、16 ビット Ethernet V2 スタイルの SAP 値を伝送する SNAP ヘッダーが必要です。

SNAP ヘッダーは、宛先 SAP 0xAA が指定された LLC ヘッダーとして伝送されます。SAP 値が 255 より大きい発信パケットには、GLD は LLC+SNAP ヘッダーを作成します。このヘッダは、必ず次のような形式になります。

```
AA AA 03 00 00 00 XX XX
```

XX XX は、Ethernet V2 スタイルの type に対応する 16 ビットの SAP を表します。これがサポートされる唯一の SNAP ヘッダーのクラスです。ゼロ以外の OUI フィールドおよび 03 以外の LLC 制御フィールドは、SAP 0xAA を持つ LLC パケットとみなされます。これ以外の SNAP フォーマットを使用するクライアントは、LLC を使用し、SAP 0xAA に結合する必要があります。

着信パケットは上記のフォーマットに準拠しているかどうかが検証されます。このフォーマットに準拠するパケットは、そのパケットの 16 ビット SNAP タイプに結合された Stream と照合されるとともに、LLC SNAP SAP 0xAA と一致するとみなされます。

LLC SAP として受信されたパケットは、メディアタイプ DL_ETHER の箇所で説明したように、LLC SAP と結合されているすべての Stream に渡されます。

タイプ DL_TPR：ソースルーティング

タイプ DL_TPR デバイスについては、GLD は最小限のソースルーティングサポートを実装します。ソースルーティングにより、ブリッジングされたメディア上でパケットを送信するステーションは、ネットワーク上でパケットがたどるルートを決定づけるルーティング情報を (パケットの MAC ヘッダーに) 指定できます。

GLD が提供するソースルーティングは、ルートを学習し、可能性のある複数のルートについて情報を要求するように指示し、要求に応答し、使用できるルートの中から選択する機能があります。さらに、発信パケットの MAC ヘッダーに Routing Information Fields を追加したり、着信パケット内のこのようなフィールドを認識したりします。

GLD のソースルーティングは、ISO 8802-2 (IEEE 802.2) の Section 9 で指定されている Route Determination Entity (RDE) を完全に実装するわけではありません。しかし、同じ (または縮約された) ネットワークに存在し得るソースルーティング実装方式と相互運用を行うように設計されています。

Style 1 および Style 2 の DLPI プロバイダ

GLD は、Style 1 および Style 2 の DLPI プロバイダを両方とも実装します。Physical Point of Attachment (PPA) は、システムが物理通信メディアと接続するポイントです。その物理メディア上の通信はすべて、PPA を通過します。Style 1 のプロバイダは、オープンされているメジャー/マイナーデバイスに基づいて、特定の PPA に Stream を接続します。Style 2 のプロバイダは、DLS ユーザーが DL_ATTACH_REQ を使用して必要な PPA を明示的に特定するよう要求します。その場合、open(9E) がユーザーと GLD 間の Stream を作成し、その後、DL_ATTACH_REQ が所定の PPA とその Stream を対応付けます。Style 2 はゼロのマイナー番号で示されます。マイナー番号がゼロ以外のデバイスノードがオープンしている場合、Style 1 が示され、対応する PPA はマイナー番号から 1 を差し引いたものになります。Style 1 と Style 2 が両方とも open の場合、デバイスはクローン化されます。

実装される DLPI プリミティブ

GLD は、いくつかの DLPI プリミティブを実装します。DL_INFO_REQ プリミティブは、DLPI Stream に関する情報を要求します。メッセージは 1 つの M_PROTO メッセージブロックで構成されます。GLD はこの要求に対する DL_INFO_ACK 応答で、gld_register() に渡された gldm_mac_info(9S) 構造体に GLD ベースのドライバが指定した情報に基づいて、デバイスに依存する値を返します。ただし、GLD は、すべての GLD ベースのドライバに代わって次の値を返します。

- バージョンは DL_VERSION_2 です。
- サービスモードは DL_CLDLS です。GLD はコネクションレスモードのサービスを実装します。
- プロバイダスタイルは、Stream がどのようにオープンされたかにより、DL_STYLE1 または DL_STYLE2 になります。
- オプションの Quality Of Service (QOS) サポートはありません。QOS フィールドはゼロです。

注 - DLPI 仕様と異なり、GLD は、Stream が PPA に接続される前であっても、DL_INFO_ACK でデバイスの正確なアドレス長およびブロードキャストアドレスを返します。

PPA と Stream を対応付けるために、DL_ATTACH_REQ プリミティブが使用されます。この要求は、Style 2 の DLS プロバイダが通信を行う物理メディアを特定するために必要です。完了すると、状態が DL_UNATTACHED から DL_UNBOUND に変わります。メッセージは 1 つの M_PROTO メッセージブロックで構成されます。ドライバを Style 1 モードで使用している場合は、この要求は許可されません。Style 1 を使用してオープンされた Stream は、オープンの完了時には PPA にすでに接続されているからです。

DL_DETACH_REQ プリミティブは、Stream から PPA を切り離すことを要求します。これが認められるのは、Stream が Style 2 を使用してオープンされた場合だけです。

DL_BIND_REQ プリミティブおよび DL_UNBIND_REQ プリミティブは、DLSAP と Stream の結合および結合解除を行います。1 つの Stream と対応付けられた PPA は、Stream の DL_BIND_REQ の処理が完了する前に初期化を完了します。複数の Stream を同じ SAP に結合できますが、このような Stream はそれぞれ、その SAP で受信したパケットのコピーを受け取ります。

DL_ENABMULTI_REQ プリミティブおよび DL_DISABMULTI_REQ プリミティブは、個々のマルチキャストグループアドレスの受け付けを可能または不可能にしま

す。アプリケーションまたは他の DLS ユーザーは、これらのプリミティブを繰り返し使用して 1 組のマルチキャストアドレスを Stream 単位で作成または変更することが可能です。これらのプリミティブが受け入れられるように、Stream を PPA に接続する必要があります。

DL_PROMISCON_REQ プリミティブおよび DL_PROMISCOFF_REQ プリミティブは、物理レベルまたは SAP レベルのどちらかで、プロミスキュアス (promiscuous) モードを Stream 単位で有効または無効にします。DL Provider は、DL_DETACH_REQ または DL_PROMISCOFF_REQ を受信するまで、または Stream がクローズされるまで、そのメディアで受信したすべてのメッセージを DLS ユーザーにルーティングします。物理レベルのプロミスキュアスの受信を、そのメディア上の全パケットに指定することも、マルチキャストパケットに限定して指定することもできます。

注 - これらのプロミスキュアスモードのプリミティブが受け入れられるように、Stream を PPA に接続する必要があります。

DL_UNITDATA_REQ プリミティブは、コネクションレス型転送でデータを送信する場合に使用します。これは未承認のサービスなので、配信の保証はありません。メッセージは 1 つの M_PROTO メッセージブロックとそれに続く 1 つ以上の M_DATA ブロック (1 バイト以上のデータを含む) で構成されます。

DL_UNITDATA_IND タイプは、パケットを受信してアップストリームに渡す場合に使用します。パケットは、プリミティブを DL_UNITDATA_IND に設定した M_PROTO メッセージに格納されます。

DL_PHYS_ADDR_REQ プリミティブは、Stream に接続された PPA にその時点で対応付けられている MAC アドレスを要求します。このアドレスは、DL_PHYS_ADDR_ACK プリミティブによって返されます。Style 2 を使用している場合、このプリミティブが有効なのは DL_ATTACH_REQ が成功した場合に限られます。

DL_SET_PHYS_ADDR_REQ プリミティブは、Stream に接続された PPA にその時点で対応付けられている MAC アドレスを変更します。このプリミティブは、現在および将来にわたりこのデバイスに接続された他のすべての Stream に作用します。いったん変更すると、現在オープンしている、または今後オープンされてこのデバイスに接続されるすべての Stream が、この新しい物理アドレスを取得します。新しい物理アドレスは、このプリミティブを使用して再び物理アドレスを変更するまで、またはドライバがリロードされるまで有効です。

注 - スーパーユーザーは、他の Stream が同じ PPA に結合されている際には PPA の物理アドレスを変更することができます。

DL_GET_STATISTICS_REQ プリミティブは、Stream に接続された PPA に対応する統計情報を収めた DL_GET_STATISTICS_ACK 応答を要求します。DL_ATTACH_REQ を使用して Style 2 の Stream を特定の PPA に接続しておかないと、このプリミティブは成功しません。

実装される ioctl 関数

GLD は、以下で説明する ioctl *ioc_cmd* 関数を実装します。認識できない ioctl コマンドを受信した GLD は、gld(9E) に記述されているように、デバイス固有のドライバの gldm_ioctl() ルーチンにそのコマンドを渡します。

DLIOCRAW ioctl 関数は、snoop(1M) コマンドをはじめ、一部の DLPI アプリケーションで使用されます。DLIOCRAW コマンドは、Stream を raw モードにし、それによってドライバが、着信パケットの報告に通常使用される DL_UNITDATA_IND 形式に変換するのではなく、M_DATA メッセージで MAC レベルの着信パケット全体をアップストリーム送信するようにします。パケット SAP のフィルタリングは、raw モードの Stream にも実行されます。Stream のユーザーがすべての着信パケットの受け取りを希望する場合は、適切なプロミスキュアスモード (複数可) も選択しなければなりません。raw モードを正しく選択しているアプリケーションは、フルフォーマットのパケットを伝送する M_DATA メッセージとしてドライバに送ることもできます。DLIOCRAW は引数を使用しません。raw モードが有効になった Stream は、クローズされるまでそのモードのままです。

GLD ドライバの要件

GLD ベースのドライバには、ヘッダーファイル <sys/gld.h> が組み込まれていなければなりません。

また、GLD ベースのドライバには次の宣言が含まれていることも必要です。

```
char _depends_on[] = "misc/gld";
```

GLD はデバイス固有のドライバのために、open(9E) および close(9E) 関数の他に、要求される STREAMS の put(9E) および srv(9E) 関数を実装します。さらに、ドライバ用に getinfo(9E) 関数も実装します。

module_info(9S) 構造体の mi_idname 要素は、ドライバ名を指定する文字列です。ファイルシステムに存在するドライバモジュールの名前と正確に一致しなければなりません。

読み取り側の qinit(9S) 構造体では、次の要素を指定する必要があります。

```
qi_putp             NULL

qi_srvp             gld_rsrv

qi_qopen            gld_open

qi_qclose           gld_close
```

書き込み側の qinit(9S) 構造体では、次の要素を指定する必要があります。

```
qi_putp             gld_wput

qi_srvp             gld_wsrv

qi_qopen            NULL

qi_qclose           NULL
```

dev_ops(9S) 構造体の devo_getinfo 要素では、getinfo(9E) ルーチンとして gld_getinfo を指定する必要があります。

ドライバの attach(9E) 関数は、ハードウェア固有のデバイスドライバと GLD 機能を対応付け、デバイスとドライバを使用できるように準備を整える一切の作業を行います。

attach(9E) 関数は、gld_mac_alloc() を使用して gld_mac_info(9S) (macinfo) 構造体を割り当てます。ドライバは通常、1 つのデバイスについて、macinfo 構造体で定義されているより多くの情報を保存しなければならないので、必要な追加分のデータ構造体を割り当て、gld_mac_info(9S) 構造体の gldm_private メンバーにそのデータ構造体へのポインタを保存する必要があります。

attach(9E) ルーチンは、gld_mac_info(9S) に記述されているように macinfo 構造体を初期化し、その後 gld_register() を呼び出して、ドライバと GLD モジュールを結びつけなければなりません。ドライバは必要に応じてレジスタをマップ (対応付け) し、完全に初期化して、gld_register() を呼び出す前に割り込みを受け付けるように準備する必要があります。attach(9E) 関数が割り込みを追加することはあっても、デバイスに割り込みを発生させてはなりません。ドライバ

は、ハードウェアが確実に静止しているように、gld_register() を呼び出す前に、ハードウェアをリセットする必要があります。gld_register() の呼び出し前に、デバイスが起動されたり、割り込みが発生するような状態になったりしてはなりません。これは、gld(9E) に記載されているように、後に GLD がドライバの gldm_start() エントリポイントを呼び出した時点で行います。gld_register() が正常に完了すると、GLD はいつでも gld(9E) エントリポイントを呼び出すことができます。

attach(9E) ルーチンは、gld_register() が正常に完了した場合は、DDI_SUCCESS を返します。gld_register() がエラーになった場合は、DDI_FAILURE を返し、gld_register() を呼び出す前に割り当てられたあらゆるリソースを、attach(9E)ルーチンで割り当て解除する必要があります。そして、DDI_FAILURE を返します。どのような状況でも、エラーの起きた macinfo 構造体を再利用することがあってはなりません。gld_mac_free() を使用して、割り当てを解除する必要があります。

detach(9E) 関数は、GLD からドライバの登録を解除しようとします。これは、gld(9F) に記載されている gld_unregister() を呼び出すことによって行います。detach(9E) ルーチンは、ddi_get_driver_private(9F) を使用することによって、デバイスの専用データから必要な gld_mac_info(9S) 構造体に対するポインタを取得できます。gld_unregister() は、一定の条件をチェックして、ドライバを切り離せないかどうかを調べます。このチェックに失敗すると、gld_unregister() は DDI_FAILURE を返します。その場合、ドライバの detach(9E) ルーチンはデバイスを動作状態にしたまま、DDI_FAILURE を返さなければなりません。

チェックが成功すると、gld_unregister() は、(必要に応じてドライバの gldm_stop() ルーチンを呼び出して) デバイスの割り込み中止を確認し、ドライバを GLD フレームワークから切り離し、DDI_SUCCESS を返します。この場合、detach(9E) ルーチンは割り込みを削除し、attach(9E) ルーチンに割り当てられていたすべてのデータ構造を (macinfo 構造体の割り当てを解除する gld_mac_free() を使用して) 割り当て解除したうえで、DDI_SUCCESS を返さなければなりません。gld_mac_free() を呼び出す前に割り込みを削除することが重要です。

ネットワーク統計

Solaris ネットワークドライバは統計変数を実装しなければなりません。一部のネットワーク統計については、GLD 本体で記録されますが、他のネットワーク統計は

GLD ベースのドライバごとにカウントする必要があります。GLD は、GLD ベースのドライバによるネットワークドライバ統計の標準セットのレポートをサポートします。統計は、 kstat(7D) および kstat(9S) メカニズムを使用して、GLD が報告します。DLPI コマンドの DL_GET_STATISTICS_REQ を使用して、現在の統計カウンタを検索することもできます。統計は特に指定が無い限り、いずれも符号無しで維持され、32 ビットです。

GLD が維持および報告する統計は、次のとおりです。

rbytes64	インタフェース上で正常に受信したトータルのバイト数 (64 ビット)
rbytes	インタフェース上で正常に受信したトータルのバイト数
obytes64	インタフェース上で送信を要求したトータルのバイト数 (64 ビット)
obytes	インタフェース上で送信を要求したトータルのバイト数
ipackets64	インタフェース上で正常に受信したトータルのパケット数 (64ビット)
ipackets	インタフェース上で正常に受信したトータルのパケット数
opackets64	インタフェース上で送信を要求したトータルのパケット数 (64 ビット)
opackets	インタフェース上で送信を要求したトータルのパケット数
multircv	グループおよび機能アドレスを含め、正常に受信したマルチキャストパケット (long)
multixmt	グループおよび機能アドレスを含め、送信を要求したマルチキャストパケット (long)
brdcstrcv	正常に受信したブロードキャストパケット (long)

brdcstxmt	送信を要求したブロードキャストパケット (long)
unknowns	どの Stream も受け付けなかった有効な受信パケット (long)
noxmtbuf	送信バッファが使用中だった、または送信バッファを割り当てることができなかったために、出力で廃棄されたパケット (long)
blocked	キューがフロー制御にされていたために、受信パケットを Stream に格納できなかった回数 (long)
xmtretry	リソース不足のために送信が遅延された後で、送信が再試行された回数 (long)
promisc	インタフェースの現在の「プロミスキュアス」状態 (文字列)

デバイス依存型のドライバは、次の統計情報をカウントし、専用のインスタンス別構造体でその記録を保管します。統計情報を報告するように求められた GLD は、gld(9E) に記述されているように、ドライバの gldm_get_stats() エントリポイントを呼び出して、gld_stats(9S) 構造体のデバイス固有の統計情報を更新します。GLD は次に、以下に示す名前付き統計変数を使用して、更新された統計情報を報告します。

ifspeed	ビット/秒で表したインタフェースの現在の帯域幅の見積り (64ビット)
media	デバイスが現在使用しているメディアタイプ (文字列)
intr	割り込みハンドラが呼び出され、割り込みが要求された回数 (long)
norcvbuf	受信バッファを割り当てることができなかったために、有効な着信パケットが廃棄されたことが判明している回数 (long)
ierrors	エラーが含まれていたために処理できなかったトータルの受信パケット (long)

oerrors	エラーが原因で正常に送信されなかったトータルのパケット (long)
missed	受信時にハードウェアによってドロップされたことが判明しているパケット (long)
uflo	送信時の FIFO アンダーフロー回数 (long)
oflo	受信時の受信側オーバーフロー回数 (long)

次の統計グループは、DL_ETHER タイプのネットワークに適用されます。これらの統計は、前述のように、そのタイプのデバイス固有のドライバによって維持されます。

align_errors	フレーム指示エラーの起きた受信パケット (オクテットの整数ではない) (long)
fcs_errors	CRC エラーの起きた受信パケット (long)
duplex	インタフェースの現在の二重モード (文字列)
carrier_errors	送信試行時にキャリアが失われたか、または全く検出されなかった回数 (long)
collisions	送信時の Ethernet の衝突 (long)
ex_collisions	送信時に発生した衝突が多すぎて、送信エラーとなったフレーム (long)
tx_late_collisions	遅れて (512 ビットタイム) 発生した送信衝突の回数 (long)
defer_xmts	衝突は発生しなかったが、メディアがビジー状態だったために最初の送信試行が遅延されたパケット (long)
first_collisions	1 回の衝突だけで正常に送信されたパケット
multi_collisions	複数の衝突が起きたが正常に送信されたパケット
sqe_errors	SQE テストエラーが報告された回数
macxmt_errors	キャリアおよび衝突エラー以外の送信 MAC エラーが検出されたパケット

macrcv_errors	align_errors、fcs_errors、toolong_errors 以外の、MAC エラーが起きた受信パケット
toolong_errors	許容される最大長を超えていた受信パケット
runt_errors	許容される最小長に満たなかった受信パケット (long)

次の統計グループは、DL_TPR タイプのネットワークに適用されます。これらの統計は、前述のように、そのタイプのデバイス固有のドライバによって維持されます。

line_errors	非データビットまたは FCS エラーのあった受信パケット
burst_errors	ハーフビットタイマー 5 回の間に変位が発生しなかったことが検出された回数
signal_losses	リングでシグナル損失状態が検出された回数
ace_errors	AMP フレームの介入の無い、別の同様の SMP フレームが続く、A、C が共に 0 の AMP または SMP フレームの数
internal_errors	ステーションが内部エラーを認識した回数
lost_frame_errors	送信中に TRR タイマが期間満了した回数
frame_copied_errors	このステーション宛てのフレームが、FS フィールドの「A」ビットを 1 に設定して受信された回数
token_errors	アクティブモニターとして動作しているステーションが、トークンの送信を必要とするエラー状態を認識した回数
freq_errors	着信信号の周波数が予期された周波数と異なっていた回数

次の統計グループは、DL_FDDI タイプのネットワークに適用されます。これらの統計は、前述のように、そのタイプのデバイス固有のドライバによって維持されます。

mac_errors	別の MAC ではエラーが検出されず、この MAC ではエラーが検出されたフレーム

mac_lost_errors	フレームが取り除かれるなどのフォーマットエラーが起きた受信フレーム数
mac_tokens	受信トークン数 (制限無しと制限付きの合計)
mac_tvx_expired	TVX が期間満了になった回数
mac_late	この MAC がリセットされて以後、またはトークンの受信以後、TRT が期間満了になった回数
mac_ring_ops	リングが「Ring Not Operational」状態から「Ring_Operational」状態になった回数

宣言とデータ構造

gld_mac_info 構造体

GLD の MAC 情報 (gld_mac_info) 構造体は、デバイス固有のドライバと GLD 間のメインデータインタフェースです。この構造体には、GLD が必要とするデータ、およびオプションの追加のドライバ固有情報の構造体に対するポインタが含まれます。

gld_mac_info 構造体は、gld_mac_alloc() を使用して割り当て、gld_mac_free() を使用して割り当てを解除します。ドライバ側でこの構造の長さを想定してはなりません。長さは Solaris の各リリース、各 GLD、またはその両方によって異なる可能性があるためです。このマニュアルには記載されていない GLD 専用の構造体のメンバーをデバイス固有のドライバで設定したり読み取ったりしてはなりません。

gld_mac_info(9S) 構造体には、次のフィールドがあります。

```
caddr_t             gldm_private;             /* Driver private data */
int                 (*gldm_reset)();          /* Reset device */
int                 (*gldm_start)();          /* Start device */
int                 (*gldm_stop)();           /* Stop device */
int                 (*gldm_set_mac_addr)();   /* Set device phys addr */
int                 (*gldm_set_multicast)();  /* Set/delete multicast addr */
int                 (*gldm_set_promiscuous)(); /* Set/reset promiscuous mode */
int                 (*gldm_send)();           /* Transmit routine */
uint_t              (*gldm_intr)();           /* Interrupt handler */
int                 (*gldm_get_stats)();      /* Get device statistics */
int                 (*gldm_ioctl)();          /* Driver-specific ioctls */
char                *gldm_ident;              /* Driver identity string */
uint32_t            gldm_type;                /* Device type */
uint32_t            gldm_minpkt;              /* Minimum packet size */
```

```
                                                    /* accepted by driver */
uint32_t             gldm_maxpkt;                 /* Maximum packet size */
                                                    /* accepted by driver */
uint32_t             gldm_addrlen;                /* Physical address length */
int32_t              gldm_saplen;                 /* SAP length for DL_INFO_ACK */
unsigned char        *gldm_broadcast_addr;        /* Physical broadcast addr */
unsigned char        *gldm_vendor_addr;           /* Factory MAC address */
t_uscalar_t          gldm_ppa;                    /* Physical Point of */
                                                    /* Attachment (PPA) number */
dev_info_t           *gldm_devinfo;               /* Pointer to device's */
                                                    /* dev_info node */
ddi_iblock_cookie_t  gldm_cookie;                 /* Device's interrupt */
                                                    /* block cookie */
```

デバイスドライバは、gld_mac_info 構造体のメンバーを認識できます。

gldm_private　この構造体のメンバーは、デバイス固有のドライバ専用であり、GLD が使用したり変更したりすることはありません。これは、従来専用データに対するポインタとして使用されたもので、ドライバが定義し、またドライバが割り当てたインスタンス別データ構造体を指し示します。

次の構造体メンバーのグループは、gld_register() を呼び出す前にドライバで設定しなければなりません。以後は、ドライバ側で変更してはなりません。gld_register() がこれらの構造体のメンバーの値を使用またはキャッシュすることがあるので、gld_register() を呼び出した後でドライバが変更を行うと、予期せぬ結果を招く可能性があります。

gldm_reset　ドライバのエントリポイントに対するポインタ。gld(9E) を参照

gldm_start　ドライバのエントリポイントに対するポインタ。gld(9E) を参照

gldm_stop　ドライバのエントリポイントに対するポインタ。gld(9E) を参照

gldm_set_mac_addr　ドライバのエントリポイントに対するポインタ。gld(9E) を参照

gldm_set_multicast　ドライバのエントリポイントに対するポインタ。gld(9E) を参照

gldm_set_promiscuous	ドライバのエントリポイントに対するポインタ。gld(9E) を参照
gldm_send	ドライバのエントリポイントに対するポインタ。gld(9E) を参照
gldm_intr	ドライバのエントリポイントに対するポインタ。gld(9E) を参照
gldm_get_stats	ドライバのエントリポイントに対するポインタ。gld(9E) を参照
gldm_ioctl	ドライバのエントリポイントに対するポインタ。NULL にすることができる。gld(9E) を参照
gldm_ident	デバイスに関する短い記述を含む文字列に対するポインタ。システムメッセージ内のデバイスを識別する場合に使用します。
gldm_type	ドライバが処理するデバイスのタイプ。GLD で現在サポートしている値は、DL_ETHER (ISO 8802-3 [IEEE 802.3] および Ethernet Bus)、DL_TPR (IEEE 802.5 Token Passing Ring)、および DL_FDDI (ISO 9314-2 Fibre Distributed Data Interface) です。GLD を正しく動作させるには、この構造体のメンバーを適切に設定する必要があります。
gldm_minpkt	最小の Service Data Unit サイズ - デバイスが送信する最小パケットサイズです (MAC ヘッダーは含まない)。デバイス固有のドライバが必要なすべてのパディングを処理する場合は、このサイズをゼロにすることができます。
gldm_maxpkt	最大の *Service Data Unit* サイズ - デバイスが送信できる最大パケットサイズです (MAC ヘッダーは含まない)。Ethernet の場合、この値は 1500 です。
gldm_addrlen	デバイスが処理する物理アドレスの長さ (バイト数)。Ethernet、Token Ring、および FDDI の場

	合、この構造体のメンバーの値は 6 でなければなりません。
gldm_saplen	ドライバが使用する SAP アドレスの長さ (バイト数)。GLD ベースのドライバでは、これは常に -2 に設定します。この値は、2 バイトの SAP 値がサポートされること、および DLSAP アドレスで物理アドレスの後に SAP が来ることを意味します。詳細は、DLPI 仕様の「Message DL_INFO_ACK」を参照してください。
gldm_broadcast_addr	送信に使用されるブロードキャストアドレスが格納されるバイト配列の長さ gldm_addrlen に対するポインタ。ドライバはブロードキャストアドレスを収めるスペースを提供し、そのスペースに適切な値を入れ、その値を指すように gldm_broadcast_addr を設定する必要があります。Ethernet、Token Ring、および FDDI の場合、ブロードキャストアドレスは通常、0xFF-FF-FF-FF-FF-FF です。
gldm_vendor_addr	ベンダーが提供したデバイスのネットワーク物理アドレスが格納される、バイト配列の長さ gldm_addrlen に対するポインタ。ドライバはこのアドレスを収めるスペースを提供し、そのスペースにデバイスから読み取った情報を入れ、その情報を指すように gldm_vendor_addr を設定する必要があります。
gldm_ppa	デバイスのこのインスタンスに対応する PPA 番号。通常、ddi_get_instance(9F) から返されたインスタンス番号に設定します。
gldm_devinfo	このデバイスの dev_info ノードに対するポインタ
gldm_cookie	ddi_get_iblock_cookie(9F)、ddi_add_intr(9F)、ddi_get_soft_iblock_cookie(9F)、または ddi_add_softintr(9F) から返された割り込

みブロック cookie。これは、gld_recv() の呼び出し元となる、デバイスの受信割り込みに対応しなければなりません。

gld_stats 構造体

GLD 統計 (gld_stats) 構造体は、 gld(9E) および gld(7D) で記述されているように、ドライバの gldm_get_stats() ルーチンから戻るときに、GLD ベースのドライバから GLD に統計情報および状態情報を伝えるために使用します。この構造体のメンバーは、GLD ベースのドライバによってセットされ、GLD が統計情報を報告するときに使用されます。後出の表では、GLD が報告する統計変数の名前がコメントで示されています。個々の統計情報の詳細については、 gld(7D) のマニュアルページを参照してください。

ドライバはこの構造体の長さに関して仮定することができません。Solaris、GLD、またはその両方のリリースごとに異なる可能性があります。このマニュアルに記載されていない GLD 専用の構造体のメンバーは、デバイス固有のドライバによって設定したり読み取ったりしてはなりません。

次の構造体のメンバーは、すべてのメディアタイプに対して定義されます。

```
uint64_t        glds_speed;                     /* ifspeed */
uint32_t        glds_media;                     /* media */
uint32_t        glds_intr;                      /* intr */
uint32_t        glds_norcvbuf;                  /* norcvbuf */
uint32_t        glds_errrcv;                    /* ierrors */
uint32_t        glds_errxmt;                    /* oerrors */
uint32_t        glds_missed;                    /* missed */
uint32_t        glds_underflow;                 /* uflo */
uint32_t        glds_overflow;                  /* oflo */
```

次の構造体のメンバーは、メディアタイプ DL_ETHER に対して定義されます。

```
uint32_t        glds_frame;                     /* align_errors */
uint32_t        glds_crc;                       /* fcs_errors */
uint32_t        glds_duplex;                    /* duplex */
uint32_t        glds_nocarrier;                 /* carrier_errors */
uint32_t        glds_collisions;                /* collisions */
uint32_t        glds_excoll;                    /* ex_collisions */
uint32_t        glds_xmtlatecoll;               /* tx_late_collisions */
uint32_t        glds_defer;                     /* defer_xmts */
uint32_t        glds_dot3_first_coll;           /* first_collisions */
uint32_t        glds_dot3_multi_coll;           /* multi_collisions */
uint32_t        glds_dot3_sqe_error;            /* sqe_errors */
uint32_t        glds_dot3_mac_xmt_error;        /* macxmt_errors */
uint32_t        glds_dot3_mac_rcv_error;        /* macrcv_errors */
uint32_t        glds_dot3_frame_too_long;       /* toolong_errors */
uint32_t        glds_short;                     /* runt_errors */
```

次の構造体のメンバーは、メディアタイプ DL_TPR に対して定義されます。

```
uint32_t        glds_dot5_line_error              /* line_errors */
uint32_t        glds_dot5_burst_error             /* burst_errors */
uint32_t        glds_dot5_signal_loss             /* signal_losses */
uint32_t        glds_dot5_ace_error               /* ace_errors */
uint32_t        glds_dot5_internal_error          /* internal_errors */
uint32_t        glds_dot5_lost_frame_error        /* lost_frame_errors */
uint32_t        glds_dot5_frame_copied_error      /* frame_copied_errors */
uint32_t        glds_dot5_token_error             /* token_errors */
uint32_t        glds_dot5_freq_error              /* freq_errors */
```

次の構造体のメンバーは、メディアタイプ DL_FDDI に対して定義されます。

```
uint32_t        glds_fddi_mac_error;              /* mac_errors */
uint32_t        glds_fddi_mac_lost;               /* mac_lost_errors */
uint32_t        glds_fddi_mac_token;              /* mac_tokens */
uint32_t        glds_fddi_mac_tvx_expired;        /* mac_tvx_expired */
uint32_t        glds_fddi_mac_late;               /* mac_late */
uint32_t        glds_fddi_mac_ring_op;            /* mac_ring_ops */
```

上記の統計変数のほとんどは、特定のイベントが検出された回数を示すカウンタです。例外は次のとおりです。

glds_speed
: インタフェースの現在の帯域幅の概算値 (ビット/秒）。帯域幅の変動のないインタフェース、または正確な推定ができないインタフェースの場合、このオブジェクトには公称の帯域幅が入ります。

glds_media
: ハードウェアで使用されているメディア (配線) またはコネクタのタイプ。現在サポートされているメディア名には、 GLDM_AUI、GLDM_BNC、GLDM_TP、GLDM_10BT、 GLDM_100BT、GLDM_100BTX、 GLDM_100BT4、GLDM_RING4、 GLDM_RING16、 GLDM_FIBER、および GLDM_PHYMII が含まれます。GLDM_UNKNOWN を指定することもできます。

glds_duplex
: インタフェースの現在の二重状態。サポートされる値は、GLD_DUPLEX_HALF および GLD_DUPLEX_FULL です。GLD_DUPLEX_UNKNOWN を指定することもできます。

GLD ルーチンで使用される引数

macinfo	gld_mac_info(9S) 構造体に対するポインタ。
macaddr	有効な MAC アドレスが格納されたキャラクタ配列の先頭に対するポインタ。この配列は、gld_mac_info(9S) 構造体の gldm_addrlen 要素で、ドライバによって指定された長さになります。
multicastaddr	マルチキャスト、グループ、または機能アドレスが格納されたキャラクタ配列の先頭に対するポインタ。この配列は、gld_mac_info(9S) 構造体の gldm_addrlen 要素で、ドライバによって指定された長さになります。
multiflag	マルチキャストアドレスの受け付けを可能にするか不可能にするかを示すフラグ。この引数は、GLD_MULTI_ENABLE または GLD_MULTI_DISABLE として指定します。
promiscflag	有効になっているプロミスキュアスモードのタイプを示すフラグ。この引数は、GLD_MAC_PROMISC_PHYS、GLD_MAC_PROMISC_MULTI、または GLD_MAC_PROMISC_NONE として指定します。
mp	gld_ioctl() は、実行する ioctl が格納されている STREAMS メッセージブロックへのポインタとして *mp* を使用します。gld_send() は、送信するパケットが格納されている STREAMS メッセージブロックへのポインタとして *mp* を使用します。gld_recv() は、受信パケットが格納されているメッセージブロックへのポインタとして *mp* を使用します。
stats	統計カウンタの現在値が入る、gld_stats(9S) 構造体に対するポインタ。
q	ioctl への応答で使用される、queue(9S) 構造体に対するポインタ。

dip デバイスの dev_info 構造体に対するポインタ。

name デバイスのインタフェース名

エントリポイント

これらのエントリポイントは、GLD とのインタフェースとして設計されたデバイス固有のネットワークドライバによって実装される必要があります。

gld(7D) で記述されているように、デバイス固有のドライバと GLD モジュール間の通信に関するメインデータ構造は、gld_mac_info(9S) 構造体です。この構造体の一部の要素は、ここで説明するエントリポイントへの関数ポインタです。デバイス固有のドライバは、attach(9E) ルーチンで、これらの関数ポインタを初期化してから gld_register() を呼び出さなければなりません。

```
int prefix_reset(gld_mac_info_t * macinfo);
```

gldm_reset() は、ハードウェアを初期状態にリセットします。

```
int prefix_start(gld_mac_info_t * macinfo);
```

gldm_start() により、デバイスは割り込みを発生させ、受信データパケットを GLD に配信する目的で、ドライバが gld_recv() を呼び出せるようにします。

```
int prefix_stop(gld_mac_info_t * macinfo);
```

gldm_stop() は、デバイスが割り込みを発生させることを不可にし、データパケットを GLD に配信する目的でドライバが gld_recv() を呼び出すことを止めさせます。GLD は gldm_stop() ルーチンによって、デバイスがこれ以上割り込みをかけなくなることを確実に保証しなければなりません。この関数は常に、GLD_SUCCESS を返します。

```
int prefix_set_mac_addr(gld_mac_info_t * macinfo, unsigned char * macaddr);
```

gldm_set_mac_addr() は、ハードウェアがデータの受信に使用する物理アドレスを設定します。この関数は、デバイスを渡された MAC アドレス *macaddr* にプログラムしなければなりません。現在利用できるリソースが不足していて要求を満たすことができない場合は、GLD_NORESOURCES を返します。要求された関数がサポートされていない場合は、GLD_NOTSUPPORTED を返します。

```
int prefix_set_multicast(gld_mac_info_t * macinfo, unsigned char * multicastaddr,
    int multiflag);
```

gldm_set_multicast() は、特定のマルチキャストアドレスのデバイスレベルでの受け付けを可能または不可能にします。3 番目の引数 *multiflag* が GLD_MULTI_ENABLE に設定されている場合、この関数は 2 番目の引数によって示されたマルチキャストアドレスを持つパケットを受信するよう、インタフェースを設定します。*multiflag* が GLD_MULTI_DISABLE に設定されている場合、ドライバは指定されたマルチキャストアドレスの受け付けを不可にすることが許可されます。

この関数は、GLD がマルチキャスト、グループ、または機能アドレスの受け付けを可能または不可能に設定するたびに呼び出されます。GLD は、デバイスがどのような方法でマルチキャストをサポートするのか、どのような方法でこの関数を呼び出して特定のマルチキャストアドレスを有効または無効にするのかについて、何も想定を行いません。デバイスによっては、ハッシュアルゴリズムとビットマスクを使用して、マルチキャストアドレスの集合を有効にするものもあります。この手順は認められており、GLD が余分なパケットをフィルタリングして除外します。1 つのアドレスを無効にするとデバイスレベルで複数のアドレスが無効になる可能性がある場合、GLD が有効にしているアドレスを無効にしてしまうことがないように、必要な情報を保存するのはデバイスドライバ側の役目です。

gldm_set_multicast() は、すでに有効になっている特定のマルチキャストアドレスを有効にするために呼び出されることはなく、また現在有効になっていないアドレスを無効にするために呼び出されることもありません。GLD は同じマルチキャストアドレスに対する複数の要求を追跡し、特定のマルチキャストアドレスを有効にすることを求める最初の要求、または無効にすることを求める最後の要求に限り、ドライバのエントリポイントを呼び出します。そのとき、利用できるリソースが不足していて要求を満たせない場合は、GLD は GLD_NORESOURCES を返します。要求された関数がサポートされていない場合は、GLD_NOTSUPPORTED を返します。

```
int prefix_set_promiscuous(gld_mac_info_t * macinfo, int promiscflag);
```

gldm_set_promiscuous() は、プロミスキュアスモードを有効または無効にします。この関数は、GLD がメディア上のすべてのパケットの受信、またはメディア上のすべてのマルチキャストパケットの受信を可能または不可能に設定するたびに呼び出されます。2 番目の引数 *promiscflag* が GLD_MAC_PROMISC_PHYS の値に設定されている場合は、この関数は物理レベルのプロミスキュアスモードを有効にし、その結果、メディア上のすべてのパケットが受信されます。*promiscflag* が

GLD_MAC_PROMISC_MULTI に設定されている場合は、すべてのマルチキャストパケットの受信が可能になります。*promiscflag* が GLD_MAC_PROMISC_NONE に設定されている場合は、プロミスキュアスモードが不可になります。

プロミスキュアスマルチキャストモードの要求の場合、マルチキャスト専用プロミスキュアスモードを備えていないデバイスのドライバは、デバイスを物理プロミスキュアスモードに設定して、すべてのマルチキャストパケットが受信されるようにしなければなりません。その場合、ルーチンは GLD_SUCCESS を返さねばなりません。GLD ソフトウェアが余分なパケットをフィルタリングして除外します。そのとき、利用できるリソースが不足していて要求を満たせない場合、GLD_NORESOURCES を返します。要求された関数がサポートされていない場合は、GLD_NOTSUPPORTED を返します。

上位互換性を維持するために、gldm_set_promiscuous() ルーチンは、*promiscflag* として認識できないすべての値を GLD_MAC_PROMISC_PHYS のように扱う必要があります。

```
int prefix_send(gld_mac_info_t * macinfo, mblk_t * mp);
```

gldm_send() は、デバイスへのパケットを送信待ちのキューに入れます。このルーチンには、送信されるべきパケットが入った STREAMS メッセージが渡されます。メッセージには複数のメッセージブロックが含まれることがあり、送信されるパケット全体にアクセスするために、送信ルーチンはそのメッセージ内のすべてのメッセージブロックを通らなければなりません。連結内の長さがゼロのメッセージ継続ブロックを認識してスキップするように、ドライバを準備する必要があります。ドライバでは、パケットが許容される最大パケットサイズを超えていないかどうかをチェックし、必要であれば、許容される最小パケットサイズになるまでパケットにパディングをしなければなりません。送信ルーチンがパケットを正常に送信した場合、またはキューに格納した場合は、GLD_SUCCESS を返します。

送信ルーチンは、送信パケットをすぐに受け付けることができない場合、GLD_NORESOURCES を返します。その場合、GLD は後で再試行を行います。gldm_send() が GLD_NORESOURCES を返した場合、ドライバはリソースが利用できるようになった時点で、gld_sched() を呼び出さなければなりません。この gld_sched() への呼び出しは、ドライバが以前送信用のキューに入れることができなかったパケットを再試行するように、GLD に通知します (ドライバの gldm_stop() ルーチンが呼び出されても、後に gldm_send() ルーチンから再び GLD_NORESOURCES を返すまで、ドライバはこの義務を免除されます。ただし、gld_sched() を余分に呼び出しても、誤った動作になることはありません)。

ドライバの送信ルーチンが GLD_SUCCESS を返し、ドライバとハードウェアでそのメッセージがもう不要になったときに、メッセージを解放するのはドライバの役目です。送信ルーチンがデバイスにメッセージをコピーした場合、または専用バッファにコピーした場合は、コピー後に送信ルーチンでメッセージを解放できます。ハードウェアが DMA を使用して、メッセージデータブロックから直接データを読み取る場合は、ハードウェアによるデータの読み取りが完了するまで、ドライバはメッセージを解放してはなりません。この場合、ドライバは割り込みルーチンでメッセージを解放するか、または将来の送信動作の開始時にバッファ再請求動作でメッセージを解放するのが一般的です。送信ルーチンが GLD_SUCCESS 以外のものを返した場合、ドライバはメッセージを解放してはなりません。ネットワークまたはリンクパートナーとの間に物理接続が無いときに gldm_send() が呼び出された場合は、GLD_NOLINK を返します。

```
int prefix_intr(gld_mac_info_t * macinfo);
```

gldm_intr() は、デバイスに割り込みがかけられている可能性がある場合に呼び出されます。割り込みを他のデバイスと共有する可能性があるので、ドライバはデバイスの状態を調べ、実際に割り込みが発生したかどうかを判別しなければなりません。ドライバが制御しているデバイスで割り込みが起きなかった場合、このルーチンは DDI_INTR_UNCLAIMED を返さなければなりません。それ以外の場合は、割り込みに対処し、DDI_INTR_CLAIMED を返さなければなりません。パケットの正常な受信によって割り込みが発生した場合、このルーチンは受信パケットを M_DATA タイプの STREAMS メッセージに格納し、メッセージを gld_recv() に渡す必要があります。

gld_recv() は、着信パケットをアップストリーム方向に、ネットワークプロトコルスタックの該当する次の層に渡します。gld_recv() を呼び出す前に、STREAMS メッセージの b_rptr および b_wptr メンバーを正しく設定しておくことが重要です。

ドライバは、gld_recv() の呼び出し時に、相互排他 (mutex) ロックまたは他のロックを保持していないようにしなければなりません。特に、送信スレッドが使用する可能性のあるロックは、gld_recv() の呼び出し時に保持されていてはなりません。場合によっては、gld_recv() を呼び出す割り込みスレッドが、発信パケットの送信を含めた処理を実行してしまい、その結果、ドライバの gldm_send() ルーチンが呼び出されてしまうことがあるためです。gldm_intr() ルーチンが gld_recv() を呼び出すときに保持していた相互排他ロックを、gldm_send() ルーチンが取得しようとした場合、相互排他エントリが繰り返されることになり、パニックに陥る可能性があります。gld_recv() を呼び合いドライバが保持する相

互排他ロックを他のドライバのエントリポイントが取得しようとした場合、デッドロックに陥る可能性があります。

割り込みコードは、あらゆるエラーに関する統計カウンタを増分しなければなりません。このエラーには、受信データ用バッファの割り当てエラーをはじめ、CRC エラーやフレーミングエラーなどのハードウェア固有のエラーが含まれます。

```
int prefix_get_stats(gld_mac_info_t * macinfo, struct gld_stats * stats);
```

gldm_get_stats() は、ハードウェア、ドライバ専用カウンタ、またはその両方から統計情報を収集し、*stats* で指し示された gld_stats(9S) 構造体を更新します。このルーチンは、統計要求を受けた時に GLD によって呼び出され、GLD が統計要求に対する応答を作成する前に、ドライバからデバイスに依存する統計情報を得るためのメカニズムを提供します。定義されている統計カウンタの詳細は、gld_stats(9S) および gld(7D) のマニュアルページを参照してください。

```
int prefix_ioctl(gld_mac_info_t * macinfo, queue_t * q, mblk_t * mp);
```

gldm_ioctl() は、すべてのデバイス固有の ioctl コマンドを実装します。ドライバが ioctl 関数を全く実装しない場合、この要素は NULL として指定できます。メッセージブロックを ioctl 応答メッセージに変換し、GLD_SUCCESS を呼び出す前に greply(9F) 関数を呼び出すのは、ドライバの役目です。この関数は常に GLD_SUCCESS を返すべきです。ドライバに報告させる必要のあるエラーは、greply(9F) に渡すメッセージで返させる必要があります。gldm_ioctl 要素が NULL として指定されている場合、GLD は M_IOCNAK タイプのメッセージを EINVAL というエラーとともに返します。

戻り値

これまでに説明した戻り値および制限事項のほかに、一部の GLD エントリポイント関数は次の値を返すことができます。

GLD_BADARG	関数が不適切な引数、たとえば、不良マルチキャストアドレス、不良 MAC アドレス、不良パケット、不良パケット長などを検出した場合
GLD_FAILURE	ハードウェア障害の場合
GLD_SUCCESS	成功した場合

サービスルーチン

```
gld_mac_info_t * gld_mac_alloc(dev_info_t * dip);
```

gld_mac_alloc() は、新しい gld_mac_info(9S) 構造体を割り当て、その構造体に対するポインタを返します。この構造体の GLD 専用の要素のなかには、gld_mac_alloc() を返す前に初期化されるものがありますが、他の要素はすべてゼロに初期化されます。デバイスドライバは、mac_info 構造体へのポインタを gld_register() へ渡す前に、gld_mac_info(9S) に記述されているように、一部の構造体メンバーを初期化する必要があります。

```
void gld_mac_free(gld_mac_info_t * macinfo);
```

gld_mac_free() は、以前に gld_mac_alloc() によって割り当てられていた gld_mac_info(9S) 構造体を解放します。

```
int gld_register(dev_info_t * dip, char * name, gld_mac_info_t * macinfo);
```

gld_register() は、デバイスドライバの attach(9E) ルーチンから呼び出され、GLD ベースのデバイスドライバと GLD フレームワークを結びつけるために使用されます。デバイスドライバの attach(9E) ルーチンは、gld_register() を呼び出す前に、gld_mac_alloc() を最初に使用して gld_mac_info(9S) 構造体を割り当て、その構造体の要素のいくつかを初期化しなければなりません。詳細は、gld_mac_info(9S) のマニュアルページを参照してください。gld_register() が正常に呼び出されると、次の動作が発生します。

- デバイス固有のドライバを GLD システムに接続する
- デバイス固有のドライバの専用データポインタが (ddi_set_driver_private(9F) を使用して) macinfo 構造体を指し示すように設定する
- マイナーデバイスノードを作成する
- DDI_SUCCESS を返す

gld_register() に渡すデバイスインタフェース名は、ファイルシステムに存在しているドライバモジュール名と完全に一致しなければなりません。

ドライバの attach(9E) ルーチンは、gld_register() が正常に完了した場合に DDI_SUCCESS を返す必要があります。gld_register() が DDI_SUCCESS を返さなかった場合、attach(9E) ルーチンは gld_register() を呼び出す前に割り当

てたすべてのリソースの割り当てを解除し、その後 DDI_FAILURE を返さなければなりません。

```
int gld_unregister(gld_mac_info_t * macinfo);
```

gld_unregister() は、デバイスドライバの detach(9E) 関数によって呼び出され、成功した場合は次の作業を実行します。

- デバイスの割り込みが中止されたことを確認し、必要に応じてドライバの gldm_stop() ルーチンを呼び出す
- マイナーデバイスノードを削除する
- GLD システムとデバイス固有のドライバ間のリンクを切断する
- DDI_SUCCESS を返す

gld_unregister() が DDI_SUCCESS を返した場合、detach(9E) ルーチンは attach(9E) ルーチンで割り当てられたすべてのデータ構造体を割り当て解除し、gld_mac_free() を使用して、macinfo 構造を割り当て解除し、DDI_SUCCESS を返します。gld_unregister() が DDI_SUCCESS を返さなかった場合、ドライバの detach(9E) ルーチンはデバイスを動作状態にしたまま、DDI_FAILURE を返さなければなりません。

```
void gld_recv(gld_mac_info_t * macinfo, mblk_t * mp);
```

gld_recv() は、ドライバの割り込みハンドラによって呼び出され、受信したパケットをアップストリームに渡します。ドライバは raw パケットを格納した STREAMS M_DATA メッセージを作成して渡さなければなりません。gld_recv() がパケットのコピーを受け取るべき STREAMS キュー (あれば) を判別し、必要に応じてコピーします。さらに、必要であれば DL_UNITDATA_IND メッセージをフォーマットして、データを該当するすべての Stream に渡します。

ドライバは、gld_recv() の呼び出し時に、相互排他 (mutex) ロックまたは他のロックを保持していないようにしなければなりません。特に、送信スレッドが使用するロックは、gld_recv() の呼び出し時には保持できません。場合によっては、gld_recv() を呼び出す割り込みスレッドが、発信パケットの送信を含めた処理を実行してしまい、その結果、ドライバの gldm_send() ルーチンが呼び出されることがあるためです。gldm_intr() ルーチンが gld_recv() を呼び出すときに保持していた相互排他ロックを、gldm_send() ルーチンが取得しようとした場合、相互排他エントリが繰り返されることになり、パニックになる可能性があります。gld_recv() を呼び合いドライバが保持する相互排他ロックを他のドライバ

のエントリポイントが取得しようとした場合、デッドロックに陥る可能性があります。

```
void gld_sched(gld_mac_info_t * macinfo);
```

gld_sched() は、据え置かれていた発信パケットを再スケジューリングするために、デバイスドライバによって呼び出されます。ドライバの gldm_send() ルーチンが GLD_NORESOURCES を返すたびに、ドライバは後で gld_sched() を呼び出して、以前送信できなかったパケットについて再試行するように、GLD フレームワークに通知しなければなりません。gld_sched() は、リソースが利用可能になった時点で、できるかぎり迅速に呼び出され、GLD がドライバの gldm_send() ルーチンに対する発信パケットの受け渡しを、タイミングよく再開できるようにしなければなりません (ドライバの gldm_stop() ルーチンが呼び出されても、gldm_send() ルーチンから再び GLD_NORESOURCES が返されるまで、ドライバはこの義務を免除されます。ただし、gld_sched() を余分に呼び出しても、誤った動作になることはありません)。

```
uint_t gld_intr(caddr_t);
```

gld_intr() は、GLD のメインの割り込みハンドラです。通常、デバイスドライバの ddi_add_intr(9F) 呼び出しで、割り込みルーチンとして指定します。割り込みハンドラに対する引数 (ddi_add_intr(9F) 呼び出しに *int_handler_arg* として指定) は、gld_mac_info(9S) 構造体へのポインタでなければなりません。gld_intr() は、該当する場合、デバイスドライバの gldm_intr() 関数を呼び出し、そのポインタを gld_mac_info(9S) 構造体に渡します。しかし、ドライバが上位レベルの割り込みを使用する場合は、独自の上位割り込みハンドラを提供し、その中からソフト割り込みを起動しなければなりません。この場合、gld_intr() は通常、ddi_add_softintr() 呼び出しにソフト割り込みハンドラとして指定されます。gld_intr() は、割り込みハンドラに適した値を返します。

第 6 章

言語サポートについてのトピック

このトピックでは、Solaris 環境における言語サポートについて記述します。次の章で構成されています。

75ページの「ヨーロッパ言語版 Solaris ソフトウェア用に追加された部分ロケール」	新しい部分ロケールを記載します。
第 8 章	mp プリントフィルタの動作をカスタマイズする方法、およびトラブルシューティング (障害追跡) の情報を記載します。

第 7 章

追加の部分ロケール

Solaris 8 10/00 リリースで追加の部分ロケールが新規に追加され、Solaris 8 4/01 リリースで東ヨーロッパのロケールが更新されました。Solaris ソフトウェアの言語サポートについての詳細は、『国際化対応言語環境の利用ガイド』を参照してください。

注 - 最新のマニュアルページを参照するには、`man` コマンドを使用してください。Solaris 8 Update リリースのマニュアルページには、「*Solaris 8 Reference Manual Collection*」には記載されていない新しい情報も提供されています。

ヨーロッパ言語版 Solaris ソフトウェア用に追加された部分ロケール

Solaris 8 10/00 リリースで、ロシア語およびポーランド語対応の UTF-8 ロケール、さらにカタロニア語対応の新しい 2 つのロケールが追加されました。ロケール名は、次のとおりです。

- ru_RU.UTF-8
- pl_PL.UTF-8
- ca_ES.ISO8859-1
- ca_ES.ISO8859-15

Solaris 8 4/01 リリースで、トルコ語および他のロケール用の UTF-8 ロケールが追加されました。

追加されたロケールは、言語サポート (メッセージの翻訳および GUI) がないので、部分的ロケールです。

Solaris 製品のローカライゼーション

中央ヨーロッパ

表 7–1 中央ヨーロッパ

ロケール	ユーザーインタフェース	地域	コードセット	言語サポート
cs_CZ.ISO8859-2	英語	チェコ	ISO8859-2	チェコ語 (チェコ)
de_AT.ISO8859-1	ドイツ語	オーストリア	ISO8859-1	ドイツ語 (オーストリア)
de_AT.ISO8859-15	ドイツ語	オーストリア	ISO8859-15	ドイツ語 (オーストリア、ISO8859-15 - Euro)
de_CH.ISO8859-1	ドイツ語	スイス	ISO8859-1	ドイツ語 (スイス)
de_DE.UTF-8	ドイツ語	ドイツ	UTF-8	ドイツ語 (ドイツ、Unicode 3.0)
de_DE.ISO8859-1	ドイツ語	ドイツ	ISO8859-1	ドイツ語 (ドイツ)
de_DE.ISO8859-15	ドイツ語	ドイツ	ISO8859-15	ドイツ語 (ドイツ、ISO8859-15 - Euro)
fr_CH.ISO8859-1	フランス語	スイス	ISO8859-1	フランス語 (スイス)
hu_HU.ISO8859-2	英語	ハンガリー	ISO8859-2	ハンガリー語 (ハンガリー)
pl_PL.ISO8859-2	英語	ポーランド	ISO8859-2	ポーランド語 (ポーランド)
pl_PL.UTF-8	英語	ポーランド	UTF-8	ポーランド語 (ポーランド、Unicode 3.0)
sk_SK.ISO8859-2	英語	スロバキア	ISO8859-2	スロバキア語 (スロバキア)

東ヨーロッパ

表 7–2 東ヨーロッパ

ロケール	ユーザーインタフェース	地域	コードセット	言語サポート
bg_BG.ISO8859-5	英語	ブルガリア	ISO8859-5	ブルガリア語 (ブルガリア)
et_EE.ISO8859-15	英語	エストニア	ISO8859-15	エストニア語 (エストニア)
hr_HR.ISO8859-2	英語	クロアチア	ISO8859-2	クロアチア語 (クロアチア)
lt_LT.ISO8859-13	英語	リトアニア	ISO8859-13	リトアニア語 (リトアニア)
lv_LV.ISO8859-13	英語	ラトビア	ISO8859-13	ラトビア語 (ラトビア)
mk_MK.ISO8859-5	英語	マケドニア	ISO8859-5	マケドニア語 (マケドニア)
ro_RO.ISO8859-2	英語	ルーマニア	ISO8859-2	ルーマニア語 (ルーマニア)
ru_RU.KOI8-R	英語	ロシア	KOI8-R	ロシア語 (ロシア、KOI8-R)
ru_RU.ANSI1251	英語	ロシア	ansi-1251	ロシア語 (ロシア、ANSI 1251)
ru_RU.ISO8859-5	英語	ロシア	ISO8859-5	ロシア語 (ロシア)
ru_RU.UTF-8 (Unicode 3.0)	英語	ロシア	UTF-8	ロシア語 (ロシア、Unicode 3.0)
sh_BA.ISO8859-2@bosnia	英語	ボスニア	ISO8859-2	ボスニア語 (ボスニア)
sl_SI.ISO8859-2	英語	スロベニア	ISO8859-2	スロベニア語 (スロベニア)
sq_AL.ISO8859-2	英語	アルバニア	ISO8859-2	アルバニア語 (アルバニア)
sr_YU.ISO8859-5	英語	セルビア	ISO8859-5	セルビア語 (セルビア)

表 7–2　東ヨーロッパ　続く

ロケール	ユーザーインタフェース	地域	コードセット	言語サポート
tr_TR.ISO8859-9	英語	トルコ	ISO8859-9	トルコ語 (トルコ)
tr_TR.UTF-8	英語	トルコ	UTF-8	トルコ語 (トルコ、Unicode 3.0)

南ヨーロッパ

表 7–3　南ヨーロッパ

ロケール	ユーザーインタフェース	地域	コードセット	言語サポート
ca_ES.ISO8859-1	英語	スペイン	ISO8859-1	カタロニア語 (スペイン)
ca_ES.ISO8859-15	英語	スペイン	ISO8859-15	カタロニア語 (スペイン、ISO8859-15 - Euro)
el_GR.ISO8859-7	英語	ギリシャ	ISO8859-7	ギリシャ語 (ギリシャ)
es_ES.ISO8859-1	スペイン語	スペイン	ISO8859-1	スペイン語 (スペイン)
es_ES.ISO8859-15	スペイン語	スペイン	ISO8859-15	スペイン語 (スペイン、ISO8859-15 - Euro)
es_ES.UTF-8	スペイン語	スペイン	UTF-8	スペイン語 (スペイン、Unicode 3.0)
it_IT.ISO8859-1	イタリア語	イタリア	ISO8859-1	イタリア語 (イタリア)
it_IT.ISO8859-15	イタリア語	イタリア	ISO8859-15	イタリア語 (イタリア、ISO8859-15 - Euro)
it_IT.UTF-8	イタリア語	イタリア	UTF-8	イタリア語 (イタリア、Unicode 3.0)

表 7–3 南ヨーロッパ 続く

ロケール	ユーザーインタフェース	地域	コードセット	言語サポート
pt_PT.ISO8859-1	英語	ポルトガル	ISO8859-1	ポルトガル語 (ポルトガル)
pt_PT.ISO8859-15	英語	ポルトガル	ISO8859-15	ポルトガル語 (ポルトガル、ISO8859-15 - Euro)

ヨーロッパのローカライゼーション

Solaris 8 ソフトウェアは、ユーロ通貨をサポートします。下位互換性を維持するために、各国のこれまでの通貨記号も引き続き使用できます。

表 7–4 ユーロ通貨をサポートするユーザーロケール

地域	ロケール名	**ISO** コードセット
オーストリア	de_AT.ISO8859-15	8859-15
ベルギー (フランス語)	fr_BE.ISO8859-15	8859-15
ベルギー (オランダ語)	nl_BE.ISO8859-15	8859-15
デンマーク	da_DK.ISO8859-15	8859-15
フィンランド	fi_FI.ISO8859-15	8859-15
フランス	fr_FR.ISO8859-15	8859-15
ドイツ	de_DE.ISO8859-15	8859-15
アイルランド	en_IE.ISO8859-15	8859-15
イタリア	it_IT.ISO8859-15	8859-15
オランダ	nl_NL.ISO8859-15	8859-15
ポルトガル	pt_PT.ISO8859-15	8859-15

表 7–4 ユーロ通貨をサポートするユーザーロケール 続く

地域	ロケール名	ISO コードセット
スペイン	ca_ES.ISO8859-15	8859-15
スペイン	es_ES.ISO8859-15	8859-15
スウェーデン	sv_SE.ISO8859-15	8859-15
英国	en_GB.ISO8859-15	8859-15
米国	en_US.ISO8859-15	8859-15

第 8 章

プリントフィルタ mp(1) の拡張

Solaris 8 4/01 リリースで、mp プリントフィルタが新規に追加されました。Solaris ソフトウェアの言語サポートについての情報の詳細は、『国際化対応言語環境の利用ガイド』を参照してください。

注 - 最新のマニュアルページを参照するには、man コマンドを使用してください。Solaris 8 Update リリースのマニュアルページには、「*Solaris 8 Reference Manual Collection*」には記載されていない新しい情報も提供されています。

---

mp(1) プリントフィルタ拡張機能の概要

mp(1) プリントフィルタの機能が拡張され、2.5.11 バージョンでは X Print Server のクライアントとして動作できるようになりました。この節では、mp(1) の動作をカスタマイズする方法について説明します。

カスタマイズには、次の 2 種類があります。

- mp(1) が正しい出力を印刷できるように変更する
- mp(1) の出力の形式を変更できるように変更する

ここでは、mp(1) が正しく出力しない場合の対応方法について簡単に説明します。mp バージョン 2.5.11 は機能拡張され、X Print Server クライアントとして動作できるようになりました。

正しく構成された X Print Server が動作している場合は、mp(1) は次の構成ファイルをまったく変更せずに、対象ロケールに正しく印刷します。X Print Server が動作していない場合は、次の節の説明に
従い、/usr/lib/lp/locale/$LANG/mp/mp.conf をローカライズ (地域化) します。mp.conf ファイルは、/usr/openwin/lib/locale/$LANG/print/ の下のいくつかのロケールにある prolog.ps ファイルと、実質的には同じ機能を果たします。prolog.ps ファイルをカスタマイズすることもできます。

/usr/lib/lp/locale/$LANG/mp/ ディレクトリにも prolog.ps ファイルが入っている場合があります。このファイルは下位互換性を維持するためにのみ提供されています。–P オプションか –D オプションを指定するときに、直接の X Print Server クライアントモードを使用するか、または mp.conf ファイルを構成することをお勧めします。出力の形式を変更するには、提供されているプロローグ (prolog) ファイルをカスタマイズします。

mp(1) が構成ファイルを選択するときの指針は、次のとおりです。

- –D <*target printer name*> または -P <*target printer name*> を指定した場合は、.xprプロローグファイルが読まれます。それ以外のファイルは読まれません。
- コマンド行で –D や –P が指定されておらず、/usr/openwin/lib/locale/$LANG/print/prolog.ps が存在する場合は、出力の先頭にそのファイルが挿入されます。印刷スタイルによっては、.ps プロローグページレイアウトファイルも出力に挿入されます。MP_LANG が環境で定義されている場合は、LANG の代わりに使用します。LANG はロケール環境変数です。
- prolog.ps ファイルが /usr/openwin/lib/locale/$LANG/print/ にない場合は、usr/lib/lp/locale/$LANG/mp/ で検索されます。$MP_LANG が定義されている場合は、$LANG の代わりに使用します。prolog.ps ファイルがそのディレクトリになく、mp.conf ファイルが存在している場合、mp.conf ファイルが走査されます。印刷スタイルによっては、ps プロローグページレイアウトファイルも出力に挿入されます。prolog.ps も mp.conf ファイルもない場合は、mp によって C ロケールの場合と同じように ASCII 文字だけが印刷されます。

次に、PostScript (.ps) または X Print クライアントページ (.xpr) のフォーマット (書式) ファイルを選択する方法の指針について説明します。

- –D <*target printer name*> か –P <*target printer name*> を指定した場合は、/usr/lib/lp/locale/C/mp の .xpr プロローグファイルを使用します。自分のディレクトリを出力プロローグファイル用に設定するときは、MP_PROLOGUE 変数がそのディレクトリを指すように設定します。
- 通常の出力モードで印刷する時は、/usr/lib/lp/locale/C/mp ディレクトリの ps ファイルを使用します。ここでは、MP_PROLOGUE 変数も使用できます。

構成ファイルのローカライゼーション

構成ファイルを使用するのは、必要に応じて、フォントエントリやフォントグループエントリの変更や追加が自在に行えるからです。

使用されているシステムのデフォルトの構成ファイルは /usr/lib/lp/locale/$LANG/mp/mp.conf です。ただし、$LANG は印刷が行われるロケールのロケール環境変数です。ユーザーは専用の構成ファイルを持つことができ、–u <*config.file path*> オプションで指定できます。

mp.conf ファイルは主に、ロケールの中間コードポイント (符号位置) を、印刷に用いるフォントのエンコーディング (符号化方式) のプレゼンテーションフォーム (表現形式) に対応付けさせるために用いられます。

注 - 互換性のための文字としてエンコーディングされた合字や可変のグリフを、プレゼンテーションフォーム (表現形式) といいます。

中間コードポイントはワイドキャラクタか Portable Layout Service (PLS) 層の出力のどちらかです。Complex Text Layout の印刷では、中間コードポイントは PLS 出力でなければなりません。mp(1) によって生成されるデフォルトの中間コードは、PLS 出力です。

現在サポートされるフォントフォーマットは、Portable Compiled Format (PCF)、TrueType、および Type1 のフォーマットです。システム常駐とプリンタ常駐の両方の Type1 フォントがサポートされています。

mp.conf 構成ファイルのローカライゼーションの目的は、特定のロケールの必要性に応じて印刷できるように mp を構成することです。次に、mp(1) 用の mp.conf 構成ファイルのフォーマットと内容について説明します。

- 行は有効なキーワード(指令)で始まらなくてはなりません。
- キーワードへの引数は、キーワードと同じ行になければなりません。

- 「#」文字で始まる行は、行の終わりまでコメントとみなされます。
- 0x で始まる数字の引数は、16 進数と解釈されます。

mp.conf ファイルに関し、次の情報をそれぞれの節で説明します。

- フォントの別名付け
- フォントグループの定義
- ロケール内の、中間コード範囲からフォントグループへの対応付け
- それぞれのフォントと、中間コードポイントをフォントのエンコーディングのプレゼンテーションフォームに対応付けする共有オブジェクトの関連付け

フォントの別名化

この節では、印刷で使用するフォントのために別名を定義します。この節の各行は、次の形式になっています。

```
keyword   font alias name    font type      font path
```

<keyword> この節のキーワードは FontNameAlias です。

<font alias name> フォント名の別名に関する一般的な慣例では、フォントのエンコーディング/スクリプト名 (文字の種類) を指定し、続けて、フォントがローマ字、ボールド、イタリック体、あるいはボールドイタリック体なのかを示す文字 (R、B、I、または BI) を指定します。たとえば /usr/openwin/lib/X11/fonts/75dpi/courR18.pcf.Z なら、iso88591 のローマ字フォントなので、別名 iso88591R を指定することができます。

<font type> .pcf フォントには PCF を、Adobe Type1 フォントには Type1 を、TrueType フォントには TrueType を指定します。この設定ファイルでは、この 3 種類のフォントだけが構成できます。

<font path> 設定フォントファイルには、絶対パス名を指定します。type1 プリンタ常駐フォントには、フォント名だけを指定します。次に、例を示します。

```
FontNameAlias    prnHelveticaR     Type1       Helvetica
```

フォントグループの定義

同じ種類のフォントを組み合わせて、フォントグループを作成できます。フォントグループのフォーマットは次のとおりです。

- この節の <keyword> は FontGroupです。
- <fontgroupname> はフォントのグループ名です。
- <GroupType> はフォントの種類です。同じ種類のフォント (PCF、Type1、TrueType) 同士でのみ、フォントグループを作成してください。
- <Roman> はフォントグループのローマ字フォント名です。
- <Bold> はフォントグループのボールドのフォント名です。
- <Italic> はフォントグループのイタリック体のフォント名です。
- <BoldItalic> はフォントグループのボールドイタリック体のフォント名です。

注 - グループの作成では、ローマ字フォントのエントリだけが必須です。ボールド、イタリック体、およびボールドイタリック体のフォントは省略できます。さまざまなフォントを使用して、メールやニュースの記事の見出し行が表示できます。ローマ字のフォントだけが定義されている場合は、他のフォントの代わりに使用します。

ロケール内の、中間コード範囲からフォントグループへの対応付け

- この節の <keyword> は MapCode2 です。この節のフォントは MapCode2Font です。
- <range_start> は、0x で始まる 4 バイトの 16 進の値で、1 つ以上のフォントグループに対応付けするコード範囲の開始点を示します。
- <range_end> は対応付けするコード範囲の終わりを示します。「-」で示すこともでき、その場合は単一の中間コードポイントだけが対象フォントに対応付けされます。
- <group>は、Type1、PCF、あるいは TrueType フォントグループで、これによりプレゼンテーションフォームが印刷されます。

それぞれのフォントと、中間コードポイントをフォントのエンコーディングのプレゼンテーションフォームに対応付けする共有オブジェクトとの関連付け

- <keyword> は CnvCode2Fontです。

- <font alias name> は、フォント用に定義される別名です。
- <mapping function> は中間コードで入力し、フォントのエンコーディングでプレゼンテーションフォームを出力し、次にそれを使用してグリフインデックスが取得され、グリフが描画されます。
- <file path having mapping function> は、対応付け関数がある .so ファイル名です。dumpcs ユーティリティを使用して、EUC ロケールの中間コードセットが検索できます。

注 - mp(1) で現在使用されている TrueType 処理系は、フォーマット 4 と PlatformID 3 の cmap しか処理できません。つまり、Microsoft の .ttf ファイルしか構成できません。また、TrueType フォント処理系が正しく動作するためには、文字対応付けエンコーディングは Unicode か Symbol でなければなりません。Solaris 環境の .ttf フォントの大部分はこれらの制限事項に従っているため、mp.conf ファイルで Solaris ソフトウェアの TrueType フォントがすべて対応付けできます。

X Logical Fonts Description (XLFD) に対応するフォントを対応付けする共有オブジェクトを作成するときは、次のことを考慮します。PCF/Type1 フォントを対応付けしている場合は、中間コード範囲から XLFD によって指定されたエンコーディングへと対応付けする共有オブジェクトを作成します。次に例を示します。

```
-monotype-arial-bold-r-normal-bitmap-10-100-75-75-p-54-iso8859-8
```

対応する PCF フォントは次のようになります。

```
/usr/openwin/lib/locale/iso_8859_8/X11/fonts/75dpi/ariabd10.pcf.Z
```

このフォントは iso8859-8 で符号化されているため、共有オブジェクトは中間コードと対応する iso8859-8 のコードポイントとの間で対応付けしなければなりません。

しかし、次の XLFD で示される TrueType フォントに

```
-monotype-arial-medium-r-normal--0-0-0-0-p-0-iso8859-8
```

次のフォントが対応する場合、

```
/usr/openwin/lib/locale/iso_8859_8/X11/fonts/TrueType/arial__h.ttf
```

この TrueType フォントの cmap エンコーディングは Unicode なので中間コードと Unicodeの間で対応付けします。この TrueType フォントの例で、たとえば en_US.UTF-8 ロケールで中間コードがヘブライ文字 (PLS 層で生成) の 0xe50000e9 である場合は、次の点を考慮する必要があります。フォントが Unicode で符号化されているため、0xe50000e9 を入力したときに、出力が Unicode のプレゼンテーショ

ンフォームに対応するように、対応する .so モジュール内の関数を設計する必要があります。ここでの例は 0x000005d9 です。

<mapping function> 関数のプロトタイプは、次のとおりでなければなりません。

```
unsigned int function(unsigned int inter_code_pt)
```

次に示されているのは、mp.conf で使用できる任意のキーワード/値のペアです。

```
PresentationForm        WC/PLSOutput
```

デフォルトの値は、PLSOutput です。ユーザーが「WC」を指定している場合、生成される中間コードポイントはワイドキャラクタです。CTL の印刷では、デフォルト値を使用する必要があります。

ロケールが CTL 以外のロケールでキーワードの値が PLSOutput の場合は無視され、mp(1) は代わりにワイドキャラクタコードを生成します。

ロケールで CTL がサポートされている場合は、次に示す任意のキーワード/値のペアが使用できます。これらの変数は、表の右側に記載されているどの値でもとることができます。

Orientation	ORIENTATION_LTR/ ORIENTATION_RTL/ ORIENTATION_CONTEXTUAL	デフォルトは ORIENTATION_LTR
Numerals	NUMERALS_NOMINAL/ NUMERALS_NATIONAL/ NUMERALS_CONTEXTUAL	デフォルトは NUMERALS_NOMINAL
TextShaping	TEXT_SHAPED/ TEXT_NOMINAL/ TEXT_SHFORM1/ TEXT_SHFORM2/ TEXT_SHFORM3/ TEXT_SHFORM4	デフォルトは TEXT_SHAPED

次の例では、構成ファイルに新たに PCF、TrueType、あるいは Type1 プリンタ常駐フォントを追加するときの手順について説明します。

0x00000021 ～ 0x0000007f の範囲の文字表示用フォントを、現在構成されている PCF フォントではなく TrueType フォントに置き換えます。

新たにフォントを追加する前に、次のように、現在構成されているフォントに対応する構成ファイルのさまざまなコンポーネントを調べます。

```
FontNameAlias   iso88591R       PCF     /usr/openwin/lib/X11/fonts/75dpi/courR18PCF.Z
FontNameAlias   iso88591B       PCF     /usr/openwin/lib/X11/fonts/75dpi/courB18PCF.Z
.
```

```
.
.
FontGroup       iso88591          PCF       iso88591R iso88591B
.
.
.
MapCode2Font    0x00000020      0x0000007f      iso88591
.
.
.
CnvCode2Font    iso88591R _xuiso88591 /usr/lib/lp/locale/$LANG/mp/xuiso88591.so
CnvCode2Font    iso88591B _xuiso88591 /usr/lib/lp/locale/$LANG/mp/xuiso88591.so
```

en_US.UTF-8 ロケールで対応付けを行う候補として、/usr/openwin/lib/locale/ja/X11/fonts/TT/HG-MinchoL.ttf を選んだとします。これは Unicode 文字で対応付けされた TrueType フォントファイルであるため、.so モジュール内の対応付け関数では、入ってくる UCS-2 コードポイントを直接返す関数だけが必要となります。

```
unsigned short _ttfjis0201(unsigned short ucs2) {
                return(ucs2);
        }
```

これを ttfjis0201.c ファイルに保存します。次のように共有オブジェクトを作成します。

```
cc -G -Kpic -o ttfjis0201.so ttfjis0201.c
```

しかし、/usr/openwin/lib/locale/ja/X11/fonts/75dpi/gotmrk20.pcf.Z などの PCF ファイルを対応付けしている場合は、/usr/openwin/lib/locale/ja/X11/fonts/75dpi/ ディレクトリにある fonts.dir ファイルを参照します。次のような XLFD に対応したエンコーディングに慣れる必要があります。

```
-sun-gothic-medium-r-normal--22-200-75-75-c-100-jisx0201.1976-0
```

JIS X 0201 がエンコーディングである場合、UCS-2 から JIS X 0201 へと対応付けする共有オブジェクトを作成します。.so モジュールを作成するためには、対応付けテーブルを取得する必要があります (手元にない場合)。Unicode ロケールでは、ftp.unicode.org/pub/MAPPINGS/ の下にある、多くのキャラクタセット (文字集合) から Unicode への対応付けが有用です。xu2jis0201.c ファイルを書くためには、これらの対応付けに従う必要があります。

```
 unsigned short _xu2jis0201(unsigned short ucs2) {
                        if(ucs2 >= 0x20 && ucs2 <= 0x7d )
                                return (ucs2);
                        if(ucs2==0x203e)
                                return (0x7e);
                        if(ucs2 >= 0xff61 && ucs2 <= 0xff9f)
```

```
                        return (ucs2 - 0xff60 + 0xa0);
                return(0);
        }
```

対応付けファイルを作成するときに、UCS-2 から jisx0201 への対応のすべての場合を組み込みます。

```
cc  -G -o xu2jis0201.so xu2jis0201.c
```

この例では、共有オブジェクトファイルを作成します。

mp.conf の対応する節に次の行を追加し、このフォントを追加します。この例では、TrueType フォントを追加する方法を示します。PCF フォントも同様ですが、キーワードは TrueType ではなく PCF に変更します。

```
FontNameAlias   jis0201R TrueType
/home/fn/HG-Minchol.ttf
FontGroup     jis0201        TrueType    jis0201R
MapCode2Font    0x0020  0x007f   jis0201
CnvCode2Font    jis0201R    _ttfjis0201 <.so path>
```

この行は追加する前に mp.conf から削除する必要があります。

```
                    MapCode2Font    0x0020  0x007f  jis0201 CnvCode2Font
                    jis0201R  _ttfjis0201 <.so path>
```

.so パスは xu2jis0201.so ファイルを指しています。

変更した mp.conf ファイルを使用して mp(1) を実行すると、0x0020 ～ 0x007f の範囲が新しいフォントで印刷されます。同じ.so ファイルを使用して、0x0000FF61 0x0000FF9F の範囲など、他の日本語の文字範囲も対応付けします。

下位互換性を維持するため、/usr/openwin/lib/locale/$LANG/print/prolog.ps ファイルが存在すれば、それを使用して現在のロケールで出力を作成します。ただし、$LANG はロケールコンポーネントの 1 つになります。この場合、構成ファイルの機構は使用されません。

mp.conf ファイルのサンプルである、/usr/lib/lp/locale/en_US.UTF-8/mp/mp.conf を参照してください。

既存のプロローグファイルのカスタマイズと新規プロローグファイルの追加

プロローグファイルは次の 2 つのカテゴリに分類されます。

- PostScript プロローグファイル (.ps)
- X print server client プロローグファイル (.xpr).

まず、PostScript ファイルのカスタマイズについて説明します。

- 共通プロローグファイル
- 印刷レイアウトプロローグファイル
- ロケール依存プロローグファイル

次に、プロローグファイルの各カテゴリのカスタマイズについて説明します。

共通プロローグファイル

共通プロローグファイルには、次のものがあります。

- `mp.pro.ps`
- `mp.common.ps`
- `mp.pro.alt.ps`
- `mp.pro.fp.ps`
- `mp.pro.ps`
- `mp.pro.ts.ps`
- `mp.pro.altl.ps`
- `mp.pro.ff.ps`
- `mp.pro.l.ps`
- `mp.pro.tm.ps`

上記のファイルは印刷レイアウトプロローグファイルで、`mp.common.ps` は他のプロローグファイルの前に挿入される共通プロローグファイルです。

共通プロローグファイルである `mp.common.ps` は、`/usr/lib/lp/locale/C/mp/` ディレクトリにあり、それには、StandardEncoding から ISOLatin1Encoding にフォントを再度符号化する PostScript ルーチンが含まれています。reencodeISO ルーチンが印刷レイアウトプロローグファイルから呼び出され、フォントのエンコーディングが変更されます。通常、このプロローグファイルには、カスタマイズは必要ありません。ユーザーが自分のプ

ロローグファイルを作成している場合は、修正されたプロローグファイルがあるディレクトリを指すように環境変数 MP_PROLOGUE を設定します。

印刷レイアウトプロローグファイル

印刷レイアウトプロローグファイルである mp.*.ps ファイルには、印刷時にページレイアウトを制御するルーチンが入っています。これらのプロローグファイルは印刷ページのヘッダーとフッターで使用するユーザー名、印刷日、ページ番号に加えて、他の情報も設定できます。たとえば、有効な印刷領域のサイズや、使用する用紙の縦横の向きが指定できます。

一連の標準的な関数については、プロローグファイルごとに定義する必要があります。これらの関数は、新しい印刷ページの開始、印刷ページの終了、あるいは新しいカラムの終了時に呼び出されます。これらの関数の実装内容によって、印刷出力の印刷属性が定義されます。

次の PostScript 変数は mp(1) バイナリによって実行時に定義されます。すべての印刷レイアウトファイルはこれらの変数を使用して、user name、subject、print time などの動的な情報を印刷できます。変数から取得したこの情報は、通常印刷ページのヘッダーやフッターに記載されます。

User システムの passwd ファイルから取得される、mp を実行しているユーザーの名前

MailFor 印刷する記事の種類の名前を保持するため使用される変数。この変数の値には、次のものがあります。

- 「Listing for」—入力がテキストファイルである時
- 「Mail for」—入力がメールファイルである時
- 「Article from」—入力がニュースグループの記事である時

Subject メールやニュースのヘッダーから引用された主題。「-s」オプションを使用すると、メールやニュースのファイルや標準テキストファイルにも主題が強制できます。

Timenow ヘッダーやフッターに記載される印刷時刻。この情報は localtime() 関数から取得されます。

次は、印刷レイアウトプロローグファイルに実装されている関数です。これらの関数にはすべて副関数が使用できます。

endpage 使用法 : page_number endpage

印刷ページの下端に達した時に呼び出されます。この関数はページのグラフィックのコンテキストを復元し、showpage を発行します。プロローグファイルによっては、ヘッダーとフッターの情報はカラムごとではなくページごとでしか表示されません。ヘッダーとフッターでグレースケールの模様を表示する副関数を呼び出すように、この関数を実装することも可能です。

newpage 使用法 : page_number newpage

新しいページの開始時に実行されるルーチンやコマンド。ルーチンの機能には、ランドスケープモードの設定、印刷グラフィックコンテキストの保存、ページ座標の変換などがあります。

endcol 使用法 : page_number col_number endcol

ヘッダーやフッター情報の表示。新しい印刷位置への移動など。

新規に印刷レイアウトプロローグファイルを追加するときは、印刷レイアウトプロローグファイル内で次の変数を明示的に定義する必要があります。

NumCols	<1 印刷ページのカラムの数>
PrintWidth	<印刷領域のインチでの幅>
PrintHeight	<印刷領域のインチでの高さ>

- /NumCols 2 def
- /PrintWidth 6 def
- /PrintHeight 9 def

ロケール依存プロローグファイル

ロケール依存プロローグファイルは /usr/openwin/lib/locale/$LANG/print/prolog.ps で、通常は PostScript ファイルです。このファイルには、PostScript プロローグ情報といくつかの追加の PostScript ルーチンを定義する組み込み Type1 フォントを入れることができます。

プロローグファイルの主な目的の 1 つに、前もって mp(1) で定義され使用される一連のフォント名に対し、別名でロケールのフォントを設定することがあります。

/usr/bin/mp が prolog.ps でも動作するよう、このファイルに対するサポートが提供されています。このファイルが存在するときは優先され、下位互換性を維持するために mp.conf ファイルが走査されることはありません。

以降の mp.conf ファイルに関する節は、『*OpenWindows Localization Guide*』から引用したものです。

prolog.ps とは

prolog.ps ファイルの目的は、非汎用フォントを設定することです。アプリケーションは、前もって定義されたこれらの PostScript フォント名を使用して印刷します。プロローグファイルは、Desk Set Calendar マネージャと mp に対し、少なくとも次のフォント名を定義しなくてはなりません。

- LC_Times-Roman
- LC_Times-Bold
- LC_Helvetica
- LC_Helvetica-Bold
- LC_Courier
- LC_Helvetica-BoldOblique
- LC_Times-Italic

これらのフォントは、次の使用例でローカル (地域) の文字セットを印刷できなければなりません。

- `100 100 moveto`
- `/LC_Times-Roman findfont 24 scale font setfont`
- (ロケールのテキスト文字列) `show`

prolog.ps ファイルの例

ローカライゼーションキットには、日本の環境向けに、prolog.ps のサンプルが用意されています。また、このファイルは/usr/openwin/lib/locale/ja/print/ディレクトリにもあります。

既存の prolog.ps ファイルへの複合フォントの追加および変更方法

たとえば次の例では、LC_Base-Font という複合フォントが定義されます。

```
%
(Foo-Fine) makecodeset12
(Base-Font) makeEUCfont
%
```

LC_Base-Font は Foo-Fine と Base-Font というベースフォントの複合フォントです。Foo-Fine は、ローカルのキャラクタセットがあるフォントです。フォントの追加や変更を行うために、PostScript に精通している必要はありません。

prolog.ps ファイルの作成方法

実際の例を調べてみることが、学習への早道といえます。prolog.ps の例では、makecodeset12 と makeEUCfont の 2 つのルーチンを書く必要があります。Makecodeset12 がローカルフォントのエンコーディング情報を設定します。このルーチンはロケールによって異なる場合があります。MakeEUCfont はベースフォントとローカルフォントを組み合わせて、複合フォントを形成します。プロローグファイルの作成者は makecodeset12 と makeEUCfont を書くために、PostScript に精通している必要があります。

prolog.ps ファイルに対するサポートは、下位互換性を維持する目的でのみ継続されています。ロケールで印刷するために新たに prolog.ps ファイルの作成はせず、代わりに mp.conf を使用します。

prolog.ps の場所

パスは次のとおりです。

```
/usr/openwin/lib/locale/$LANG/print/prolog.ps
```

.xpr ファイルのカスタマイズ

これらのファイルは、デフォルトで /usr/lib/lp/locale/C/mp/ にあります。.xpr ファイルは、mp.common.ps を除いて、各 PostScript プロローグ

レイアウトファイルに対応しています。MP_PROLOGUE 環境変数を定義すると、別のプロローグディレクトリが定義できます。

これらのファイルはキーワード/値のペアとして動作します。# で始まる行はコメントとみなされます。次の説明で明示的に指示がない限り、スペースによって異なるトークンに分離されます。各 .xpr ファイルの主な 3 つの領域は、次のキーワードのペアに結び付きます。

- STARTCOMMON/ENDCOMMON
- STARTPAGE/ENDPAGE
- STARTCOLUMN/ENDCOLUMN

この 3 つの領域で、特定のキーワード/値のペアが使用できます。次に、各領域について説明します。

STARTCOMMON/ENDCOMMON のキーワード

STARTCOMMON キーワードの後と ENDCOMMON キーワードの前にあるキーワード/値のペアは、すべて印刷ページの一般的な属性を定義します。1 つのキーワードに対し有効な値が複数あるときは、/ を使用して区切ります。

ORIENTATION 0/1

0 は縦長、1 は横長の向きで印刷を行うことを示します。

PAGELENGTH <符号無し整数**>**

論理ページごとの行数を示す値。

LINELENGTH <符号無し整数**>**

1 つのカラムの行の文字数を示す値。

NUMCOLS <符号無し整数**>**

物理ページごとの論理ページ数。

HDNGFONTSIZE <符号無し整数**>**

ヘッダーフォントのポイントサイズ (デシポイント)。

BODYFONTSIZE <符号無し整数>

本文のフォントポイントサイズ (デシポイント)。

PROLOGDPI <符号無し整数>

現在の .xpr ファイルが作成される「インチ当たりのドット数」の大きさ。

YTEXTBOUNDARY <符号無し整数>

1 ページあるいは論理ページ(カラム)でテキストを印刷するときに、この y 座標によって境界が設定されます。この境界は、テキスト印刷が希望どおりの領域で行われているかどうかを確認するための追加的な検査に使用します。この境界は、Complex Text Layout や EUC の印刷で必要となります。これは対応するフォントから取得した文字の高さの情報が間違っている可能性があるためです。

STARTTEXT <符号無し整数> <符号無し整数>

ある物理ページの最初の論理ページで実際のテキスト印刷が開始される x/y 点 (デシポイント)。

PAGESTRING 0/1

1 は、見出しでページ番号の前に page の文字列を追加する必要があることを指定します。

0 は、ページ番号だけを表示するよう指定します。

EXTRAHDNGFONT "font string 1, font string 2, ... font string n”

font string 1 から font string n は、X Logical Font Description です。コンマで区切られたフォント名のリストから EXTRAHDNGFONT キーワードを分離するトークンは " で、スペースやタブではありません。見出しの印刷時には、これらのフォントが組み込みフォントより優先されます。通常は、EXTRABODYFONT を使用してプリンタ常駐フォントを割り当てます。プリンタ常駐フォントは、/usr/openwin/server/etc/XpConfig/C/print/models/<model name>/fonts ディレクトリで構成されています。fonts.dir にはプリンタ常駐フォントの XLFD が入っています。

通常は、.xpr ファイルで次のようにフォントを指定します。

"-monotype-Gill Sans-Regular-r-normal- -*-%d-*-*-p-0-iso8859-2"

%d がある場合は、.xpr ファイルにおける現在の見出しフォントのポイントサイズに、mp(1) によって置き換えられます。x 解像度と y 解像度は * で指定し、平均幅のフィールドは 0 に設定して、できれば、拡大縮小可能なフォントを選ぶように指示します。さらに細かくフォント名が指定できます。

EXTRABODYFONT　“font string 1, font string 2, ... font string n”

これは、これらのフォントがページ本体を印刷するときに使用されることを除いて、EXTRAHDNGFONT と同じです。

XDISPLACEMENT <符号付き / 符号無し整数>

ページの x 方向への遷移量。この遷移量は +ve または -ve の値になります。

YDISPLACEMENT <符号付き / 符号無し整数>

このパラメータは、y 方向に移動する以外、x の遷移量と同じです。

標準ではない余白幅のプリンタの場合や、1 ページの印刷内容を入れ換える必要があるときに、この 2 つのキーワードは便利です。

STARTPAGE/ENDPAGE

この節におけるキーワードの値のペアは、STARTPAGE キーワードと ENDPAGE キーワードに結び付きます。この節には、物理ページ 1 枚に適用する描画情報や見出し情報を入れます。物理ページ 1 枚に、複数の論理ページがある場合がありますが、これらのキーワードの間にある描画ルーチンはすべて、1 物理ページ 1 枚に 1 度だけ適用されます。

有効な描画エンティティは LINE と ARCです。XDrawLine 関数と XDrawArc 関数が、これらのキーワードの値に対して実行されます。

この領域におけるサイズは PROLOGDPI の単位で対応付けされます。角度は度の単位で示されます。

LINE x1 y1 x2 y2

/y 符号なしの座標によって、行を引くときの点の組を定義します。

ARC x y width height angle1 angle2

x と y はともに、弧の原点を示す符号なし整数です。幅 (width) と高さ (height) は、弧の幅と高さを表す、符号なしの整数です。

USERSTRINGPOS x y

符号なしの座標は、見出しでユーザー情報を印刷する位置を示します。

TIMESTRINGPOS x y

符号なしの座標は、見出しで印刷時刻を印刷する位置を示します。

PAGESTRINGPOS x y

符号なしの座標は、各印刷ページのページ文字列を印刷する位置を示します。

SUBJECTSTRINGPOS x y

符号なしの座標は、ページで主題を印刷する位置を示します。

STARTCOLUMN/ENDCOLUMN

すべてのキーワードは前述の STARTPAGE/ENDPAGE の領域と同じですが、この領域では、エントリは物理ページ 1 枚に NUMCOLS 回適用されます。

NUMCOLS が 3 である場合、物理ページの印刷可能な領域は 3 つに分割され、行、弧、あるいは見出しの装飾がページごとに 3 回になります。

新しい .xpr ファイルの作成

次に、.xpr ファイルで STARTCOMMON/ENDCOMMON 領域に対し、さまざまなキーワードの値に対する値が指定されていない場合の、mp(1) プログラムのデフォルトを示します。

- ORIENTATION 0
- PAGELENGTH 60
- LINELENGTH 80
- YTEXTBOUNDARY 3005
- NUMCOLS 01
- HDNGFONTSIZE 120
- BODYFONTSIZE 90
- PROLOGDPI 300

- STARTTEXT 135 280
- PAGESTRING 0

STARTPAGE/ENDPAGE と STARTCOLUMN/ENDCOLUMN で示される他の 2 つの領域では、デフォルトの値はありません。

`.xpr` プロローグファイルを新たに作成するときは、デフォルトと異なる値のみを指定する必要があります。

装飾がなく、縦長のフォーマットで、物理ページごとに論理ページを 4 枚使用するページを作成するときは、以下の指定をします。

- STARTCOMMON
- NUMCOLS 04
- LINELENGTH 20
- ENDCOMMON

この場合、次に示す他の 2 つの領域は必要ありません。

– STARTPAGE/ENDPAGE

– STARTCOLUMN/ENDCOLUMN

印刷ページを装飾しない場合は、これらのパラメータは必要ありません。
PROLOGDPI キーワードを指定していない限り、すべての座標はデフォルトの 300 dpi になります。出力先プリンタの解像度が異なる場合は、その解像度に合うように、プログラムによって `.xpr` ファイルの大きさが調整されます。

`.xpr` ファイルを作成するときは、用紙のサイズを事前に知っておく必要があります。用紙が米国規格となっている 8.5x11 インチの場合、解像度 300 dpi のプリンタでは、2550X3300 が全体のサイズとなります。たいていのプリンタでは、用紙の左上隅を印刷することはできず、物理的な用紙の周囲にある程度マージンが配置されます。つまり、0,0 から印刷しようとしても、ページの左上隅は印刷されません。`.xpr` ファイルを新たに作成するときは、こうした制限を考慮する必要があります。

開発ツールについてのトピック

このトピックでは、Solaris 環境における開発ツールについて記述します。次の章で構成されています。

第 10 章	`appcert` の使い方、`appcert` から報告された問題に対する対処方法、`appcert` に準拠したコードの作成方法について記述します。
第 11 章	Sun WBEM Software Developer's Toolkit (SDK) について記述します。
第 12 章	『リンカーとライブラリ』での新しい機能について記述します。
第 13 章	『*Solaris* モジューラデバッガ』での新しい情報について記述します。
第 14 章	『マルチスレッドのプログラミング』での新しい情報について記述します。

第 10 章

appcert の使用

Solaris 8 4/01 リリースで、appcert が新規に追加されました。詳細は、『システムインタフェース』を参照してください。

この章では、次の項目について説明します。

- appcert ユーティリティの目的
- appcert の実行方法と使い方
- appcert が報告した問題への対処方法
- appcert 準拠コードの作成方法

注 - 最新のマニュアルページを参照するには、man コマンドを使用してください。Solaris 8 Update リリースのマニュアルページには、「*Solaris 8 Reference Manual Collection*」には記載されていない新しい情報も提供されています。

appcert ユーティリティの目的

Solaris の新しいリリースが発表されると、ライブラリインタフェースの中には、動作が変わったり、完全になくなるものがあります。そうしたインタフェースに依存しているアプリケーションも機能しなくなってしまいます。Solaris Application Binary Interface (ABI) では、アプリケーションが使用できる、安全で安定した実行時ライブラリインタフェースを定義します。Solaris ABI に準拠して書かれたアプリケーションであれば、今後の Solaris のリリースでも安定性が確保できま

す。appcert ユーティリティは、アプリケーションが Solaris ABI に準拠しているかどうかを、開発者が確認できるように設計されています。

appcert のチェック項目

appcert ユーティリティがアプリケーションで検査するのは、次の項目です。

- 非公開シンボルの使用
- 静的リンク
- 結合されていないシンボル

非公開シンボルの使用

非公開 (private) シンボルとは、Solaris ライブラリがお互いに呼び出すときに使用する関数やデータです。非公開シンボルの意味上の動作は変わる可能性があり、シンボルは場合によっては削除されることがあります (このようなシンボルを降格されたシンボルといいます)。非公開シンボルの変更が可能な性質は、非公開シンボルに依存するアプリケーションが不安定になる原因となります。

静的リンク

Solaris ライブラリ間の非公開シンボルの呼び出しの意味が、リリースごとに変わる可能性があるため、アーカイブに対する静的リンクを作成すると、アプリケーションのバイナリの安定性が低下することになります。アーカイブの対応する共用オブジェクトファイルへ動的リンクを作成すると、この問題が回避できます。

結合されていないシンボル

appcert ユーティリティは動的リンカーを使用して、検査されるアプリケーションが使用するライブラリシンボルを解決します。動的リンカーが解決できないシンボルを、結合されていないシンボルといいます。結合されていないシンボルの原因は、LD_LIBRARY_PATH 変数の間違った設定などの環境の問題や、コンパイル時に -l*lib* オプションや -z オプションの定義を省略したなどの作成上の問題にありま

す。この例は重大とはならないものの、appcert が報告する結合されていないシンボルは、重大な問題となる場合があります。

appcert がチェックしない項目

appcert に検査させたいオブジェクトファイルがライブラリに依存している場合は、そうした依存関係がオブジェクトに記録されている必要があります。そのためには、コードのコンパイル時に、コンパイラの -l オプションを確実に使用するようにします。オブジェクトファイルが他の共用ライブラリに依存している場合は、appcert の実行時に LD_LIBRARY_PATH や RPATH を介して、それらのライブラリにアクセスできなければなりません。

appcert アプリケーションでは、マシンが 64 ビットの Solaris カーネルを実行していない場合、64 ビットのアプリケーションはチェックできません。appcert が 64 ビットのアプリケーションをチェックしているときは、静的リンクのチェックは行われません。

appcert ユーティリティでは次の項目は検査できません。

- 完全にあるいは部分的に静的にリンクされているオブジェクトファイル。完全に静的にリンクされているオブジェクトは、不安定であると報告されます。
- アクセス権が設定されていない実行可能ファイル。appcert ユーティリティはこうした実行可能プログラムの処理を飛ばします。実行アクセス権のない共用オブジェクトは、通常どおり検査されます。
- ユーザー ID が root に設定されているオブジェクトファイル。
- シェルスクリプトなどの ELF 以外の実行可能プログラム。
- C 言語以外の Solaris インタフェース。コード自体は C 言語である必要はありませんが、Solaris ライブラリへの呼び出しは C 言語でなければなりません。

appcert での作業

appcert を使用してアプリケーションをチェックするには、次のように実行します。

```
appcert object | directory
```

object|*directory* は、次のどちらかです。

- appcert に検査させたいオブジェクトの完全なリスト
- そうしたオブジェクトがあるディレクトリの完全なリスト

注 - チェックされるアプリケーションが実行される環境とは異なる環境で appcert を実行する場合、appcert ユーティリティは Solaris ライブラリインタフェースへの参照を正しく解決できない可能性があります。

---

appcert ユーティリティは Solaris 実行時リンカーを使用して、実行可能プログラムや共用オブジェクトファイルごとにインタフェースへの依存関係のプロファイルを作成します。このプロファイルを使用すると、アプリケーションが依存する Solaris システムインタフェースが判別できます。プロファイルに記述された依存関係は Solaris ABI と比較され、準拠しているかどうかが確認されます (非公開インタフェースがあってはいけません)。

appcert ユーティリティは再帰的にディレクトリを検索してオブジェクトファイルを探しますが、ELF 以外のオブジェクトファイルは無視します。appcert によるアプリケーションのチェックが完了すると、標準出力 (stdout、通常は画面) に終了報告が出力されます。同じ内容が作業用ディレクトリ、通常は /tmp/appcert.*pid* の、Report という名前のファイルに書き込まれます。サブディレクトリ名の *pid* は appcert のプロセス ID で 1〜6 桁の数字で示されます。appcert が出力ファイルを書き込むディレクトリ構造の詳細については、108ページの「appcert の結果」を参照してください。

appcert のオプション

次のオプションによって appcert ユーティリティの動作が変更できます。コマンド行では、appcert コマンドの後、*object*|*directory* オペランドの前に、オプションを入力します。

-B

バッチモードで appcert を実行します。

バッチモードでは、appcert が作成するレポートの各行に、チェックしたバイナリが記録されます。

PASS で始まる行は、その行に指定されているバイナリに対して appcert の警告を発しなかったことを示します。

FAIL で始まる行は、そのバイナリで問題が見つかったことを示します。

INC で始まる行は、その行に指定されているバイナリが、完全にはチェックできなかったことを示します。

`-f` ***infile***

infile ファイルはチェックの対象となるファイルのリストで、1 行に 1 つのファイル名を指定します。コマンド行でファイルが指定されている場合は、そのファイルと共にチェックの対象となります。このオプションを使用する場合は、コマンド行でオブジェクトやディレクトリを指定する必要はありません。

`-h`

`appcert` の使用方法の情報を出力します。

`-L`

デフォルトでは、`appcert` はアプリケーションの共用オブジェクトをすべてチェックし、共用オブジェクトがあるディレクトリを `LD_LIBRARY_PATH` に追加します。`-L` オプションを指定すると、この動作が無効になります。

`-n`

デフォルトでは、`appcert` はディレクトリを検索してチェックするバイナリを探すときに、シンボリックリンクをたどります。`-n` オプションを指定すると、この動作が無効になります。

`-S`

`LD_LIBRARY_PATH` に Solaris ライブラリディレクトリである `/usr/openwin/lib` と `/usr/dt/lib` を追加します。

`-w` ***working_dir***

ライブラリコンポーネントの実行と、一時ファイルの作成を行うディレクトリを指定します。このオプションが指定されていない場合、`appcert` は `/tmp` ディレクトリを使用します。

appcert の結果

appcert ユーティリティによる、アプリケーションのオブジェクトファイルに対する分析結果は、appcert ユーティリティの作業用ディレクトリ (通常は /tmp) に作成されるサブディレクトリの下に置かれるいくつかのファイルに書き込まれます。作業用ディレクトリに作成されるサブディレクトリ名は appcert.*pid* で、*pid* は appcert のプロセス ID です。

Index	チェック済みのバイナリと、そのバイナリに対する appcert の出力が置かれるサブディレクトリとの間の対応が入ります。
Report	appcert の実行時に stdout に表示された、終了報告のコピーが保存されます。
Skipped	appcert がチェックするように要求されたが処理を飛ばさざるを得なかったバイナリのリストが入ります。各バイナリが処理を飛ばされた理由も記載されます。理由には、次のものがあります。 ■ ファイルがバイナリのオブジェクトでない ■ ファイルをユーザーが読み取ることができない ■ ファイル名にメタキャラクタが含まれている ■ ファイルに実行ビットが設定されていない
objects/***object_name***	objects のサブディレクトリの下には、appcert が検査したオブジェクトごとにサブディレクトリが作られます。それぞれのサブディレクトリには、次のファイルが入っています。 check.demoted.symbols 降格された Solaris シンボルの可能性があると appcert が判断したシンボルのリストが入っています。 check.dynamic.private

オブジェクトが直接結合されている、非公開 Solaris シンボルのリストが入っています。

`check.dynamic.public`

オブジェクトが直接結合されている、公開 (public) Solaris シンボルのリストが入っています。

`check.dynamic.unbound`

`ldd -r` の実行時に、動的リンカーで結合されなかったシンボルのリストが入っています。`ldd` によって返された `file not found` と記載された行も含まれます。

`summary.dynamic`

`appcert` が検査したオブジェクトの動的結合の要約を、印刷用に整形したものが入っています。これには、各 Solaris ライブラリから使用された公開シンボルや非公開シンボルのテーブルも含まれています。

`appcert` の終了時には、次の 4 つの値のうちの 1 つが返されます。

0 バイナリが不安定となり得る要因は `appcert` では見つかりませんでした。

1 `appcert` ユーティリティは正常に実行されませんでした。

2 `appcert` がチェックしたオブジェクトの一部に、バイナリの安定性に問題がある可能性があります。

3 `appcert` ユーティリティはチェックするバイナリオブジェクトを見つけることができませんでした。

appcert が報告した問題に対する対処方法

- 非公開シンボルの使用。Solaris のリリースごとに、非公開シンボルの動作が変わっていたり、非公開シンボルがなくなっている場合があるため、非公開シンボルに依存するアプリケーションは、開発された Solaris リリースとは異なるリリース上では実行できない可能性があります。appcert によって、アプリケーションで非公開シンボルが使用されていると報告された場合は、非公開シンボルを使用しないでアプリケーションを書き直す必要があります。
- 降格されたシンボル。降格されたシンボルとは、後の Solaris リリースで削除された、または有効範囲が局所的 (ローカル) となる Solaris ライブラリの関数やデータ変数を指します。そうしたシンボルを直接呼び出すアプリケーションは、ライブラリがそのシンボルを外部参照可能としないリリースでは実行できなくなります。
- 結合されていないシンボル。結合されていないシンボルは、appcert によって呼び出されたときに動的リンカーが解決できなかったアプリケーションによって参照されるライブラリシンボルを指します。結合されていないシンボルは、必ずしもバイナリの安定性が低いことを示すものではありませんが、降格されたシンボルに対する依存関係のような、重大な問題を示している可能性があります。
- 旧式のライブラリ。旧式のライブラリは将来のリリースで Solaris から削除される可能性があります。appcert ユーティリティはそうしたライブラリが使用されていると注意を喚起します。今後のリリースでそのライブラリがなくなると、それに依存するアプリケーションが動作しなくなるからです。こうした問題を避けるため、旧式のライブラリのインタフェースは使用しないでください。
- *sys_errlist* または *sys_nerr* の使用。 sys_errlist シンボルおよび sys_nerr シンボルの使用は、sys_errlist 配列の終端より後に対して参照が行われる場合があるため、バイナリの安定性を低下させる可能性があります。こうした危険を回避するため、代わりに strerror を使用してください。
- 強いシンボルや弱いシンボルの使用。今後の Solaris のリリースでは動作が変わる可能性があるため、弱いシンボルと関連付けられている強いシンボルは、非公開として予約されています。アプリケーションは、弱いシンボルを直接的にのみ参照しなければなりません。強いシンボルの例に _socket がありますが、これは弱いシンボル socketと関連付けられています。

第 11 章

WBEM SDK

Solaris 8 4/01 リリースで WBEM SDK が新規に追加されました。WBEM SDK についての詳細は、『*Sun WBEM SDK* 開発ガイド』を参照してください。

注 - 最新のマニュアルページを参照するには、man コマンドを使用してください。Solaris 8 Update リリースのマニュアルページには、「*Solaris 8 Reference Manual Collection*」には記載されていない新しい情報も提供されています。

Web-Based Enterprise Management (WBEM)

Web-Based Enterprise Management (WBEM) では、複数のプラットフォーム上のシステム、ネットワーク、およびデバイスを Web ベースで管理するための標準が提供されています。Sun WBEM Software Developer's Toolkit (SDK) では、ソフトウェア開発者は、Solaris オペレーティング環境におけるリソースを管理する、標準ベースのアプリケーションの作成が可能になります。また、開発者はこのツールキットを使用して、データにアクセスするときに管理対象のリソースと通信するプログラムである、プロバイダを作成することもできます。Sun WBEM SDK には、Common Information Model (CIM) のリソースの記述や管理を行うためのクライアントアプリケーションプログラミングインタフェース (API: Client Application Programming Interface)、および管理対象リソースの動的データの取得や設定を行うためのプロバイダ API が含まれています。また、Sun WBEM SDK では、システム上で管理対象リソースの作成や表示を行うための Java アプリケーションである CIM WorkShop、および WBEM のクライアントプログラムとプロバイダプログラムのサンプル集も提供しています。

第 12 章

『リンカーとライブラリ』の更新

Solaris 8 1/01 および 10/00 リリースで『リンカーとライブラリ』が更新されました。次の表は、この変更点を記載したものです。このマニュアルは、『リンカーとライブラリ』を参照してください。

注 - 最新のマニュアルページを参照するには、man コマンドを使用してください。Solaris 8 Update リリースのマニュアルページには、「*Solaris 8 Reference Manual Collection*」には記載されていない新しい情報も提供されています。

『リンカーとライブラリ』の変更点

表 12–1 『リンカーとライブラリ』の更新事項

『リンカーとライブラリ』の Solaris 8 1/01 リリースでの更新事項	■ dladdr(3DL) から入手できるシンボリック情報が、dladdr1() の導入により拡張されました。 ■ 動的オブジェクトの $ORIGIN が dlinfo(3DL) から入手できます。 ■ crle(1) で作成された実行時構成ファイルの管理が、構成ファイルの作成に使用されたコマンドラインオプションの表示によって簡単になりました。また、拡張機能も利用できます (–u オプションを参照)。 ■ 実行時リンカーおよびデバッガインタフェースが拡張され、プロシージャリンクテーブルエントリの解決を検出できるようになりました。この拡張は、新しいバージョンナンバーで識別することができます。「エージェント操作」の節を参照してください。この更新により rd_plt_info_t 構造体が機能拡張されます。「プロシージャのリンクテーブルのスキップ」の節を参照してください。 ■ 新しい mapfile セグメント記述子 STACK を使用してアプリケーションスタックを非実行可能ファイルに定義することができます。「セグメントの宣言」の節を参照してください。
『リンカーとライブラリ』の Solaris 8 10/00 リリースでの更新事項	■ 実行時リンカーが環境変数 LD_BREADTH を無視します。「初期設定および終了ルーチン」の節を参照してください。 ■ 実行時リンカーおよびそのデバッガインタフェースが拡張され、実行時分析とコアファイル分析の性能が向上しました。この更新は、新しいバージョン番号で識別されます。「エージェント操作」の節を参照してください。この更新により、rd_loadobj_t 構造体の rl_flags、rl_bend、および rl_dynamic フィールドが拡張されました。「読み込み可能オブジェクトの走査」の節を参照してください。 ■ ディスプレイスメント再配置されたデータがコピー再配置で使用されるか、使用される可能性があることを検査する機能が提供されるようになりました。「ディスプレイスメント再配置」の節を参照してください。 ■ 64 ビットフィルタが、リンカーの -64 オプションを使用してマップ (対応付け) ファイルから単独で構築できるようになりました。「標準フィルタの生成」の節を参照してください。 ■ $ORIGIN 動的文字列トークンの拡張がなぜセキュアアプリケーション内に限定されるのかの説明が追加されました。「セキュリティ」の節を参照してください。 ■ 動的オブジェクトの依存関係の検索に使用される検索パスを、dlinfo(3DL) を使用して調べることができるようになりました。 ■ dlsym(3DL) と dlinfo(3DL) 検索の方法が新しいハンドル RTLD_SELF によって拡張されました。 ■ 動的オブジェクトの再配置に使用される実行時シンボル検索メカニズムは、各動的オブジェクト内に直接結合情報を確立することによって、大幅に削減されるようになりました。「外部結合」と「直接結合」の節を参照してください。

第 13 章

『*Solaris* モジューラデバッガ』の更新

Solaris 8 10/00 リリースで『*Solaris* モジューラデバッガ』が更新されました。このマニュアルは、『*Solaris* モジューラデバッガ』を参照してください。

注 - 最新のマニュアルページを参照するには、man コマンドを使用してください。Solaris 8 Update リリースのマニュアルページには、「*Solaris 8 Reference Manual Collection*」には記載されていない新しい情報も提供されています。

『*Solaris* モジューラデバッガ』の変更点

- 第 3 章の「演算機能の拡張」の節が更新され、単項演算が含まれるようになりました。
- 技術的なマイナーエラーが修正されました。

第 14 章

『マルチスレッドのプログラミング』の更新

Solaris 8 1/01 リリースで『マルチスレッドのプログラミング』が更新されました。このマニュアルは、『マルチスレッドのプログラミング』を参照してください。

注 - 最新のマニュアルページを参照するには、man コマンドを使用してください。Solaris 8 Update リリースのマニュアルページには、「*Solaris 8 Reference Manual Collection*」には記載されていない新しい情報も提供されています。

SPARC:『マルチスレッドのプログラミング』の変更点

『マルチスレッドのプログラミング』が更新され、バグ ID 4308968、4356675、4356690 が修正されました。

第 15 章

インタフェースの開発についてのトピック

このトピックでは、Solaris オペレーティング環境におけるインタフェースの開発について記述します。次の章で構成されています。

第 16 章	『システムインタフェース』での変更点について記述します。

第 16 章

『システムインタフェース』の更新

Solaris 8 6/00 リリースで『システムインタフェース』が更新されました。このマニュアルは、『システムインタフェース』を参照してください。

注 - 最新のマニュアルページを参照するには、man コマンドを使用してください。Solaris 8 Update リリースのマニュアルページには、「*Solaris 8 Reference Manual Collection*」には記載されていない新しい情報も提供されています。

『システムインタフェース』の変更点

『システムインタフェース』が更新され、旧版でのバグが修正されました。テキストおよびサンプルソースコードのタイプミスが修正されています。

第 17 章

Java 2 Standard Edition および JDK についてのトピック

このトピックでは、Java 2 Standard Edition および JDK の機能について記述します。次の章で構成されています。

第 18 章	Solaris 8 Update リリースにおける Java 2 SDK および JDK の各リリースでの機能更新の詳細を記述します。

第 18 章

Java 2 Standard Edition および JDK の新しい機能について

Solaris 8 4/01 リリースで、J2SE 1.3.0 とも呼ばれる Java 2 SDK Standard Edition バージョン 1.3.0 が含まれるようになりました。ここでは、その新しい機能について説明します。また、以前の Update リリースにおける J2SE リリースおよび JDK リリースについても説明します。ここに記載する情報は、『*Java 2 SDK* 開発ガイド *(Solaris* 編*)*』に記載されている情報を補足するものです。以下では、各 Update リリースにおける新しい機能について記述します。

注 - 最新のマニュアルページを参照するには、`man` コマンドを使用してください。Solaris 8 Update リリースのマニュアルページには、「*Solaris 8 Reference Manual Collection*」には記載されていない新しい情報も提供されています。

Java 2 SDK Standard Edition バージョン 1.3.0

J2SE 1.3.0 は Java 2 SDK のアップグレード版です。このソフトウェアには、次のような新機能や拡張機能が組み込まれています。

- パフォーマンスの向上

  Java HotSpot 技術と最適化された実行時ライブラリにより、J2SE 1.3.0 は前回の Java 2 SDK のバージョンに比べ、多くの機能で高速化を実現しています。

- Web への容易な展開

アプレットのキャッシュやオプションパッケージの自動インストールなど、J2SE 1.3.0 の Java Plug-In コンポーネントによる機能が新たに追加されたため、プログラムを Web 上に展開するときの時間が短縮され柔軟性が向上しました。

- 企業レベルでの相互運用性

  J2SE 1.3.0 にRMI/IIOP と Java Naming and Directory Interface が追加されたため、Java 2 プラットフォームの相互運用性が向上しました。

- セキュリティの強化

  RSA 電子署名、動的信用管理、X.509 証明書、Netscape 署名ファイルの検証が新たにサポートされたため、開発者はさまざまな手段で電子データを保護できるようになりました。

- Java サウンド

  J2SE 1.3.0 には強力なサウンド API が新たに提供されています。これまでのプラットフォームのリリースでは、オーディオのサポートがオーディオクリップの基本的な再生に限定されていました。今回のリリースでは、低レベルのオーディオサポートに対する標準のクラスやインタフェースが、Java 2 プラットフォームによってはじめて定義されました。

- 拡張された API と開発しやすさの向上

  開発側からの要望に応えて、J2SE 1.3.0 は Java 2 プラットフォームに多彩な機能を追加しています。追加された機能によって、プラットフォームの機能性がさらに向上し、一層強力なアプリケーションが開発できるようになりました。また、新規の機能の多くは、開発工程の短縮や効率化を実現する機能です。

それぞれの機能について、次に詳しく説明します。J2SE 1.3.0 の新しい機能の詳細は、`http://java.sun.com/j2se/1.3/docs` に掲載されている Java 2 Platform に関するマニュアルを参照してください。Java 2 Platform のマニュアルには J2SE 1.3.0 の API 仕様が組み込まれています。

パフォーマンスの向上

Java 2 SDK バージョン1.3.0 ではさまざまな機能拡張が行われ、パフォーマンスが向上しています。変更点としては、Java HotSpot Client Virtual Machine (VM) と Java HotSpot Server VM の追加などがあります。この 2 つは、高性能な Java HotSpot 技術を実装するものです。Java HotSpot Client VM はクライアントシステムのパフォーマンスを最大限に向上させるためのチューニングが行われ、起動時間とメモリーの使用量の分野でパフォーマンスが向上しています。Java HotSpot Server VM

はチューニングによりプログラム実行速度のパフォーマンスが最大になるように最適化され、起動時間やメモリーの使用量がさほど問題にはならないサーバーアプリケーションを対象としています。

J2SE 1.3.0 にはまた、新たにチューニングされたクラスライブラリが組み込まれ、実行時のパフォーマンスが改善されています。

Web への容易な展開

アプレットキャッシュ機能

J2SE 1.3.0 で新たにアプレットキャッシュ機能が提供され、アプレットをローカルキャッシュに保存することによって、使用頻度の高いアプレットの高速なロードや起動が可能になりました。アプレットが 2 回以上ダウンロードされたときは、ローカルのアプレットキャッシュに保存されている可能性があります。保存しておくことで、その後アプレットが必要になるたびに、ネットワーク経由でアプレットを表示してダウンロードする必要がなくなり、ローカルにキャッシュされているアプレットを使用できます。

この機能は、大容量かつ使用頻度の高いアプレットにとっては大変便利です。たとえば、多くの企業のアプレットがメガバイトのサイズに達し、それほどのサイズのアプレットになると、ネットワークからロードするのに何十分もかかってしまいます。新しいアプレットキャッシュ機能ではダウンロード時間がなくなるので、ビジネスシーンでいままで以上に強力なアプレットを数多く使用することができます。

オプションパッケージの自動適用

J2SE 1.3.0 では、オプションパッケージの自動適用もサポートされています。オプションパッケージは、Java 2 Platform Standard Edition には含まれていない、特殊なプログラミング用に使用する開発者向けの機能や API のセットであり、別途入手できます。例としては、Java Media Framework 技術や JavaHelp™ オプションパッケージなどあります。

J2SE 1.3.0 より前のバージョンでは、アプレットでオプションパッケージを使用している場合、アプレットを実行したいすべてのクライアントに最新のバージョンのオプションパッケージがインストールされていることが前提とされていました。したがって、クライアントに適切なオプションパッケージがインストールされてい

なければ、アプレットが意図しない動作をするか、あるいはまったく実行できない状態でした。

J2SE 1.3.0 では、アプレットが必要とするオプションパッケージのバージョンやベンダー情報を指定することができます。開発者は必要なオプションパッケージの最新バージョンを、次の状況のいずれかに当てはまる場合にダウンロードできる URL を、アプレットに指定することができます。

- ローカルにはオプションパッケージがまだインストールされていない。
- オプションパッケージはインストールされているが、古いバージョン番号である。
- オプションパッケージはインストールされているが、指定されたベンダーのものではない。

J2SE 1.3.0 では、オプションパッケージに組み込まれている独自のインストーラや Java 言語のインストーラプログラムがサポートされており、ネットワークから新しいバージョンのオプションパッケージを取り込むと、自動的にインストーラプログラムを起動します。

企業レベルでの相互運用性

Java IDL および RMI-IIOP

J2SE 1.3.0 には、CORBA 技術をサポートする重要な拡張機能が 2 つ追加されています。Java 言語で書かれた運用 CORBA IDL コンパイラとRMI over IIOP (RMI-IIOP) APIです。CORBA Interface Definition Language (IDL) は、分散システム用のインタフェースだけを定義する言語です。CORBA はインタフェースの定義に中立的な言語を使用しているため、複数の言語をサポートすることができます。標準 Java 言語バインドに言語的に中立的な CORBA IDL をコンパイルする IDL コンパイラが Java 2 SDK Standard Edition に組み込まれたのは、これが初めてです。これらの言語バインドは Java IDL Object Request Broker (ORB) と連携して、Java プログラミング言語で従来の CORBA プログラミングをサポートします。

Java プラットフォームのバージョン 1.1 から、Remote Method Interface (RMI) のおかげで、プログラマは分散環境のインタフェースを直接 Java 言語で書くことができるようになりましたが、RMI は独自の通信プロトコルを使用していたため、プログラマは RMI を使用するときに、他の言語で書かれたオブジェクトとの通信能力を断

念しなければなりませんでした。RMI-IIOP では Java IDL ORB が使用されているため、標準の CORBA 通信プロトコルである Internet InterORB Protocol (IIOP) を RMI と一緒に使用することができます。すべての通信に IIOP が使用されているため、C++ などの他言語で書かれたオブジェクトも、RMI-IIOP 分散オブジェクトと通信できます。さらに RMI は、Java プログラミング言語のインタフェースを CORBA IDL へマップ (対応付け) するときの CORBA 標準として認められました。他の言語でのプログラミングをやりやすくするため、CORBA 標準 IDL は RMI が有効なクラスから生成することができます。既存の RMI プログラムは、通常はわずかな変更で、IIOP プロトコルを使用するように変換できます。

RMI-IIOP では RMI のプログラミングのしやすさに加えて、他言語で書かれたソフトウェアと JavaIDL の CORBA 準拠の通信が可能です。RMI プログラマは多少の制限事項に従えば、CORBA の IIOP 通信機能を使用して、あらゆる種類のクライアントとでも通信できるようになりました。クライアントが全体を Java プログラミング言語で書いたものでも、他の CORBA に準拠している言語で書かれたコンポーネントで構成されたものでも、通信が可能です。

Java Naming and Directory Interface (JNDI) API

J2SE 1.3.0 の新しい Java Naming and Directory Interface (JNDI) API を使用すると、開発者は Java プログラミング言語で書かれたアプリケーションに、ネーミング機能およびディレクトリ機能が追加できます。JNDI の開発目的は、特定のネーミングサービスやディレクトリサービスの実装に依存することなく、異種混在の企業のネーミングサービスおよびディレクトリサービスにシームレスに接続することです。したがって、新規のサービス、拡大中のサービス、すでに展開済みのサービスなど、さまざまなサービスに一貫した方法でアクセスできます。この業界標準のインタフェースを使用すれば、開発者は J2SE 1.3.0 で強力かつ移植性がある、ディレクトリが使用可能なアプリケーションの構築が可能になります。

JNDI アーキテクチャは、API と Service Provider Interface (SPI) で構成されています。Java アプリケーションはこの API を使用して、さまざまなネーミングサービスやディレクトリサービスにアクセスします。SPI では、さまざまなネーミングサービスやディレクトリサービスを透過的にプラグインできるため、Java アプリケーションがサービスにアクセスできるようになります。J2SE 1.3.0 のリリースの JNDI には、次のサービスにアクセスできる、サービスプロバイダが同梱されています。

- Lightweight Directory Access Protocol (LDAP) — ディレクトリサービスにアクセスするためのインターネット標準

- Common Object Services (COS) Name Server — CORBA オブジェクト参照を格納するためのネームサーバー
- RMI レジストリサービスプロバイダ — RMI リモートのオブジェクトを格納するためのネームサーバー

セキュリティの強化

J2SE 1.3.0 で提供されているセキュリティの拡張機能を使用すると、開発者はさらに多くのツールで、自らの技術投資を自在に保護することができます。RSA 署名が新たにサポートされ、J2SE 1.3.0 の拡張された動的信頼管理機能により、Web ベースの展開が非常に簡単になりました。

RSA 署名に対するサポート

J2SE 1.3.0 では暗号サービスプロバイダが提供されています。これは Web で配信される電子署名ソフトウェアに対応して、広く使用されている RSA 署名をサポートするものです。VeriSign や Thawte などの署名を含め、標準的な RSA 証明書がサポートされています。

J2SE 1.3.0 より前のバージョンでは、Java プラットフォームユーザーが RSA 証明書を使用したいと考えた場合、自分で RSA サービスプロバイダを書くか、あるいは Sun 以外から RSA サービスプロバイダを購入する必要がありました。今では RSA プロバイダが J2SE 1.3.0 に標準として組み込まれています。

動的信頼管理

J2SE 1.3.0 の新しい動的信頼管理機能では、ポップアップダイアログが表示されてユーザはアプレット署名者を検証できるため、署名されたアプレットを実行する各クライアントにセキュリティキーファイルを配布する必要がなくなりました。

以前は、ユーザーが通常は禁じられている操作をアプレットにさせようとして、信頼するソースからのアプレットに追加のセキュリティ許可を与えようとすると、アプレットの信頼するソースの証明書を認識できるように、信頼する署名者の証明書のローカルキャッシュを事前に設定しておく必要がありました。この作業は、アプレットが実行される可能性があるクライアントマシンのすべてで行うことになります。

J2SE 1.3.0 には、アプレットのコードソースからアプレットの署名者を抽出してブラウザに渡す機能によってより有効な対応策を提供しています。ブラウザはその大元の証明書までずっと証明書の連鎖を検証して、ブラウザの持つ、信頼する大元の

証明書のデータベースに、その大元の証明書があるかどうかを調べます。そこに大元の証明書があれば、認証された署名者の連鎖がブラウザに表示され、ユーザーはアプレットに対するセキュリティの制限事項をすべて解除できます。

公開鍵証明書に対するサポートの向上

J2SE 1.3.0 では、X.509 公開鍵証明書に対するサポートが強化されています。最近提案された標準プロトコル (RFC 2459) で必須のあるいは推奨されている X.520 の属性がすべてサポートされるようになりました。また、J2SE 1.3.0 は相対識別名内で、複数の属性 / 値の断定を処理できます。

Java サウンド

Java Sound API では、Java プログラムでオーディオと Musical Instrument Digital Interface (MIDI) データの取り込み、処理、再生ができます。これらの新しい機能を使用すると、開発者は次のような新しいタイプのアプリケーションが作成できます。

- 会議や電話通信のアプリケーションなどを含む、通信フレームワーク。
- エンドユーザー向けコンテンツ配信システム。これらのシステムは、単純なデスクトップのメディアプレーヤーからストリーム化された音楽の配信システムや、ライブイベントのブロードキャスト音声アプリケーションにまで及びます。
- ユーザーの操作に応えて動的にサウンドを生成するゲームや Web サイトなど、対話型のアプリケーション。
- オリジナルのオーディオや音楽のコンテンツを作成したり編集したりするためのツールやツールキット。

Java Sound API は効率的なサウンドエンジンによってサポートされており、プラットフォームでの高品質なオーディオミキシングや MIDI 合成機能が保証されています。J2SE 1.3.0 に組み込まれている実装では、具体的には次の機能がサポートされています。

- オーディオファイルフォーマット： AIFF、AU、および WAV
- 音楽ファイルフォーマット： MIDI タイプ 0 とタイプ 1 および Rich Music Format
- オーディオコーデック： u-law と a-law
- オーディオデータフォーマット：モノラルとステレオでの 8 ビットと 16 ビットのオーディオ、サンプルレートは 8 kHz から 48 kHz まで。

- ソフトウェアでの MIDI ウェーブテーブルの合成とシーケンシングとハードウェア MIDI デバイスへのアクセス
- 最大 64 チャネルがデジタルオーディオと合成 MIDI 音楽のミキシングおよびレンダリングが可能な、オールソフトウェアミキサー。

さらに、開発者が現在の実装の機能を拡張するときに使用できるサービスプロバイダインタフェースが API で定義されています。また、ユーザーは上記以外のファイルフォーマット、コーデック、デバイスをサポートするモジュールをインストールできます。API には、システムで現在利用できるリソースに対し、照会やアクセスができる手段も提供されています。

拡張された API と開発しやすさの向上

AWT の拡張機能

J2SE 1.3.0 には、新しいロボット API が組み込まれています。これは、Abstract Window Toolkit (AWT) や Swing のテストの自動化を行うために設計されたものです。ロボット API を使用すると、Java プログラミング言語で書かれたコードで、低レベルのネイティブのマウスやキーボードの入力イベントが生成できます。イベントはオペレーティングシステムレベルで生成されるため、他の AWT への実際のユーザー入力と区別することはできません。

ロボット API は主としてテストの容易性を向上させるため設計されたものですが、次のような利点もあります。

- 利用のしやすさ (アクセシビリティ) を考慮したアプリケーションでは、さらに多くのフィードバックが可能になります。たとえば、ユーザーが音声コマンドを使用して画面オブジェクトを操作する場合に、マウスポインタを動かして、操作中のオブジェクトを示すこともできます。
- ロボット API では、コンピュータを利用したトレーニング (CBT) などのデモを行うアプリケーションが作成できます。

J2SE 1.3.0 では、印刷に対する API も改善されています。新しい印刷 API では、開発者はプラットフォーム固有の機能を使用して AWT コンポーネントから簡単に印刷できます。新しい API を使用すると、開発者は印刷ジョブのプロパティを制御することができます。制御可能なプロパティには、出力先、コピー数、ページ範囲、ページサイズ、方向、印刷の品質などがあります。

Java 2D 技術の拡張

J2SE 1.3.0 では、同じアプリケーションによる複数のモニター上の GUI フレームとウィンドウのレンダリングがサポートされるようになりました。Java 2D™ API では、マルチスクリーンの次の 3 つの構成がサポートされています。

- 複数の独立したスクリーン
- 複数のスクリーンで、1 つのスクリーンが一次スクリーンとなり、他のスクリーンでは一次スクリーンに表示されているもののコピーが表示される
- 仮想デスクトップを形成する複数のスクリーン

J2SE 1.3.0 の新しい動的フォントロード API を使用すると、開発者は実行時に TrueType フォントの作成やロードが行えます。開発者は Java 2D API を使用して、動的にロードされたフォントに対し、サイズ、スタイル、変形などの希望する機能を与えることができます。

J2SE 1.3.0 の Java 2D API では、Portable Graphics Network (PGN) フォーマットがサポートされるようになりました。このフォーマットは、ラスターイメージを劣化することなく可搬性のある方法で保存できる、柔軟で広範囲な非独占的なファイルフォーマットです。PGN ではグレースケール、索引付きのカラー、トゥルーカラーイメージをサポートしており、アルファチャネルを利用できます。

Java Platform Debugger Architecture (JPDA)

JPDA 技術とは複層構造のデバッグアーキテクチャで、プラットフォーム、仮想マシンの実装、および J2SE バージョンの枠を超えて実行できるデバッガアプリケーションを、ツール開発者が簡単に作成できます。

JPDA には次の 3 つの層があります。

- JVMDI - Java Virtual Machine Debug Interface

  VM がデバッグ時に提供する必要があるデバッグサービスを定義します。

- JDWP - Java Debug Wire Protocol

  デバッグ中のプロセスと、Java Debug Interface を実装するデバッガフロントエンドの間で転送される、情報と要求のフォーマットを定義します。

- JDI - Java Debug Interface

  ツール開発者がリモートデバッガアプリケーションを書くときに、簡単に使用できる高水準の Java プログラミング言語インタフェースを定義します。

国際化

J2SE 1.3.0リリースおける国際化の向上によって、開発者は自国ユーザー向けにアプリケーションを今まで以上に柔軟にローカライズできるようになりました。ここでは、2 つの新しい機能について説明します。

入力方式とは、アプリケーションに対してテキスト入力を生成するため、キー入力や話しかけるなどのユーザーの操作を解釈するソフトウェアコンポーネントです。国際的なロケールでは、入力方式はテキストの入力に重要な役割を果たします。キーボードから直接入力できる英文テキストと異なり、日本語や中国語などの言語でのテキスト入力には、さらに高度な入力方式のフレームワークが必要となります。J2SE 1.3.0 では、開発者がそうした作業に対応するために必要となる強力なツール群が提供されています。

最新のテキスト編集コンポーネントでは、テキストが最終的に表示される文書の文脈内に、入力されたテキストが表示されます。これを「入力位置内 (on-the-spot)」入力といい、Java 2 Platform では常にサポートされてきました。

J2SE 1.3.0 では、中国などの国で好評な「入力位置下 (below-the-spot)」という、第二の入力スタイルに対するサポートが追加されています。「入力位置下」テキスト編集では、入力済みのテキストは別の編集ウィンドウに表示され、そのウィンドウはテキストの挿入位置の近くに自動的に配置されます。

開発者としては、入力方式のフレームワークの一部として表示されるウィンドウを変更したり、カスタマイズしたいと考える可能性があります。その場合、J2SE 1.3.0 では、入力方式のエンジンである Service Provider Interface (SPI) に対応する新しい API が提供されているため、開発者は自在に変更やカスタマイズができます。SPI を使用して、開発者は、ソフトウェアの必要性に応じて独自の入力方式のエンジンを作成できます。

国際ロケールに対する新たなサポートの例は他にもあります。J2SE 1.3.0 では、アラビア語やヘブライ語などのロケール用に、ツールバーやメニューバーを右から左という方向になるように、アプリケーションのフレームやダイアログボックスをレンダリングすることができます。

プラットフォームライブラリやツールのその他の機能拡張

J2SE 1.3.0 では、Sun と協力関係にある企業との協議の結果や開発者からの要望に応えて、新たな機能がプラットフォームや Java 2 SDK ツールスイートに追加されています。拡張機能には、次のものがあります。

- 新しい javac コンパイラ

J2SE 1.3 では、javac コンパイラがゼロから再実装され、前回のバージョンの Java 2 SDK のコンパイラと比べて、多くのアプリケーションを高速で処理できます。

- 動的なプロキシクラス

  J2SE 1.3 には、動的プロキシクラスに対する新しい API が組み込まれています。動的なプロキシクラスは、実行時に指定される一連のインタフェースを実装するもので、そのクラスのインタフェースを通してメソッド呼び出しはコード化され、別のオブジェクトに統一的なインタフェースを介して振り分けられます。したがって、動的プロキシクラスを使用して、一連のインタフェースに対する型安全 (type-safe) なプロキシオブジェクトが作成でき、コンパイル時ツールなどを使用してプロキシクラスを事前に生成しておく必要がありません。インタフェース API が示されたオブジェクトに対し型安全な呼び出しの反射的な振り分けを提供したい開発者にとって、動的なプロキシクラスは便利です。

  たとえば動的プロキシクラスを使用して、任意のイベントリスナーインタフェースを複数実装するオブジェクトを作成し、そのイベントをログファイルに書き込むなど、一貫した方法で多種多様なイベントを処理することができます。

- コレクション API の拡張

  コレクション API の J2SE 1.3 バージョンが使いやすくなりました。1.3 のコレクション API には、List や Mapに対応した便利なメソッドやコピーコンストラクタが組み込まれています。

- 拡張 Java 基本クラス / Swing 機能

  J2SE 1.3.0 の設計時には、Java 基本クラス API の Swing コンポーネントのチューニングや機能拡張に重点が置かれました。また、Swing ライブラリの性能のチューニングに加えて、いくつかの分野で新しい JFC/Swing 機能が Swing ライブラリに追加されました。一例として、軽量テーブルコンポーネントの可変的な高さの行が、新たにサポートされるようになりました。

- 数字ライブラリやユーティリティライブラリの改善

  J2SE 1.3.0 には、同じ API を持つ数学に関連した 2 つのクラス、`Math` と `StrictMath` が組み込まれています。`StrictMath` は、数値演算に対してビット単位で再現可能な結果を返すものとして、その保証を必要とする開発者向けに定義されています。一方、`Math` クラスの実装は、指定された制約条件の中で変更されることがあり、パフォーマンスの向上を柔軟に図ることができます。パフォーマンスの向上は望んでいるが、複数のプラットフォームでビットごとの再現可能な結果は必要でないという開発者であれば、数値コードに `StrictMath` ではなく、`Math` を使用することができます。

任意精度算術演算用の J2SE API である BigInteger クラスと BigDecimal クラスでは、桁あふれを起こしたり精度が下がることがない数値演算が可能で、財務関連の計算など、さまざまな計算に不可欠な機能です。BigInteger クラスは、純粋な Java プログラミング言語コードに再実装されています。以前は、BigInteger クラスの実装は下位の C ライブラリをベースにしていました。新しい実装では、大半の標準的な演算が従来の実装より高速に実行できます。また、新しい API は新たに便利な機能が提供されており、使いやすくなっています。

新しい Timer API が Java 2 Platform に追加され、アニメーション、人間の操作に対するタイムアウト、画面上の時計とカレンダー、仕事のスケジューリングルーチン、忘備機能などがサポートされるようになりました。

仮想マシンのシャットダウンフック用の API が、java.lang.Runtime クラスに追加されています。これにより、ネットワーク接続の切断、セッション状態の保存、一時ファイルの削除など、Java プログラミング言語で書かれたアプリケーションがシャットダウンの動作を開始できるように、下位のオペレーティングシステムのプロセスシャットダウン通知機能に対し、簡単で移植性があるインタフェースが提供されています。

Zip ファイルや Jar ファイルを開くときに、新たに「終了時に削除」モードが追加され、長時間稼動しているサーバーアプリケーションは不要となった JarFile オブジェクトやデータを削除し、ディスク容量を解放できるようになりました。

Java 2 SDK Standard Edition バージョン 1.2.2_07a と以前のリリース

次に、J2SE リリースの新しい機能について説明します。

表 18–1 以前の Java 2 Standard Edition (J2SE) リリース

Java リリース	Update リリース
J2SE 1.2.2_07a では、J2SE 1.2.2 シリーズの前回のリリースで見つかったバグが修正されています。J2SE 1.2.2_07a での重要なバグ修正に、J2SE 1.2.2_05 で発生したパフォーマンスの低下に対する修正があります。J2SE 1.2.2_07a におけるバグ修正の詳細は、`http://java.sun.com/j2se/1.2/ReleaseNotes.html` を参照してください。	4/01
J2SE 1.2.2_06 が前回のリリースのバグ修正によって改良されました。	1/01
J2SE v. 1.2.2_05a は、v. 1.2.2_05 (末尾に「a」が無い) のバグ修正版であり、次の新機能および拡張機能が組み込まれています。 ■ スケーラビリティの向上 (20 以上の CPU に対応) プリミティブおよびスレッドの並行処理能力の向上により、マルチスレッド化プログラムの性能が向上し、多数のスレッドを使用するプログラムにおいてガーベッジコレクションによる休止時間が大幅に削減されました。 ■ JIT コンパイラの最適化の向上 JIT コンパイラが実行する新しい最適化機能は、仮想および非仮想方式のインライン化、拡張基本ブロック内の CSE、配列結合の検査を不要にするループ分析、およびより速いタイプチェックです。 ■ テキストの描画性能の向上 Direct Graphics Access (DGA) サポートが組み込まれていない Solaris ソフトウェアプラットフォーム上の Java 2 Standard Edition では、いくつかのグラフィック最適化機能によって、テキストの描画性能が大幅に向上しました。これらのプラットフォームには、Ultra 5、Ultra 10、Solaris (Intel 版) オペレーティング環境、およびすべてのリモートディスプレイシステムが含まれます。 ■ `poller` クラスのデモパッケージ Java アプリケーションから C の `poll(2)` ルーチンの関数に、効率的にアクセスできるようになりました。これは、サンプルサーバーを組み込んだデモパッケージとして提供されます。 ■ Swing の向上 Swing クラスに関して、品質と性能の両面で大幅な改善がなされました。詳細については、次の URL にアクセスしてください。 ■ `http://Java.sun.com/products/jdk/1.2/changes.html` ■ `http://java.sun.com/products/jdk/1.2/fixedbugs/index.html`	10/00

JDK のリリース

Solaris Update リリースには、次の JDK リリースが組み込まれています。

表 **18–2** JDK リリース

JDK リリース	**Update** リリース
JDK 1.1.8_12。前回リリースでのバグ修正によって改良されています。	1/01
JDK 1.1.8_10。バグ修正によって改良されています。	10/00

Apache Web サーバーにおける Java Servlet のサポート

Solaris 8 10/00 リリースで、`mod_jserv` モジュールおよび関連ファイルの追加により、Apache Web サーバーソフトウェアで Java Servlet がサポートされるようになりました。現在、`/etc/apache` に次の構成ファイルが保存されています。

- `zone.properties`
- `jserv.properties`
- `jserv.conf`

`mod_jserv` モジュールは、他の Apache ソフトウェアと同様、オープンなソースコードであり、Sun 以外のグループによって保守されています。このグループは、旧リリースの Apache および `mod_jserv` との互換性の維持に努めています。