HP Operations エージェント

Windows®、Linux、HP-UX、Solaris、AIX オペレーティング システム向け

ソフトウェア バージョン : 11.00

ユーザー ガイド

ドキュメント リリース日付 : 2010 年 10 月
ソフトウェア リリース日付 : 2010 年 10 月

ご注意

保証について

HP 製品およびサービスに関する保証は、これらの製品およびサービスに付随する明示的保証書に記載された内容に限定されます。本文書には、追加の保証を規定している箇所はありません。HP は、本文書に含まれる技術的または編集上の誤りや遺漏に対して、責任を負わないものとします。

この情報は予告なしに変更されることがあります。

法律上の権利の制限について

本書で取り扱っているコンピュータ ソフトウェアは秘密情報であり、その保有、使用、または複製には、HP から使用許諾を得る必要があります。FAR 12.211 および 12.212 に従って、商用コンピュータ ソフトウェア、コンピュータ ソフトウェアのドキュメント、および商用アイテムの技術データは、ベンダの標準商用ライセンスに基づいて米国政府にライセンスが付与されます。

著作権について

商標について

Intel® および Itanium® は、米国およびその他の国における Intel Corporation の商標です。

Microsoft®、Windows®、Windows® XP、および Windows Vista® は、Microsoft Corporation の米国内での登録商標です。

UNIX® は The Open Group の登録商標です。

謝辞

本製品には、Eric Young (eay@cryptsoft.com) 氏によって作成された暗号化ソフトウェアが含まれています。

本製品には、OpenSSL ツールキットで使用するために OpenSSL プロジェクトによって開発されたソフトウェアが含まれています (http://www.openssl.org/)。

本製品には、Tim Hudson (tjh@cryptsoft.com) 氏によって作成されたソフトウェアが含まれています。

この製品には、Apache Software Foundation (http://www.apache.org/) が開発したソフトウェアが含まれています。

この製品には、「zlib」汎用圧縮ライブラリのインターフェイス (Copyright © 1995-2002 Jean-loup Gailly and Mark Adler) が含まれています。

本製品には、Carnegie Mellon University によって作成されたソフトウェア (Copyright 1989, 1991, 1992 Carnegie Mellon University) が含まれています。

本製品には、The Regents of the University of California によって作成されたソフトウェア (Copyright 1996, 1998-2000 The Regents of the University of California) が含まれています。

ドキュメントのアップデートについて

本書のタイトル ページには、以下の識別情報が記載されています。

- ソフトウェアのバージョンを示すソフトウェア バージョン番号。
- ドキュメントが更新されるたびに変更される、ドキュメントのリリース日。
- このバージョンのソフトウェアのリリース日を示す、ソフトウェアのリリース日。

最新の更新を確認する、またはドキュメントの最新エディションを使用しているかどうかを確認するには、以下の URL にアクセスしてください。

http://h20230.www2.hp.com/selfsolve/manuals

このサイトでは、HP Passport への登録とサイン インが必要となります。HP Passport ID を登録するには、次の URL にアクセスしてください。

http://h20229.www2.hp.com/passport-registration.html

または、HP Passport のログイン ページで [**New users - please register**] リンクをクリックします。

適切な製品サポート サービスに登録することで、本書の最新版を受け取ることもできます。詳細は、正規販売代理店にお問い合わせください。

サポート

以下の HP ソフトウェア サポート オンライン Web サイトにアクセスしてください。

www.hp.com/go/hpsoftwaresupport

この Web サイトには HP Software の製品、サービス、サポートに関する詳細情報と連絡先が示してあります。

HP ソフトウェア サポート オンライン Web サイトでは、セルフソルブ技術情報を提供しています。ここでは、ビジネスの管理に必要なインタラクティブ テクニカル サポート ツールに迅速かつ効率的にアクセスできます。サポート契約されているお客様がサポート サイトで実行できることは以下のとおりです。

- 関心のある情報について技術情報ドキュメントを検索する
- サポート ケースと改善依頼を記録 / 追跡する
- ソフトウェア パッチをダウンロードする
- サポート契約を管理する
- HP のサポート契約を参照する
- 利用できるサービスに関する情報を参照する
- 他のソフトウェア利用者と意見を交換する
- ソフトウェア トレーニングの情報を調べ、登録する

ほとんどのサポート領域では、HP Passport ユーザーとして登録し、サイン インする必要があります。また、多くの場合、サポート契約が必要となります。HP Passport ユーザー ID を登録するには、次の URL にアクセスしてください。

http://h20229.www2.hp.com/passport-registration.html

サポートのアクセス レベルの詳細を確認するには、以下の URL にアクセスしてください。

http://h20230.www2.hp.com/new_access_levels.jsp

目次

1 はじめに 11
ドキュメント マップ 12
関連ドキュメント 13

2 データ収集の管理 15
logglob 16
logappl 16
logproc 16
logpcmd 16
logdev 17
logtran 17
logls 17
logindx 17
parm ファイルの変更 18
コアに基づいた正規化によって計算されたメトリックの記録 38
データ収集の終了 39
データ収集の再開始 40
夏時間 40
システムの時刻の手動変更 40
ログ ファイルにより使用されるディスク スペースの制御 41
データのアーカイブ 43

3 HP Operations エージェント の操作 45
監視対象オブジェクトの永続化 47
Windows でのデフォルト ユーザーの変更 51
UNIX/Linux でのデフォルト ユーザーの変更 53

4 utility プログラムの使い方 55
対話型モードとバッチ モードの使用例 56
コマンド ライン インターフェイスの使用例 59
初期値 60
初期 parm ファイルのアプリケーション定義 61
年代順の詳細 61
要約 63

5 utility のコマンド 67

6 extract プログラムの使い方 89
構文 90
データのエクスポート方法 96

エクスポート タスクのサンプル....97
データ ファイルのエクスポート....98
エクスポート テンプレート ファイルの構文....99
カスタム グラフまたはレポートの作成....102
エクスポート ファイルの出力....103
ASCII フォーマットと datafile フォーマットに関する注意....104
binary フォーマットに関する注意....105

7 extract のコマンド....111
weekdays....142
weekly....143
yearly....145

8 cpsh プログラムの使用....147
リアルタイム メトリックの表示....148
メトリック クラスの変更....148
メトリックのヘルプの表示....149
要約メトリック データの表示....149

9 パフォーマンス アラーム....151
アラーム ジェネレータ....152
Network Node Manager への SNMP トラップの送信....152
Operations Manager へのメッセージの送信....152
ローカル アクションの実行....153
アラーム処理のエラー....153
アラームの履歴データの分析....154
アラーム定義の構成要素....156
アラーム構文のリファレンス....157
表記について....157
共通要素....157
ALARM 文....160
ALERT 文....165
EXEC 文....166
PRINT 文....167
IF 文....168
LOOP 文....169
INCLUDE 文....170
USE 文....171
VAR 文....172
ALIAS 文....173
SYMPTOM 文....174
アラーム定義の例....176
アラーム定義のカスタマイズ....178

10 RTMA コンポーネントのアドバイザ....179
複数のシステム上での adviser スクリプトの実行....180
構文表記....181

コメント 181
条件 182
定数 182
式 182
アドバイザ構文内のメトリック名 183
出力リスト 184
変数 184
アドバイザ構文 185

11 Windows での Performance Collection Component の使用 211
データ型とクラス 212
ログ ファイル データの抽出 214
ファイル属性 216
エクスポート ファイルのテンプレート 218
デフォルトのエクスポート ファイル 219
ログ ファイル データのアーカイブ 225
アーカイブ期間 225
アーカイブ データの追加 225
アーカイブに関するヒント 226
分析するデータの範囲 227
分析レポート 227
データ ソース ファイルのフォーマット 237
パフォーマンス カウンタ収集の構築 244
パフォーマンス カウンタ収集の管理 244
コマンド ラインからの ECBM の管理 245

12 データ ソース統合の概要 247
クラス仕様の作成 248
データの収集と記録 248
データの使用 249

13 データ ソース統合の使用方法 251
ログ ファイル フォーマットの定義 252
ログ ファイルの編成方法 252
ログ ファイル セットの作成 254
クラスの仕様ファイルとログ処理のテスト (オプション) 254
ログ ファイル セットへのデータの記録 255
ログ データの使用 256

14 DSI クラスの仕様 257
クラスの仕様構文 258
クラスの記述 259
クラス 259
ラベル 260
INDEX BY, MAX INDEXES, ROLL BY 260
ログ ファイル サイズの制御 265
RECORDS PER HOUR 267

CAPACITY . . . 268
メトリックの記述 . . . 269
METRICS . . . 269
ラベル . . . 270
要約方法 . . . 271
PRECISION . . . 271
TYPE TEXT LENGTH . . . 272
クラス仕様のサンプル . . . 274
15 DSI プログラムに関する参考資料 . . . 277
sdlcomp コンパイラ . . . 278
コンパイラ構文 . . . 278
コンパイラ出力のサンプル . . . 279
構成ファイル . . . 282
DSI メトリックのアラーム定義 . . . 282
アラーム処理 . . . 282
dsilog によるデータ処理方法 . . . 286
sdlgendata によるログ処理のテスト . . . 286
フォーマット ファイルの作成 . . . 289
クラス仕様の変更 . . . 291
DSI データのエクスポート . . . 292
DSI ログファイル データをエクスポートする extract プログラムの使用例 . . . 292
Performance Manager でのデータの表示 . . . 292
sdlutil によるデータの管理 . . . 293
構文 . . . 293
16 データ ソースの統合例 . . . 295
dsilog スクリプトの記述 . . . 296
vmstat データの記録 . . . 297
クラス仕様のファイルの作成 . . . 297
クラス仕様ファイルのコンパイル . . . 297
dsilog によるログ処理の開始 . . . 298
データへのアクセス . . . 298
クラス仕様ファイルの作成 . . . 300
クラス仕様ファイルのコンパイル . . . 301
DSI ログ処理の開始 . . . 302
複数ファイルの sar データの記録 . . . 303
複数のクラスの仕様ファイルの作成 . . . 303
複数のクラスの仕様ファイルのコンパイル . . . 307
DSI ログ処理の開始 . . . 308
複数オプションの sar データの記録 . . . 309
システム ユーザー数の記録 . . . 315
17 エラー メッセージ . . . 317
一般エラー メッセージ . . . 332

18 トランザクション追跡の紹介 335
トランザクション追跡の利点 336
ユーザーの視点から見たトランザクション時間 336
トランザクション データ 336
サービス レベルの目標 337
リアルタイム注文処理の必要事項 337
注文処理アプリケーションの準備 338
ARM の使用のガイドライン 339
19 トランザクション追跡の動作 341
ARM 2.0 のサポート 342
arm_complete_transaction コール 343
ARM 組み込みアプリケーションの例 343
アプリケーション名とトランザクション名の指定 344
トランザクション追跡デーモン (ttd) 345
ARM API コール ステータス リターン 345
トランザクション構成ファイル (ttd.conf) 347
新しいアプリケーションの追加 347
新しいトランザクションの追加 347
range 値または slo 値の変更 347
構成ファイルのキーワード 348
構成ファイルのフォーマット 349
構成ファイル例 350
ガイドライン 351
ディスク I/O オーバーヘッド 352
CPU オーバーヘッド 352
メモリー オーバーヘッド 353
20 トランザクションについて 355
開始する前に 355
サービス レベルの目標 (SLO) の定義 356
parm ファイルの変更 357
トランザクション データの収集 357
構成ファイルのカスタマイズ (オプション) 358
アラーム 361
21 トランザクション追跡のメッセージ 363
22 トランザクション メトリック 365
23 トランザクション追跡の例 367
例 1 (注文処理の擬似コード例) 369
例 2 369
例 3 369
例 4 370

24 高度な機能 371
25 トランザクション ライブラリ 375
プラットフォームごとの C コンパイル オプション例 381
ARM NOP ライブラリ 383
例 383
アプリケーションの設定 (arm_init) 383
UDM を使用したトランザクションの設定 384
メトリックの追加 384
相関係数を使用したトランザクション インスタンスの起動 386
相関係数を使用しないトランザクション インスタンスの起動 387
UDM を使用したトランザクション インスタンス データの更新 387
UDM を使用しないトランザクション インスタンス データの更新 387
メトリックを更新するトランザクション インスタンスの停止 388
メトリックを更新しないトランザクション インスタンスの停止 388
UDM を使用した完全なトランザクションの使用 389
UDM 使用しない完全なトランザクションの使用 389
26 記録とトレース 391
記録ポリシーの設定 392
アプリケーションの特定 392
トレース タイプの設定 394
トレース設定ファイルの概要 395
設定ファイルの作成 398
トレース機構の有効化 404
トレース メッセージの表示 405
トレースのフィルタリング 409
27 トラブルシューティング 411
索引 415

1 はじめに

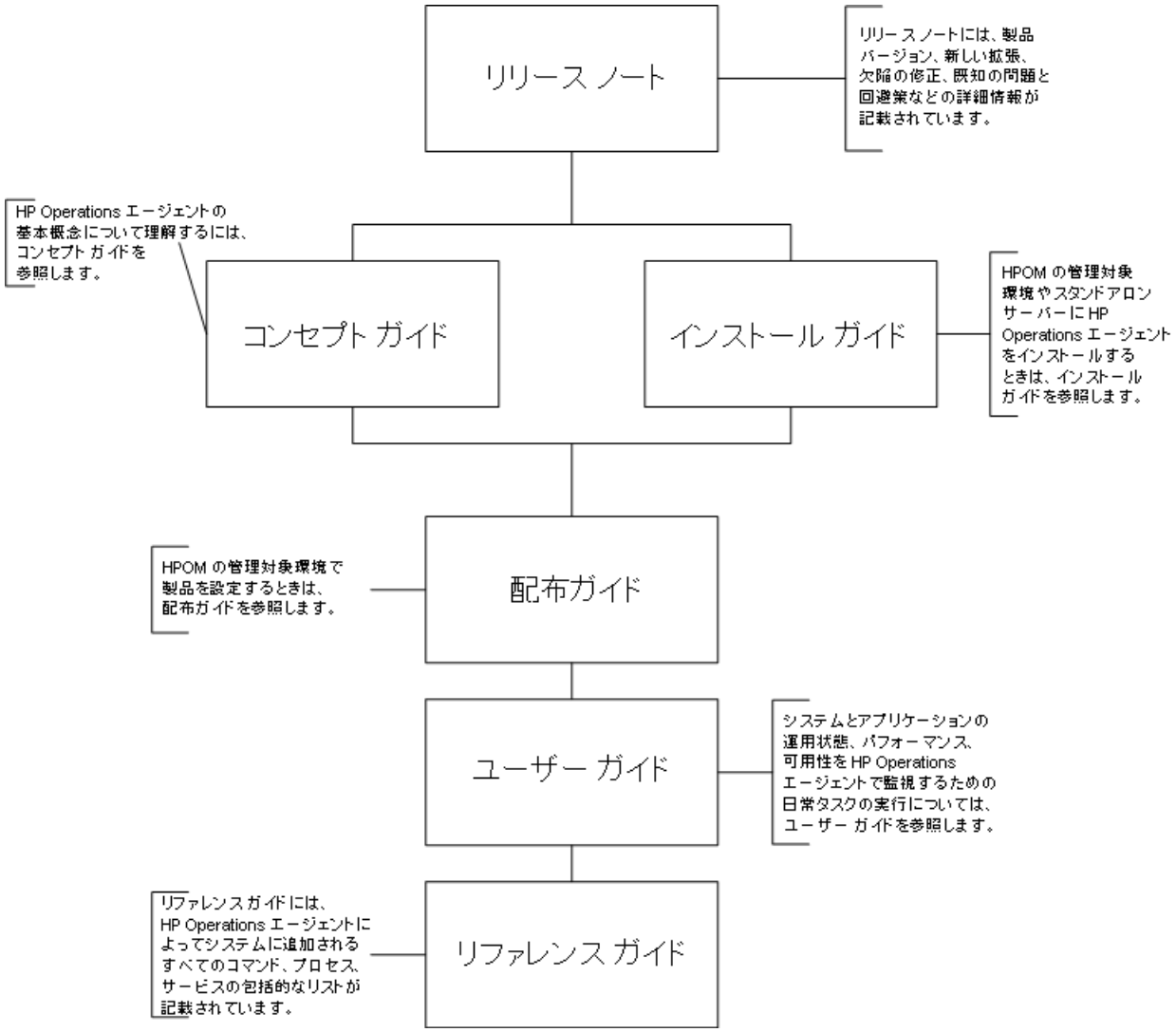
HP Operations エージェントは、システムの重要要素の運用状況、パフォーマンス、可用性を表すメトリックを収集し、システムを監視するユーザーに提供します。

HP Operations エージェントはログ ファイルを使用して、収集されたメトリックを格納し、収集されたメトリックが事前設定されたしきい値に一致しない場合に、アラート メッセージを生成する機構を提供します。収集されたメトリック データは、HP Reporter や HP Performance Manager などのデータ分析ツールで表示できます。HP Operations Manager (HPOM) 管理サーバーで動作するようにエージェントをインストールして設定すると、HPOM コンソールから一元的にエージェント ノードのヘルスおよびパフォーマンスを監視できます。

ドキュメント マップ

ドキュメント マップは、HP Operations エージェント に関する主なドキュメントをすべて一覧しています。このマップを使用すれば、支援が必要なときに必要なドキュメントを特定することができます。

図 1 HP Operations エージェントのドキュメント マップ



関連ドキュメント

HP Operations エージェントに関するすべてのユーザー向けドキュメントは、製品メディアの paperdocs ディレクトリに存在します。最新の更新を確認する、またはドキュメントの最新エディションを使用しているかどうかを確認するには、以下の URL にアクセスしてください。

http://h20230.www2.hp.com/selfsolve/manuals

このサイトでは、HP Passport への登録とサイン インが必要となります。HP Passport ID を登録するには、次の URL にアクセスしてください。

http://h20229.www2.hp.com/passport-registration.html

または、HP Passport ログイン ページの [**New users - please register**] リンクをクリックしてください。

表 1　　HP Operations エージェントのユーザー ドキュメント

ドキュメント	用途	主なトピック
リリース ノート	製品バージョン、新機能、既知の問題に関する情報については、このドキュメントを参照します。	• 新機能 • 機能強化 • 修正 • 既知の問題と制限
コンセプト ガイド	異なる環境で HP Operations エージェントがどのように機能するかを理解するには、コンセプト ガイドが役立ちます。	• HP Operations エージェントの概要 • HP Operations エージェントの主要コンポーネント

表 1　　HP Operations エージェントのユーザー ドキュメント

ドキュメント	用途	主なトピック
インストール ガイド	インストール ガイドを参照することで、HP Operations エージェントを以下の環境にインストールできます。 • HPOM 管理サーバー (HPOM の管理対象となる分散管理環境で使用する場合) • スタンドアロン サーバー (HP Performance Manager などの外部のデータ分析ツールで使用する、ローカル サーバーのシステム パフォーマンス メトリックを収集する場合)	• HPOM コンソールからの HP Operations エージェントのインストール • HP Operations エージェントの手動インストール • ライセンシング
配布ガイド	一元的な HPOM 管理サーバーから監視対象環境に HP Operations エージェントを配布するときは、このガイドを参照します。	• HPOM 管理サーバーと HP Operations エージェントの間の安全な通信チャネルのセットアップ • 高可用性クラスタ環境で動作する HP Operations エージェントの設定 • HP Operations エージェントの設定の HPOM コンソールからのリモート管理 • 他の HP ソフトウェア製品との統合
リファレンス ガイド	リファレンス ガイドには、HP Operations エージェント ノードで使用できるすべてのコマンド、プロセス、サービスの完全なリストが記載されています。	• コマンド ライン ユーティリティ • 設定変数

2　データ収集の管理

HP Operations エージェント には、監視対象システムのシステム パフォーマンス データを収集し記録するデータ コレクタが備わっています。データ収集プログラムの **scope** を使用することで、収集したデータをシステム上に格納できます。HP Performance Manager または HP Reporter を使用すれば、格納したデータの表示、分析が可能です。

scope コレクタでは、システム上で以下のタスクを実行できます。

- 監視対象システムのヘルスおよびパフォーマンスを示すメトリック データの収集
- 収集したメトリック データを異なるログ ファイルに記録

scope は NFS データを記録しませんが、ローカル ファイル システム上の HP GlancePlus を介することで NFS データを表示できます。

構成パラメータ ファイルの parm ファイルを使用すると、scope コレクタのデフォルトのデータ ログ作成機構を設定できます。parm ファイルのパラメータを変更することで、scope コレクタの以下のプロパティを制御できます。

- データ ログ作成間隔
- データ型
- ログ ファイルのサイズ

ノードに HP Operations エージェントをインストールしたら、parm ファイルを変更して scope のデータ収集機構を設定する必要があります。

ログ ファイルの収集

scope データ コレクタ (UNIX および Linux ノードでは scopeux、Windows ノードでは scopent) は、システム リソース利用率のパフォーマンス測定値を収集して要約し、parm ファイルの log line で指定されたデータ クラスに応じて、そのデータを次のログ ファイルに記録します。

- logglob
- logappl
- logproc
- logpcmd
- logdev
- logtran
- logls

- logindx

ログ ファイルのレコードのタイムスタンプは、データ収集の開始時間を示します。対象プロセスの概念はフィルタリングです。このフィルタリングは、ログに記録され、parm ファイルから制御されるデータの量を最小化するために役立ちます。

scope は NFS データを記録しませんが、ローカル ファイル システム上の GlancePlus を介することで NFS データを表示できます。

logglob

logglob ファイルには、システム全体 (グローバル) のリソース利用率情報の測定値が記録されます。scope コレクタはグローバル データを要約し、parm ファイルに指定された間隔で、データを定期的に記録します。

logappl

logappl ファイルには、parm ファイルに定義されたアプリケーションで実行されるプロセスの測定値の総計が記録されます。scope コレクタはアプリケーション データを要約し、parm ファイルに指定された間隔で、データを定期的に記録します。

logproc

scope コレクタは対象と考えられるプロセスを特定し、特定されたプロセスの測定値の総計を logproc ファイルに記録します。scope は、以下の条件を基にプロセスを特定します。

- プロセスの先頭
- プロセスの末尾
- parm ファイルで指定する構成の詳細

logpcmd

logpcmd ファイルには、logproc ファイルに記録されるプロセスで実行される、コマンドライン アクティビティの詳細が記述されています。

logpcmd ファイルのサイズ、ロールオーバー、記録間隔の制御はできません。

logpcmd ファイルは、ノードの以下のディレクトリに格納されます。

- Windows の場合 : *%ovdatadir%*\datafiles
- UNIX/Linux の場合 : /var/opt/perf/datafiles

ファイルに格納できるデータは最大 25 メガバイトです。データ収集機構が開始されると、scope コレクタが logpcmd ファイルの最初のインスタンスを作成します。このとき、末尾に 0 が付与されます。logpcmd0 ファイルが最大サイズの 25 メガバイトに達すると、scope は logpcmd ファイルの 2 つめのインスタンスである、logpcmd1 ファイルを作成します。

logpcmd1 ファイルが最大サイズの 25 メガバイトを超えると、logpcmd0 ファイルからデータのロールオーバーが開始されます。

logdev

logdev ファイルには、個々のデバイス パフォーマンスの測定値が記録されます。scope コレクタはデバイス データを要約し、parm ファイルに指定された間隔で、データを定期的に記録します。

logtran

logtran ファイルには、ARM トランザクション データの測定値が記録されます。scope コレクタはトランザクション データを要約し、parm ファイルに指定された間隔で、データを定期的に記録します。トランザクション データの収集に関する詳細は、335 ページの「トランザクション追跡の紹介」を参照してください。

logls

logls ファイルには、論理システムに関する情報が記録されます。scope コレクタは論理システム データを要約し、parm ファイルに指定された間隔で、データを定期的に記録します。

logls ファイルは、HPVM、Hyper-V ホスト、vSphere Management Assistant (vMA)、Solaris グローバル ゾーン、AIX-LPAR にある HP Operations エージェントでのみ使用可能です。

logindx

logindx には、別のログ ファイルのデータにアクセスするために必要な情報が記録されます。

scope のステータス

scope の起動時には、このログ ファイルの他に 2 つのファイルが作成されます。1 つは /var/opt/perf/datafiles/ ディレクトリにある RUN ファイルで、もう 1 つは /var/opt/perf/ ディレクトリにある status.scope ファイルです。

RUN ファイルは、scope プロセスが実行中であることを示すために作成されます。このファイルを削除すると、scope が終了します。

/var/opt/perf/status.scope ファイルは、scope プロセスのステータス ログまたはエラー ログとして使用できます。このファイルには、scope コレクタの開始時と終了時、または警告やエラーの発生時に新しい情報が追加されます。scope の最新のステータスおよびエラー情報を表示するには、perfstat -t コマンドを使用します。

parm ファイル

parm ファイルは、scope に特定のパフォーマンス測定値を記録させる命令を含んだテキストファイルです。

新規インストールの際には、HP Operations エージェント によってデフォルトの parm ファイルが 2 つの異なるディレクトリに作成されます。

- Windows:

  Windows では、parm ファイルの拡張子は .mwc (parm.mwc) となります。

  — *%ovinstalldir%*\newconfig

  — *%ovdatadir%*

- HP-UX、Solaris、Linux:

  – /opt/perf/newconfig

  – /var/opt/perf

- AIX の場合 :

  – /usr/lpp/perf/newconfig

  – /var/opt/perf

scope のデータ収集機構は、*%ovdatadir%* (Windows 版) または /var/opt/perf (UNIX、Linux 版) ディレクトリにある parm ファイルの設定により制御します。

デフォルトのデータ収集機構を変更する場合は、*%ovdatadir%* (Windows 版) または /var/opt/perf (UNIX、Linux 版) ディレクトリにある parm ファイルの設定を変更します。

ノード上の HP Operations エージェントを旧バージョンの HP Performance Agent からアップグレードする場合は、newconfig ディレクトリの parm ファイルのコピーが、アップグレードプロセスによって更新されます。他のディレクトリにある parm ファイルは影響を受けず、引き続きノード上のデータ収集機構を管理します。つまり、この方法を用いることで、製品をアップグレードした後であっても、構成済みのデータ収集機構を保持できます。製品をアップグレードした後であれば、既存の parm ファイルの設定内容と、newconfig ディレクトリにある新しいバージョンの parm ファイルとを比較し、必要な変更を加えることができます。

parm ファイルは、平均的な量のログ ファイル データを収集するように設定されています。データの最大量は、システムによって異なります。また、22 ページの「パラメータの説明」の、size パラメータの説明も参照してください。

parm ファイルの変更

parm ファイルは、ASCII フォーマットでファイルを保存できるワード プロセッサやエディタを使用して変更できます。

parm ファイルを変更または新規作成するときは、次の規則と表記規則が適用されます。

- パラメータを指定すると、デフォルト値が無効になります。デフォルト値については、newconfig ディレクトリにある parm ファイルを参照してください。
- parm ファイルにパラメータを指定する順序は、重要な意味を持ちません。

- パラメータを複数回指定した場合、最後のパラメータのインスタンスが有効となります。
- application 文を定義する file、user、group、cmd、argv1、or の各パラメータは、その application 文の後に指定する必要があります。
- アプリケーションのパラメータは、プロセスが最初に適合するアプリケーションに総計されるように、順にリストで表示します。
- すべてのコマンドおよびパラメータ文には、大文字、小文字、または大文字と小文字の組み合わせを使用できます。
- 空白またはコンマを使用して、各文に含まれるキーワードを区切ります。
- parm ファイルのパラメータにはコメントを書き込むことができます。コメント コード (/*) またはポンド記号 (#) で始まる行は無視されます。

parm ファイルの変更後は、変更を有効にするために Performance Collection Component コンポーネントを再開始する必要があります。Performance Collection Component を再起動するには、次のコマンドを実行します。

Windows の場合

%ovinstalldir%bin\ovpacmd REFRESH COL

HP-UX、Linux、Solaris の場合

/opt/perf/bin/ovpa -restart scope

AIX の場合

/usr/lpp/perf/bin/ovpa -restart scope

リアルタイム メトリック アクセス (RTMA) コンポーネントを使用する場合は、perfd プロセスを再起動します。

Windows の場合

%ovinstalldir%bin\ovpacmd REFRESH RTMA

HP-UX、Linux、Solaris の場合

/opt/perf/bin/pctl restart

AIX の場合

/usr/lpp/perf/bin/pctl restart

parm ファイルのパラメータ

scope は、収集パラメータ (parm) ファイル内の特定のパラメータによって制御されます。これらのパラメータは、次の定義と指定を行います。

- scope 生ログ ファイルに対する最大ディスク スペース量の設定
- 記録するデータの種類の指定
- データを記録する間隔の指定
- 記録するプロセスおよびメトリックの属性の指定
- 収集対象および記録対象のパフォーマンス データの種類の定義
- ユーザーが定義可能な監視対象アプリケーションの指定。アプリケーションは、グループとして監視される 1 つまたは複数のプログラムになります。

- scope が毎日のログファイルの保守アクティビティを実行する時間の指定。この指定は、システムの可用性に影響を与えないために行われます。

これらのパラメータを変更すると、監視対象システムの要件を満たすパフォーマンス データを scope で記録できます (18 ページの「parm ファイルの変更」を参照してください)。

20 ページの表 2 の parm ファイル パラメータが scope で使用されます。これらのパラメータの中には、特定のシステム専用のものもあります (表を参照)。これらのパラメータの詳細は、22 ページの「パラメータの説明」および 31 ページの「アプリケーション定義パラメータ」を参照してください。

表 2　scope が使用する parm ファイル パラメータ

パラメータ	値またはオプション
id	system ID
log	• global • application [=prm] [=all] ([=prm] HP-UX のみ) • process • device=disk,lvm,cpu,filesystem,all (lvm HP-UX のみ) • transaction=correlator,resource (resource HP-UX のみ) • logicalsystem (Solaris では、論理システムは Solaris 10 以降のオペレーティング環境に対応) AIX では、論理システムは AIX 5L V5.3 ML3 以降の LPAR、AIX 6.1 TL2 のグローバル環境の WPAR にのみ対応しています。 lpar ロギングを有効にする場合 logicalsystems=lpar logicalsystems wpar ロギングを有効にする場合 logicalsystems=wpar lpar ロギングおよび wpar ロギングを有効にする場合 logicalsystems=lpar,wpar logicalsystems=wpar,lpar logicalsystems=all
mainttime	hh:mm (24 時間制)
scopetransactions	on off
subprocinterval	値は秒単位 (HP-UX は対象外)
javaarg **注記 :** UNIX/Linux のみ。	true false

表 2　scope が使用する parm ファイル パラメータ (続き)

パラメータ	値またはオプション
procthreshold (threshold と同じ)	cpu=*percent* disk=*rate* (Linux および Windows は対象外) memory=nn (値はメガバイト単位) nonew nokilled shortlived
appthreshold	cpu=*percent*
diskthreshold	util=*rate*
bynetifthreshold	iorate=*rate*
fsthreshold	util=*rate*
lvthreshold	iorate=*rate*
bycputhreshold	cpu=*percent*
wait	cpu=*percent* (HP-UX のみ) disk=*percent* (HP-UX のみ) mem=*percent* (HP-UX のみ) sem=*percent* (HP-UX のみ) lan=*percent* (HP-UX のみ)
application	*アプリケーション名*
file	*file name [, ...]*
argv1	最初のコマンドの引数 [,]
cmd	コマンド ラインの正規表現
user	*user login name [,]*
group	*groupname [,]*
or	
priority	下位値 - 上位値 (範囲はプラットフォームによって異なる)
size	(値はメガバイト単位) process=*nn* (最大値は 4096) 下位のクラスの最大値は 2048 です。 global=*nn* application=*nn* device=*nn* transaction=*nn* logicalsystem=*nn*

表 2　scope が使用する parm ファイル パラメータ (続き)

パラメータ	値またはオプション
days	global=*nn* (値は単位) application=*nn* process=*nn* device=*nn* transaction=*nn* logicalsystem=*nn*
maintweekday	Sun|Mon|Tue|Wed|Thu|Fri|Sat
collectioninterval	process=ss (値は秒単位) global=ss
gapapp	blank unassignedprocesses existingapplicationname other
flush	ss (値は秒単位) 0 (データ　フラッシュを無効化)
zone_app	true false (Solaris 10 以降のみ)
proccmd 注記 : UNIX/Linux のみ	0 (プロセス　コマンドのロギングを無効化) nnnn (プロセス　コマンドの長さの数値を参照。最大値は 1024)
ignore_mt	true (グローバル　クラスの CPU メトリックは、システム内のコアの有効数に対して正規化した値を報告) false (グローバル　クラスの CPU メトリックは、システム内の CPU スレッドの有効数に対して正規化した値を報告) ineffective (マルチスレッドを無効化)

パラメータの説明

ここでは、parm ファイルの各パラメータについて説明します。

- ID
- Log
- Thresholds
- scopetransactions
- subprocinterval
- gapapp
- wait
- size
- mainttime

- days
- maintweekday
- javaarg
- flush
- zone_app
- proccmd
- ignore_mt

ID

システム ID 値は、システムを識別する文字列です。割り当てられるデフォルト ID はシステムのホスト名です。割り当てられるデフォルト ID を変更する場合は、すべてのシステムが一意の ID 文字列を持つことを確認してください。この識別子は、データを収集したシステムを示すためにログ ファイルに収められます。最大 39 文字を指定できます。

Log

log パラメータは、scope で収集するデータ型を指定します。

- log global を指定すると scope で logglob ファイルにグローバル レコードを記録できます。システムのパフォーマンス データを表示し、分析するには、グローバル データ レコードが必要です。グローバル メトリックは、アプリケーション データおよびプロセス データのログ オプションや値による影響は受けません。
- log application を指定すると scope で logappl ファイルにアクティブなアプリケーションのレコードを記録できます。デフォルトでは、インターバル中にアクティブなプロセスがあったアプリケーションのみが記録されます。
  - parm ファイルで log application=all を指定すると、アプリケーションがアクティブかどうかに関係なく、すべてのアプリケーションが scope によってインターバルごとに logappl ファイルに記録されます。

    application=all オプションは、アプリケーション アラームの使用と関連がある特定の状況に適しています。たとえば、アプリケーションがアクティブでなくなった場合にアラームを発生させることができます (APP_ALIVE_PROC)。

    このオプションを有効にすると、すべてのアプリケーションがインターバルごとに記録されるため、logappl のサイズがより短時間で大きくなります。utility プログラムの scan 関数を使用すると、scope ログ ファイルの利用率を監視できます。
  - log application=prm パラメータを指定すると、scope により、アクティブな Process Resource Manager (PRM) グループを logappl ファイルに書き込むことができます (HP-UX のみ)。このパラメータを指定すると、scope は parm ファイルにリストされているユーザー定義のアプリケーション セットを記録しません。また、収集されたすべてのアプリケーション メトリックは、PRM コンテキストを反映し、APP_NAME_PRM_GROUPNAME メトリックによってグループ化されます。

    アプリケーションのログ オプションは、グローバル データおよびプロセス データには影響しません。

- `log process` を指定すると scope で `logproc` ファイルに対象プロセスに関する情報を記録できます。ログの対象となるのは、プロセスが最初に作成されたとき、終了したとき、および `parm` ファイルで定義されたアプリケーションのしきい値を超えたときです。プロセスのしきい値のログ オプションは、グローバル データおよびアプリケーション データには影響しません。
- `log device=disk,lvm, cpu, filesystem` を指定すると、scope によって、各ディスク、論理ボリューム (HP-UX のみ)、CPU、ファイル システムに関する情報が `logdev` ファイルに記録されます。

  監視対象のシステムが HP-UX オペレーティング システムで実行されていない場合は、`lvm` を使用しないでください。

  デフォルトでは、インターバル中に I/O が発生したディスク、ボリューム、およびインターフェイスのみが記録されます。`netif` (論理 LAN デバイス) レコードおよびディスク レコード (HP-UX) は、選択したログ デバイス オプションに関係なく、常に記録されます。

  たとえば、各ディスク、論理ボリューム、CPU、ネットワーク インターフェイスのレコードは記録するが、各ファイル システムは記録しない場合、以下の設定を使用します。

  ```
  log device=disk,lvm,cpu.
  ```

  — `filesystem` を指定すると、マウントされているすべてのローカル ファイル システムが、アクティビティに関係なくすべてのインターバルごとに記録されます。

  — `parm` ファイルで `log device=all` を指定すると、scope によって、すべてのディスク、論理ボリューム、CPU、ネットワーク インターフェイスが、それらのデバイスがアクティブかどうかに関係なく、インターバルごとに `logdev` ファイルに記録されます。

  このオプションを有効にすると、すべてのデバイスがインターバルごとに記録されるため、`logdev` のサイズがより短時間で大きくなります。ログ ファイルの利用率とサイズをモニタするには、`utility` プログラムの `scan` 関数を使用します。
- `log transaction` を指定すると scope で `logtran` ファイルに ARM トランザクション レコードを記録できます。scope でデータを収集するには、アプリケーション応答測定 (ARM: Application Response Measurement) API が組み込まれているプロセスを実行する必要があります (詳細については、335 ページの「トランザクション追跡の紹介」を参照してください)。

  `log transaction` パラメータのデフォルト値は、`no resource` と `no correlator` です。

  リソース データ (HP-UX のみ) または相関データを収集するには、**`log transaction=resource`** または **`log transaction=correlator`** を指定します。両方のデータを記録するには、**`log transaction=resource, correlator`** を指定します。
- `log logicalsystems` を指定すると scope で `logls` ファイルに論理システムに関する情報を記録できます。論理システムに関するデータは、`parm` ファイルで指定した間隔で定期的に要約されます。

  AIX6.1 TL2 での LPAR および WPAR の BYLS ロギングは、`parm` ファイルの `logicalsystems` パラメータを使用して設定します。20 ページの表 2 を参照してください。

ログ オプションに関係なく、自動的にログ ファイルが作成されます。特定の種類の記録が無効になっていても、対応するログ ファイルが監視対象システムから自動的に削除されることはありません。

オプションなしで `log` を指定すると、scope はグローバル データとプロセス データのみを記録します。

Thresholds

しきい値パラメータを指定すると、scope は、システムに関する不必要で、重要度の低い詳細情報をフィルタリングし、重要な情報のみを記録します。

それぞれのメトリックのクラスのしきい値の指定には、以下のパラメータを使用します。データのクラスの特定のインスタンスに関して、指定したしきい値を超えた場合は、scope によってそのインスタンスに関するレコードが記録されます。

しきい値に小さな値を指定すると、scope でより多くのデータが記録され、逆にしきい値に大きな値を指定すると、scope で記録されるデータが少なくなるため、記録されるレコードの数が概ね少なくなります。使用可能なしきい値のパラメータのリストを次に示します。

- Procthreshold
- appthreshold
- diskthreshold
- bynetifthreshold
- fsthreshold
- lvthreshold
- bycputhreshold

Procthreshold

`procthreshold` パラメータは、対象プロセスの条件を指定するアクティビティ レベルを設定するために使用します。このパラメータを使用するには、プロセスの記録を有効にする必要があります。`procthreshold` は、記録されるプロセスのみに影響するもので、データの他のクラスには影響しません。

しきい値のオプションは、(コンマで区切って) 同じパラメータ行に指定します。

プロセス データ用の procthreshold オプション

`cpu` CPU の利用率を設定します。この利用率を超えたプロセスが対象プロセスとなり、記録されます。

パーセント値は、全体的な CPU の使用率を示す実数です。たとえば、`cpu=7.5` は、1 分間のサンプルで CPU の利用率が 7.5% を超えたプロセスが記録されることを示します。

`disk` (Linux および Windows では使用できません) 物理的なディスク I/O の 1 秒あたりの回数を設定します。この回数を超えたプロセスが対象プロセスとなり、記録されます。

この値は実数です。たとえば、disk=8.0 を指定すると、1 秒当りの平均物理ディスク I/O 率が 8 KB を超えたプロセスが記録されます。

`memory` メモリのしきい値を設定します。このしきい値を超えたプロセスが対象プロセスとなり、記録されます。

値はメガバイト単位で、100 KB の精度で最も近い値を指定します。メモリのしきい値が設定されている場合は、`PROC_MEM_VIRT` メトリックの値と比較されます。メモリのしきい値を超えた各プロセスは、ディスクおよび CPU のプロセス ログしきい値の場合と同様に記録されます。

nonew　新しいプロセスがしきい値を超えなかった場合に、そのプロセスの記録を停止します。指定されていない場合は、新しいプロセスはすべて記録されます。HP-UX では、shortlived が指定されていない場合は、1 秒以上の新しいプロセスのみが記録されます。

nokilled　終了したプロセスがしきい値を超えなかった場合に、そのプロセスの記録を停止します。指定されていない場合は、すべての Kill (終了) されたプロセスが記録されます。HP-UX では、shortlived が指定されていない場合は、1 秒以上の Kill されたプロセスのみが記録されます。

shortlived　インターバル内での実行時間が 1 秒未満のプロセスが記録されます (多くの場合、記録されるプロセス数が大幅に増加します)。scope によって parm ファイル内で threshold shortlived が検出されると、nonew および nokilled オプションが削除されている場合は、cpu または disk threshold の指定に関係なく、shortlived に該当するプロセスが記録されます。デフォルトでは、shortlived プロセスは記録されません (shortlived プロセスを記録する必要がない場合は、threshold パラメータで shortlived を指定しないようにしてください)。

process　procthreshold は、PROCESS クラスのしきい値を指定します。デフォルト値は以下のようになります。

- 最後のインターバル内の、CPU のプロセッサ使用率が 10% を超えているプロセス
- 900 MB を超える仮想メモリが設定されているプロセス
- 1 秒当りの平均物理ディスク I/O 率が 5 KB を超えるプロセス

appthreshold

appthreshold パラメータを使用すると、APPLICATION データ クラス (APP_CPU_TOTAL_UTIL メトリック) のしきい値を指定できます。このしきい値基準は、CPU の使用率に基づきます。この使用率を超えたアプリケーションがログ ファイルに記録されます。

parm ファイルのデフォルトの設定を使用すると、scope は CPU 使用率が 0% を超えたアプリケーションを記録します。

diskthreshold

diskthreshold パラメータを使用すると、DISK クラスのしきい値を指定できます。DISK クラスのしきい値基準は、所要時間、つまりディスクの I/O (BYDSK_UTIL メトリック) の割合に基づきます。

parm ファイルのデフォルトの設定を使用すると、scope は、I/O の実行時間が所要時間の 10% を超えているディスクの詳細を記録します。

bynetifthreshold

bynetifthreshold パラメータは、NETIF クラスのしきい値を指定します。Netif データ クラスのしきい値基準は、ネットワーク インターフェイスによる 1 秒当りのパケット転送数 (BYNETIF_PACKET_RATE メトリック) に基づきます。

parm ファイルのデフォルトの設定を使用すると、scope は 1 秒当りに 60 を超えるパケットを転送したネットワーク インターフェイスの詳細を記録します。このパラメータの値が指定されていないか、またはこのパラメータがコメント アウトされている場合、scope にはアイドル状態ではないすべてのネットワーク インターフェイスの詳細が記録されます。

fsthreshold

fsthreshold パラメータは、FILESYSTEM クラスのしきい値を指定します。file system データ クラスのしきい値基準は、ファイルシステムによるディスク使用率 (FS_SPACE_UTIL メトリック) に基づきます。

parm ファイルのデフォルトの設定を使用すると、scope はディスク容量の 70% 超えて使用しているファイルシステムの詳細を記録します。

lvthreshold

lvthreshold は、LOGICALVOLUME クラスのしきい値を指定します。論理ボリューム データ クラスのしきい値は、1 秒当りの I/O (LV_READ_RATE + LV_WRITE_RATE) を基準にしています。

parm ファイルのデフォルトの設定を使用すると、scope は 1 秒当りに 35 を超える I/O を実行した論理ボリュームの詳細を記録します。

bycputhreshold

bycputhreshold パラメータは、CPU クラスのしきい値を指定します。CPU データ クラスのしきい値基準は、CPU がビジーであった時間の割合 (BYCPU_CPU_TOTAL_UTIL) に基づきます。

parm ファイルのデフォルトの設定を使用すると、scope はビジーとなっている時間の割合が 90% を超えている CPU の詳細を記録します。

scopetransactions

scope コレクタには、トランザクションを生成するアプリケーション応答測定 (ARM: Application Response Measurement) API コールが組み込まれています。scopetransactions フラグによって、scope トランザクションを記録するかどうかが確認されます。デフォルトは scopetransactions=on です。つまり、scope は Scope_Get_Process_Metrics と Scope_Get_Global_Metrics の 2 つのトランザクションを記録します。この 2 つの scope トランザクションを記録しない場合は、scopetransactions=off を指定してください。3 番目のトランザクションの Scope_Log_Headers は必ず記録されます。つまり、scopetransactions=off の影響を受けることはありません。

ARM については、335 ページの「トランザクション追跡の紹介」を参照してください。

subprocinterval

subprocinterval パラメータを指定すると、scope がプロセス データのサンプリングに使用するデフォルトの設定が変更されます。プロセス データおよびグローバル データは、parm ファイルで指定した間隔で定期的に記録されます。ただし、scope は、短期間のアクティビティを対象とするため、数秒ごとに測定します。デフォルトでは、このインストルメンテーションのサンプリング インターバルは 5 秒です。プロセス データのログ間隔は、subprocinterval の偶数倍になります。詳細情報は、36 ページの「データ記録間隔の設定」を参照してください。

無数のアクティブなスレッドまたはプロセスがあるシステムでは、scope の全体的なオーバーヘッドを減らすために、subprocinterval を長くする必要も生じます。また、多数の短期間のプロセスを記録する必要があるシステムでは、subprocinterval を短く設定することもできますが、この場合は scope のオーバーヘッドへの影響を注意して監視しなければなりません。この設定には、parm ファイルに指定したプロセスのログのインターバルの因数となる値を指定する必要があります。

subprocinterval に小さな値を指定すると、グローバル メトリックとアプリケーションの合計とのギャップが減少します (HP-UX 以外の他のすべてのオペレーティング システムで有効)。

gapapp

parm ファイルの gapapp パラメータは、データのアプリケーション クラスの変更を制御することで、グローバル (システム全体) データとアプリケーション データの合計との差異を説明します。

アプリケーション データは、プロセス レベルのインストルメンテーションによって得られます。通常、グローバル メトリックとアプリケーションの合計には差異が生じます。プロセス作成の割合の高いシステムでは、その差異が大きくなります。以下のオプションを選択できます。

- gapapp が空白。gapapp という名前のアプリケーションがアプリケーション リストに追加されます。
- gapapp = UnassignedProcesses。UnassignedProcesses という名前のアプリケーションがアプリケーション リストに追加されます。
- gapapp = ExistingApplicationName (または) gapapp = other。指定したアプリケーションにグローバル値との差異が追加されます。個別に記録されたり、新しいエントリがアプリケーション リストに追加されることはありません。

wait

wait パラメータ (HP-UX のみ) を使用すると、システム リソースを待っているプロセスの詳細を取得できます。wait パラメータの値はパーセンテージで指定できます。プロセスがシステム リソース (cpu、disk、mem、sem、lan) を待つ場合。待ち間隔の割合が wait パラメータに指定された値を上回る場合に、プロセスの詳細が logproc ファイルに記録されます。

値およびオプションについては、「scope が使用する parm ファイル パラメータ」を参照してください。

たとえば、プロセスの記録間隔が 60 秒に、CPU の wait パラメータが 50% に定義されている場合、CPU を待っている時間が 30 秒以上となったすべてのプロセスが logproc ファイルに記録されます。

size

size パラメータは、生ログ ファイルの最大サイズを (メガバイト単位で) 設定するために使用されます。サイズは 1 メガバイト未満には設定できません。

scope コレクタは、開始時にこれらの指定を読み込みます。これらのログ ファイルのいずれかが収集作業中に最大サイズに達した場合でも、そのログ ファイルは、自動的にロール バックされる mainttime 時までは増大し続けます。ロール バック中、最も古いものから 25% のデータがログ ファイルから削除されます。size パラメータによって制限されていない場合でも、生ログ ファイルには最大 1 年分のデータしか保存されません。第 3 章にある「utility 走査レポートの詳細」のセクションの 65 ページの「ログ ファイルの内容の要約」および 66 ページの「ログ ファイルの空きスペースの要約」を参照してください。

parm ファイル内の size 指定を変更すると、scope の開始時にその変更が検出されます。最大ログ ファイル サイズが、既存のデータを収められないほど小さくなると、scope の開始時にサイズ変更が自動的に行われます。新たに指定した最大サイズで既存のデータを収められる場合、どのような動作も行われません。

resize コマンドは、元のログ ファイルを削除する前に、TMPDIR 環境変数で設定されているディレクトリに新しいファイル scopelog を作成します。「使い方」を参照してください。

ログ ファイルの最大サイズについて行った変更は、mainttime パラメータで指定した時刻に有効になります。

size パラメータに関係なく、scope ログ ファイルの最大サイズも、過去 1 年間に格納されたデータの総量に限定されます。生 scope ログ ファイルには 1 年を超えるデータを記録することができず、ログがその期間を超えた場合、1 年を超える古いデータが上書きされます。1 年を超えるデータが必要な場合のアーカイブ ログ ファイルの作成方法については、121 ページの「extract」を参照してください。

mainttime

ログ ファイルは、必要に応じて毎日特定の時刻に scope によってロール バックされます。デフォルトの時刻は、mainttime パラメータを使用して変更できます。たとえば、**mainttime=8:30** を設定すると、ログ ファイルの保守は毎日 8:30 am に行われます。

mainttime は、システム利用率が最も低い時刻に設定することをお勧めします。

ログ ファイルの保守では、1 日を超える古いデータのみがロール アウトされますので、logproc のようにすぐに大きくなって 24 時間以内に限度に達するようなログ ファイルの場合は、設定した最大サイズを超えるおそれがあります。

days

days パラメータは、すべての生データ ログ ファイルが格納可能なデータの最大日数を指定します。このパラメータの値は、1 から 365 の範囲で指定します。このパラメータを使用することで、scope データ コレクタでログ ファイルの保守を行えます。

データ収集時に、ログ ファイル内のデータ日数が days パラメータで指定した日数に達した場合、maintweekday パラメータで指定した日に一致するまでデータ収集が続行されます。maintweekday に達すると、ログ ファイルは mainttime に自動的にロール バックされます。ロール バック中、days パラメータが最大値に達した後に収集されたデータがログ ファイルから削除されます。

データ収集中にログ ファイルのロール バックが行われる際、days パラメータより前の特定の日に size パラメータに指定した値になった場合、days パラメータよりも size パラメータが優先されます。

たとえば、parm ファイルに「size global=20」および「days global=40」と設定されており、ログ ファイルがデータがログ ファイルに記録される 40 日前に最大サイズの 20 MB に達した場合、size パラメータに基づいてログ ファイルのロール バックが実行されます。

maintweekday

maintweekday パラメータは、days パラメータが満たされた場合にログ ファイルのロール バックを実行する曜日を指定します。ロール バックは mainttime に実行されます。

たとえば、parm ファイルで「maintweekday=Mon」と設定されている場合、days パラメータで指定された値が「月曜日」となると、mainttime にログ ファイルのロール バックが実行されます。システム使用率の低い曜日に maintweekday の値を設定することをお勧めします。

maintweekday パラメータはオプションのパラメータです。parm ファイルで maintweekday パラメータを指定する場合、days パラメータとと共に使用する必要があります。parm ファイルで days パラメータが設定されていない場合は、このパラメータは考慮されません。parm ファイルで maintweekday パラメータは指定せずに days パラメータを指定した場合、デフォルト値は「maintweekday=Sun」となります。

たとえば、「daysglobal=30」、「application=20」、「process=30」、「device=20」、「transaction=10」、「maintweekday=Wed」と設定されており、ログ ファイルが days パラメータで指定した日数に達した場合、データ収集は maintweekday で指定した日まで続行されます。maintweekday に達すると、超過した日数分のデータがログ ファイルの先頭から削除され、ログ ファイルのロール バックが実行されます。この保守は mainttime に実行されます。

javaarg

このパラメータは UNIX/Linux のみで有効です。

javaarg パラメータは、true または false のいずれかに設定するフラグです。proc_proc_argv1 メトリックの値にのみ影響します。

javaarg を false に設定する、または parm ファイルに指定しない場合、proc_proc_argv1 メトリックは、常にプロセスのコマンド文字列の最初の引数の値に設定されます。

javaarg が true に設定されていると、コマンド文字列に class または jar 仕様があれば、proc_proc_argv1 メトリックよりも優先されます (java プロセスの場合のみ)。つまり、オペレーティング システムの提供する引数リストにデータが存在すると仮定した場合、ファイル名が java または jre のプロセスについては、proc_proc_argv1 メトリックよりも、-classpath または -cp の後にない最初の引数 (先頭にダッシュのないもの) が優先されます。

複雑に思えるかもしれませんが、java プロセスがシステムで実行されている場合は非常に簡単です。つまり、**javaarg=true** と設定すると、proc_proc_argv1 メトリックが class または jar 名と共に記録されます。これは java 固有のアプリケーションを定義する場合に便利です。クラス名が proc_proc_argv1 にあれば、argv1= application 修飾子を使用して、クラス名でアプリケーションを定義できます。

flush

flush パラメータは、アプリケーション データおよびデバイス データのすべてのインスタンスを記録する、データの記録間隔 (秒単位) を指定します。flush 間隔は、300 から 32700 の範囲にある 300 の偶数倍の数字で指定します。

アプリケーション データおよびデバイス データのすべてのインスタンスに対する、flush 間隔のデフォルト値は 3600 秒です。

flush パラメータを無効にするには、その値に 0 (ゼロ) を指定します。flush パラメータが 0 に設定されていると、scope は parm ファイルで指定されたしきい値に一致しないアプリケーション データおよびデバイス データを記録しません。

zone_app

zone_app フラグを使用すると、Performance Collection Component で Solaris ゾーン固有のデータを収集できます。zone_app フラグが true に設定されていると、すべてのアプリケーション クラス (APP_*) メトリックの収集に影響します。parm ファイルにリストされているすべてのユーザー定義アプリケーション セットは無視され、Performance Collection Component がインストールされているマシンで実行されているゾーンに基づいて、アプリケーション メトリックが収集されます。

たとえば、2 つの非グローバル ゾーン「zone1」および「zone2」を持つ Solaris マシンを考えた場合、Performance Collection Component は parm ファイルのアプリケーション セットを無視し、「global」、「zone1」、「zone2」という名前の 3 つのアプリケーションを作成します。各アプリケーションのパフォーマンス測定は、それぞれのゾーンで実行されているプロセスから取得された測定値を基準にします。たとえば、「zone1」ゾーンの APP_CPU_TOTAL_UTIL メトリックは、zone1 で実行されているすべてのプロセスに対する CPU 使用率の値に基づいて計算されます。

zone_app フラグが無効に設定されているか、Performance Collection Component が Solaris 9 以前のバージョンで実行されている場合、APP_LS_ID メトリックは使用不可 (na) の値を報告します。アプリケーションのグループ化は parm ファイルにリストされているユーザー定義のアプリケーション セットを基準に実行されます。

➤ ゾーンは、Solaris 10 以降のバージョンのみが対応しています。

proccmd

➤ このパラメータは UNIX/Linux のみで有効です。

proccmd パラメータを使用すると、HP Operations エージェント データベースにプロセス コマンドを記録できます。プロセス コマンドの長さは、パラメータ内に数値で指定します。最大値は 1024 です。デフォルトでは、このパラメータの値は 0 に設定されており、プロセス コマンドの記録は無効になっています。

ignore_mt

このパラメータを true に設定すると、Performance Collection Component は、監視対象システム上のアクティブなコアの数に対してメトリック値を正規化した後に、Global クラスの CPU に関連するすべてのメトリックを記録します。

このパラメータを false に設定すると、Performance Collection Component は、監視対象システム上のスレッドの数に対してメトリック値を正規化した後に、Global クラスの CPU に関連するすべてのメトリックを記録します。

システムのマルチスレッド プロパティが無効になっている場合、このパラメータは Performance Collection Component によって無視されます。その結果、GBL_IGNORE_MT メトリックの値は true として記録されます。

➤ Windows、Linux、Solaris システムで、同時マルチスレッディング (Simultaneous Multithreading (SMT)) を有効 / 無効にした場合は、システムを再起動する必要があります。

アプリケーション定義パラメータ

application、file、user、group、cmd、argv1、or の各パラメータは、アプリケーションの定義に関するものです。

Performance Collection Component では、論理的に関連性のあるプロセスを 1 つのアプリケーションにまとめることで、そのプロセスがメモリや CPU などのコンピュータ リソースに与えた総合的な影響を記録できます。

PRM モード (HP-UX のみ) では、アクティブな PRM グループが記録され、parm ファイルにリストされているユーザー定義のアプリケーション セットは無視されます。

アプリケーションは、ファイルのリスト (基本プログラム名)、コマンドのリスト、またはファイルおよびコマンドの組み合わせで設定できます。ユーザー名、グループ名または引数選択で修飾することも可能です。これらのすべてのアプリケーション修飾子は個別で使用できるほか、それぞれを組み合わせて使用することもできます。たとえば、cmd および user の 2 つの修飾子が使用されている場合、プロセスを該当のアプリケーションに含めるには、プロセスが両方のコマンド文字列およびユーザー名の仕様を満たす必要があります。次に、各修飾子の詳細について説明します。

システム上の各プロセスは、1 つのアプリケーションにのみ属します。複数のアプリケーションでカウントされるプロセスはありません。

アプリケーション

アプリケーション名は、複数のプロセスや各プロセスの総合的な活動状況に関するレポートを 1 つにまとめたアプリケーションまたはクラスを定義します。

- アプリケーション名は 19 文字以内の文字列で、アプリケーションの識別に使用されます。
- アプリケーション名には、英字 (小文字 / 大文字)、数字、アンダースコア、および空白を使用できます。parm ファイル内で同じアプリケーション名を複数回使用しないでください。
- application キーワードとアプリケーション名の間の等号 (=) は省略できます。
- application パラメータは、それを参照する file、user、group、cmd、argv1、or の各パラメータの組み合わせの前に入力し、これらのすべてのパラメータが最後のアプリケーション負荷定義に適用されるようにする必要があります。
- 各パラメータは、改行文字を含めて 170 文字以内の長さにします。ただし、文字の間に空白を入れることはできません。ファイルのリストが 170 文字よりも長くなる場合は、改行して行頭に新たに file 文、user 文、group 文、cmd 文、argv1 文のいずれかを入力した後リストの入力を続けてください。
- 最大 998 個のアプリケーションを定義できます。Performance Collection Component には other という名前のアプリケーションが事前に定義されています。この other アプリケーションは、parm ファイル内の application 文で取り込まれていないすべてのプロセスを収集します。

  次に例を示します。

  ```
  application Prog_Dev
  file vi,cc,ccom,pc,pascomp,dbx,xdb

  application xyz
  file xyz*,startxyz
  ```

  すべてのアプリケーションの組み合わせに対して、file、user、group、argv1、cmd を最大で 4096 個定義できます。上記の例には、9 個のファイル指定が含まれています (xyz* は、複数のプログラム ファイルと一致するものであっても、1 つの指定と数えられます)。

  プログラム ファイルが複数のアプリケーションに含まれる場合、そのプログラム ファイルは、それを含む最初のアプリケーションに記録されます。

デフォルトの parm ファイルにはいくつかのサンプル アプリケーションが含まれており、このサンプルを修正して使用できます。examples ディレクトリにも (parm_apps というファイル内に) 別のサンプルがあり、parm ファイルにコピーして、必要に応じて修正することができます。

file

file パラメータは、アプリケーションに属するプログラム ファイルを指定します。これらのプログラムの対話式での実行またはバックグラウンドでの実行がすべて含まれます。このパラメータは、最後に発行された application 文に適用されます。application 文が見つからない場合は、エラーが生成されます。

file name には、次のいずれかを指定できます。

- vi など、1 つの UNIX プログラム ファイル。
- xyz* など、(ワイルドカードで示された) UNIX プログラム ファイルのグループ。この場合、名前が文字 xyz で始まるプログラムが含まれます。ワイルドカードによるファイル指定は、最大限指定しても 1 つの指定と数えられます。

file パラメータで指定できる名前の長さは 15 文字以内です。file パラメータとファイル名の間の等号 (=) は省略できます。

ファイル名は、1 つのパラメータ行に (コンマで区切って) 複数入力しても、file 文を分けて複数入力してもかまいません。ファイル名はパス名で修飾できません。file の指定は特定のメトリック PROC_PROC_NAME と比較されます。このメトリックはプロセスの argv[0] 値 (通常はそのベース名) に設定されています。次に例を示します。

```
application = prog_dev
file = vi,vim,gvim,make,gmake,lint*,cc*,gcc,ccom*,cfront
file = cpp*,CC,cpass*,c++*
file = xdb*,adb,pxdb*,dbx,xlC,ld,as,gprof,lex,yacc,are,nm,gencat
file = javac,java,jre,aCC,ctcom*,awk,gawk

application Mail
file = sendmail,mail*,*mail,elm,xmh
```

file パラメータを指定しないと、その他のパラメータを満たすプログラムがすべて適合することになります。

cmd 修飾子 (下記参照) を除くと、parm ファイルのアプリケーション修飾子がサポートするワイルドカード文字は、アスタリスク (*) だけです。

argv1

argv1 パラメータは、PROC_PROC_ARGV1 メトリックの値によってアプリケーションに選択されたプロセスを指定します。javaarg=true の場合以外は、コマンド ラインの最初の引数が java プロセスの class 名または jar 名であれば、通常その引数が指定されます。このパラメータは、file= および user= といった、parm パラメータと同じパターン マッチ構文を使用します。各選択基準にはワイルドカード マッチ文字としてアスタリスクを使用できるほか、カンマで区切ることで 1 つの行に複数の選択を指定することもできます。

たとえば、次のアプリケーション定義では、コマンド ラインの最初の引数が -title、-fn、-display のいずれかになっているすべてのプロセスがまとめられます。

```
application = xapps
argv1 = -title,-fn,-display
```

次のアプリケーション定義では、特定の java アプリケーションがまとめられます (javaarg=true の場合)。

```
application = JavaCollector
argv1 = com.*Collector
```

次は、argv1 パラメータを file パラメータと組み合わせる方法を示しています。

```
application = sun-java
file = java
argv1 = com.sun*
```

cmd

cmd パラメータは、実行済みのプログラムおよびその引数 (パラメータ) から構成されているコマンド文字列を使用して、アプリケーションに含めるプロセスを指定します。他の選択パラメータとは違い、このパラメータでは、アスタリスクの以外にも詳細なワイルドカード文字を使用できます。

正規表現と同じように、詳細なパターン マッチを使用できます。パターン基準の詳細な説明は、UNIX の fnmatch の man ページを参照してください。他のパラメータと違い、1 つの行には 1 つの選択しかできませんが、複数の行を指定できます。

以下は、cmd パラメータの使用方法です。

```
application = newbie
cmd = *java *[Hh]ello[Ww]orld*
```

user

user パラメータは、アプリケーションに属するユーザー (ログイン名) を指定します。形式は次のようになります。

application < **アプリケーション名** >

file < **ファイル名** >

user [< **ドメイン名** >]\< **ユーザー名** >

user パラメータのドメイン名はオプションです。ローカルでないシステムのユーザー名を指定するには、ドメイン名を指定する必要があります。

次に例を示します。

```
application test_app

file test

user TestDomain\TestUser
```

user パラメータでドメイン名を指定せずにユーザー名を指定すると、そのユーザー名はローカル システムのユーザー名であると見なされます。

次に例を示します。

```
application Prog_Dev

file vi,xb,abb,ld,lint

user ted,rebecca,test*
```

そのアプリケーションに属する類似のユーザー名を確認するには、ワイルドカード アスタリスク (*) のみを使用できます。文字列の後ろにアスタリスク (*) を付けると、アスタリスク (*) より前の文字列と同じ文字列を持つユーザー名を確認でき、文字列の前にアスタリスク (*) を付ける

と、アスタリスク (*) より後ろの文字列と同じ文字列を持つユーザー名を確認できます。user パラメータを指定しないと、その他のパラメータを満たすプログラムがすべて適合することになります。

user パラメータで指定できる名前の長さは 15 文字以内です。

group

group パラメータは、アプリケーションに属するユーザー グループ名を指定します。

次に例を示します。

```
application Prog_Dev_Group2
file vi,xb,abb,ld,lint
user ted,rebecca,test*
group lab, test
```

group パラメータを指定しないと、その他のパラメータを満たすプログラムがすべて適合することになります。

group パラメータで指定できる名前の長さは 15 文字以内です。

or

or パラメータを使用すると、1 つのアプリケーションに複数のアプリケーション定義を適用できます。1 つのアプリケーション定義内では、プロセスは各パラメータ カテゴリのうちの少なくとも 1 つを満たす必要があります。or パラメータで区切られているパラメータは、独立した定義として処理されます。プロセスがいずれかの定義の条件を満たしている場合、そのプロセスはそのアプリケーションに属することになります。

次に例を示します。

```
application = Prog_Dev_Group2
user julie
or
user mark
file vi, store, dmp
```

この例では、ユーザー julie が実行するプログラムとユーザー mark が実行する場合の別のプログラム (vi, store, dmp) で構成されたアプリケーション (Prog_Dev_Group2) を定義しています。

priority

priority パラメータに値を指定すると、アプリケーション内のプロセスを指定範囲内のプロセスだけに制限できます。

次に例を示します。

```
application = swapping
priority 128-131
```

プロセスの優先順位は -511 から 255 までの範囲で、Performance Collection Component を実行しているプラットフォームによって異なります。優先順位は、プロセスの実行中に変更できます。スケジューラがタイム シェア プロセスの優先順位を変更します。また、優先順位はプログラムを使用して、実行中に変更することもできます。

parm ファイルは入力順に処理され、修飾子が最初に一致したときに、特定のプロセスが属するアプリケーションが定義されます。よって通常は、一般的な定義よりも前に特定のアプリケーションの定義が記述されることが多くなります。

アプリケーションの定義の例

次の例はアプリケーションの定義を示したものです。

```
application firstthreesvrs
cmd = *appserver* *-option[123]*
application oursvrs
cmd = *appserver*
user = xyz,abc

application othersvrs
cmd = *appserver*
cmd = *appsvr*
or
argv1 = -xyz
```

次の例は、先行する parm ファイルを使用して、どのように各プログラムが記録されるかを示したものです。

コマンド文字列	ユーザーログイン	アプリケーション
/opt/local/bin/appserver -xyz -option1	**xyz**	firstthreesvrs
./appserver -option5	**root**	othersvrs
./appserver -xyz -option2 -abc	**root**	firstthreesvrs
./appsvr -xyz -option2 -abc	**xyz**	othersvrs
./appclient -abc	**root**	other
./appserver -mno -option4	**xyz**	oursvrs
appserver -option3 -jkl	**xyz**	firstthreesvrs
/tmp/bleh -xyz -option1	**xyz**	othersvrs

データ記録間隔の設定

scope が使用するデフォルトの収集間隔は、プロセス データの場合が 60 秒、グローバル データおよび他のすべてのクラスのデータの場合が 300 秒です。parm ファイルの collectioninterval パラメータを使用するとこのパラメータを無効にできます。値は以下の条件を満たす必要があります。

- プロセス データの収集間隔は、5 から 60 秒で設定します (5 秒間隔)。プロセス データの収集間隔は、subproc 間隔 (「subprocinterval」を参照) の倍数とし、グローバル収集間隔に等分する必要があります。
- グローバル データの収集間隔は、15、30、60、300 秒のいずれかで設定します。グローバル収集間隔はプロセス間隔以上の数値とし、プロセス収集間隔の倍数に設定する必要があります。グローバル収集間隔は、グローバル メトリックおよび、ファイルシステムやアプリケーションといったすべての非プロセス メトリックに適用されます。

例 1:

```
collectioninterval process=15, global=30
subprocinterval = 5
```

この例では、プロセス データの収集間隔が 15 秒、グローバル データおよびデータの他のすべてのクラスが 30 秒、subprocinterval が 5 秒に設定されています。

プロセス データの収集間隔は以下のとおりです。

- 有効値 (5 の倍数の値)
- subprocinterval の倍数、5 秒 (15%5 = 0)
- グローバル データの収集間隔に等分、30 秒 (30%15 = 0)

グローバル データの収集間隔は以下のとおりです。

- 有効値 (15、30、60、300 のいずれか)
- プロセス間隔の倍数、15 秒 (30%15 = 0)

よって、収集間隔のこれらの値は有効です。

例 2:

```
collectioninterval process=15, global=30
subprocinterval = 10
```

この例では、プロセス データの収集間隔が 15 秒、グローバル データおよびデータの他のすべてのクラスが 30 秒、subprocinterval が 10 秒に設定されています。

プロセス データの収集間隔は以下のとおりです。

- 有効値 (5 の倍数の値)
- subprocinterval の倍数ではない、10 秒 (15%10 != 0)
- グローバル収集間隔に等分、30 秒 (30%15 = 0)

グローバル データの収集間隔は 30 秒です

- 有効値 (15、30、60、300 のいずれか)
- プロセス間隔の倍数、15 秒 (30%15 = 0)

よって、収集間隔のこれらの値は無効です。

VMware ESX Server の Performance Collection Component では、グローバル データおよびプロセス データ以外のデータの他のすべてのクラスの記録間隔は 60 または 300 秒で指定します。

ハイパースレッディング / マルチスレッディングが有効なシステムにおける CPU メトリックの正規化

ハイパースレッディング / マルチスレッディング (HT/SMT) が有効なシステムでは、物理的 CPU が 2 つ以上のハードウェア スレッドに対応しています。結果として、複数のソフトウェア プロセスまたはソフトウェア スレッドが、ハードウェア スレッド上で同時に実行されます。マルチコア プロセッサを持つシステムでは、複数のスレッドを個々のコアで同時に実行できます。

Performance Collection Component には複数の CPU 関連メトリックがあり、監視対象システムの CPU 使用率の分析および把握が可能です。デフォルトでは、HT/SMT が有効になっているすべてのシステムで、Performance Collection Component は収集したデータを監視対象システ

ム上で利用可能なスレッドの数に対して正規化し、CPU 関連のすべてのメトリックの値を計算します。単一のスレッドが CPU コア全体を完全に使用している場合、スレッドを基にした正規化によって計算された値が、CPU 使用率の実状を表さない場合があります。

本バージョンの HP Operations エージェントには新しい構成パラメータ ignore_mt が導入されています。これを使用すると、Performance Collection Component で、コアに基づいた正規化により計算された CPU 関連のデータを記録するように設定できます。コアに基づいた正規化によるメトリック値は CPU 使用率の状態をより正確に表しているため、システムのパフォーマンスを分析する際により有効な決定を行うための手助けとなります。

コアに基づいた正規化によって計算されたメトリックの記録

HP-UX では、Performance Collection Component で、コアに基づいて正規化された CPU 関連のすべてのメトリックを記録するように設定できます。他のプラットフォームでは、Performance Collection Component で、コアに基づく正規化を使用した GLOBAL クラスの CPU 関連のメトリックを計算してから記録するように設定できます。

Performance Collection Component で、CPU 関連のすべてのメトリックにコアに基づく正規化を使用するように設定するには、次の手順を実行します。

HP-UX

1. root アカウントでシステムにログオンします。
2. ご使用の環境に合わせて parm ファイルを構成します。parm ファイルには ignore_mt フラグを設定しないでください。

   HP-UX での parm ファイルの ignore_mt フラグの値は、Performance Collection Component の操作にまったく影響しません。
3. 必要に応じてアラームのルールを定義します。
4. 次のコマンドを実行します。

   midaemon -ignore_mt
5. 次のコマンドを使用して、HP Operations エージェントを開始します。

   opcagt -start

   Performance Collection Component は、コアに基づく正規化を用いた CPU 関連のメトリック (すべてのクラス) の記録を開始します。

他のプラットフォーム

1. root アカウントまたは管理者アカウントでシステムにログオンします。
2. ご使用の環境に合わせて parm ファイルを構成します。parm ファイルの ignore_mt フラグを true に設定します。
3. 必要に応じてアラームのルールを定義します。
4. 次のコマンドを使用して、HP Operations エージェントを開始します。

   Windows の場合

   %ovinstalldir%bin\opcagt -start

   HP-UX、Linux、Solaris の場合

   /opt/OV/bin/opcagt -start

AIX の場合

/usr/lpp/OV/bin/opcagt -start

Performance Collection Component は、コアに基づく正規化を用いた、GLOBAL クラスの CPU 関連のメトリックの記録を開始します。

データ収集の終了と再開始

scope コレクタとその他の関連プロセスは、継続的に実行される仕様になっています。これらを終了する必要があるのは、次のいずれかの場合だけです。

- Performance Collection Component ソフトウェアをバージョンアップするとき
- トランザクション構成ファイル、ttd.conf 内でトランザクションを追加または削除するとき（詳細については、335 ページの「トランザクション追跡の紹介」を参照してください）。
- トランザクション構成ファイル、ttd.conf で配布範囲やサービス レベルの目標 (SLO) を変更するとき（詳細については、335 ページの「トランザクション追跡の紹介」を参照してください）。
- parm ファイルを変更し、その変更内容を反映させるとき。parm ファイルに対して行った変更は、scope を開始したときに有効になります。
- utility プログラムの resize コマンドを使用して、Performance Collection Component ログ ファイルのサイズを再変更するとき
- システムを終了するとき
- ハードウェアの追加、または構成の変更を行うとき。変更は、scope を開始したときに有効になります。

データ収集の終了

ovpa および mwa スクリプトの stop オプションを使用すると、scope とその他の Performance Collection Component プロセスの終了時にデータが失われないことが保証されます。

データ収集を手動で終了するには、次のコマンドを入力します。

- *Windows の場合：*

  *%ovinstalldir%*bin**\ovpacmd -stop**

- *HP-UX、Linux、Solaris の場合：*

  /opt/perf/bin/ovpa -stop

- *AIX の場合：*

  /usr/lpp/perf/bin/ovpa -stop

scope は NFS データを記録しませんが、ローカル ファイル システム上の GlancePlus を介することで NFS データを表示できます。

データ収集の再開始

Performance Collection Component プロセスが終了しているときや、構成ファイルを変更し、変更内容を反映させるときにデータ収集を再開始するには、次の方法があります。

coda を使用している場合、システムがダウンした後に scope とその他の Performance Collection Component のプロセスを開始するには、**<インストールディレクトリ>/ovpa start** を使用します。ここで、**インストールディレクトリ** は、Performance Collection Component がインストールされたディレクトリです。

coda を使用している場合、システムがダウンした後に scope と実行中のその他のプロセスを開始するには、**<インストールディレクトリ>/ovpa restart** を使用します。ここで、**インストールディレクトリ** は、Performance Collection Component がインストールされたディレクトリです。これにより、現在実行中のプロセスが終了し、再開始します。

scope を再開始すると、Performance Collection Component は、そのプログラムを終了する前に使用していたログ ファイル (logglob、logappl、logproc、logdev、logtran、logls、logindx) を使用し続けます。新しいレコードは既存のファイルの最後に追加されます。新しいファイル セットにデータを収集し、履歴情報を保持しない場合は、再開始する前にすべての scope ログ ファイル名を変更するか、アーカイブして削除する必要があります。これは、ファイル間でデータを同期させないためです。

ttd.conf 構成ファイルの SEM_KEY_PATH エントリを使用すると、UNIX プラットフォームの ttd および midaemon プロセスで使用されるセマフォの IPC キーを生成できます。使用されるデフォルト値は、/var/opt/perf/datafiles です。

sem id の競合が原因して midaemon または ttd が返答しない場合は、SEM_KEY_PATH の値を変更できます。

夏時間

夏時間の間は、該当するタイム ゾーン内の環境ではシステムの時刻を 1 時間戻します。この時点で、最後に記録されたレコードのタイムスタンプにシステムの時刻が同期するまで、データ収集が 1 時間にわたって停止します。

夏時間を無効に設定すると、システムの時刻が 1 時間進みますので、次に記録されるレコードのタイプスタンプが 1 時間進んでいることになります。この場合は、データ収集が停止しなくても、最後に記録されたレコードの後に 1 時間のギャップが生じます。

システムの時刻の手動変更

システムの時刻を手動で戻すと、データ収集が終了し、**perfstat** などのコマンドが停止します。これらのユーティリティはシステムの時刻を戻した場合に停止します。データの記録を続行し、すべてのコマンドからの応答を取得するには、次の手順を実行します。

1 次のコマンドを実行します。

```
ovc -stop coda
```

2 <*データディレクトリ*>\datafiles\ ディレクトリの coda.* ファイルをバックアップしてから削除します。

3 次のコマンドを実行します。

```
ovc -start coda
```

効果的なデータ収集管理

パフォーマンス分析の効率は、収集するパフォーマンス データへのアクセスがどの程度簡略化されているかによって異なります。この項では、データ収集プロセスを簡略化し、より効果的で便利なものにするために行う、ログ ファイルの管理、データのアーカイブ、システム分析などのアクティビティに対する効果的な方策について説明します。

ログ ファイルにより使用されるディスク スペースの制御

Performance Collection Component は、作成したログ ファイルの自動管理機能を備えています。ユーザーは、この自動プロセスを設定しても、特殊な目的に合わせて手動による代替プロセスを使用してもかまいません。自動ログ ファイル管理プロセスは次のように機能します。

- 各ログ ファイルには、最大サイズが設定されています。Performance Collection Component の初期インストール時にはデフォルトの最大サイズが使用されます。ただし、デフォルト値の設定は変更できます。
- 各ログ ファイルがその最大サイズに達すると、scope データ コレクタが mainttime で指定された時刻に「ロール バック」を実行します。このロール バックでは、ログ ファイル内で最も古いものから 25% のデータが削除され、新しいデータを追加するための空き領域が作られます。

自動ログ ファイルの保守は、scope や DSI の記録プロセスによって収集されるデータと似ていますが、いくつかの相違点があります。DSI ログ ファイルの保守に関する詳細については、247 ページの「データ ソース統合の概要」*を参照してください。*

mainttime の設定

通常、scope は、毎日指定された時刻にログ ファイルのロール バックを実行します。したがって、ロール バックをオフピークの時間帯に実行すると、通常のシステム使用率に影響を与えることはありません。ロール バックに関してログ ファイルを調べる時刻は、parm ファイル内の mainttime パラメータで設定します。

ログ ファイルの最大サイズの設定

ログ ファイルの最大サイズは、ディスク スペースの使用量と、直ちに分析可能な履歴データの量とのバランスを考慮して決定します。ログ ファイルのサイズを小さくするとディスク スペースを節約できますが、Performance Manager などのツールによってグラフを作成できる期間が短くなります。次に、scope ログ ファイルのサイズを変更する方法をいくつか説明します。

scope は、さまざまなデータ型を別々のログ ファイルに記録します。このため、各種のデータに割り当てるディスク スペースを決定できます。たとえば、グローバル データはかなりコンパクトですが、1 か月間のデータを振り返ってグラフを作成する場合があります。これにより、傾向変動分析や容量計画を行うための基本的な統計データを作成できます。

プロセス データは、1 分ごとに多数の対象プロセスが発生する可能性があるため、グローバル データよりも多くのディスク スペースを消費します。また、プロセス データの場合、時間値がグローバル データの場合ほど高くありません。どのようなプロセスが今日と昨日実行されたかについて詳細を把握することがきわめて重要になります。場合によっては、先週実行されたプロセスを把握する必要もあります。ただし、先月実行されたプロセスを正確に把握していても役立つことはほとんどありません。

一般に、ユーザーがオンラインで保持するデータは次のようなものです。

- 3 か月分のグローバル データ (傾向変動分析のためのデータ)
- 1 か月分のプロセス データ (トラブルシューティングのためのデータ)
- 3 か月分のアプリケーション データ (傾向変動分析および負荷配分のためのデータ)
- 2 か月分のデバイス データ (ディスク負荷配分のためのデータ)

parm ファイルを編集して、各ログ ファイルの size パラメータを設定できます。サイズはメガバイト単位で指定します。次に例を示します。

```
SIZE GLOBAL=10.0 PROCESS=30.0 APPLICATION=20.0 DEVICE=5.0
```

特定の日数のデータを保持するために必要なメガバイト数は、データ型、システム構成、およびシステムの活動状況によって異なります。システムのログ ファイルの大きさを決定する最良の方法は、1 週間ほどデータを収集した後、utility プログラムの resize コマンドを使用してログ ファイルのサイズを変更する方法です。resize コマンドは、ログ ファイルを走査して、1 日に収集されるデータ量を確認します。次に、日数からメガバイト数を自動的に算出します。この関数は parm ファイルも更新します。

サイズ変更プロセスの管理

ログ ファイルの自動保守を設定すると、他の設定作業は不要になります。ログ ファイルが、設定されている最大サイズに達すると、ログ ファイルのサイズが scope によって自動的に変更されます。

scope は、parm ファイルの mainttime で指定されている時刻に、ログ ファイルをロール バックします。parm ファイルを編集して scope を再開始しても、ログ ファイルは mainttime の時刻になるまで新しいサイズに変更されません。mainttime で指定した時刻に scope を実行していることが重要です。実行していないと、ログ ファイルはロール バックされません。

保守時刻以外の時間にログ ファイルが最大サイズの設定値を超えても、ロール バックは行われません。

ログ ファイルは、丸 1 日分のデータを保持できないサイズに変更されることはありません。つまり、ログ ファイルは、少なくとも 1 日分のデータを保持するサイズまで増大してからロール バックされます。通常、このことが問題になることはありませんが、大量のプロセス データやアプリケーション データを収集するように parm ファイルのパラメータを設定している場合、またはそのサイズを小さく設定している場合は、ログ ファイルがロール バックされる前にログ ファイルのサイズが最大サイズの設定値を大幅に超えてしまう可能性もあります。

scope は、ログ ファイルが存在しているファイル システムの空きディスク スペースを、グローバル データ収集用に parm ファイルに指定されている間隔で定期的に確認します。空きスペースが 1 メガバイト未満になると、scope は、空きスペースをそれ以上使用しないように次の処理を行います。

- 定期的なログ ファイル保守時刻を待たず、直ちにログ ファイルの保守を実行します。いずれかのログ ファイルがその最大サイズを超えた場合 (および、そのログ ファイルに 1 日分以上のデータが保持されている場合)、そのログ ファイルはロール バックされます。

- ログ ファイルの保守が終わってもディスクの空きスペースが 1 メガバイト以下の場合、scope は、適切なエラー メッセージをその status.scope ファイルに書き込み、データの収集を終了します。

データのアーカイブ

自動ログ ファイル管理は、分析に使用できる最新のログ ファイル データを保持します。生ログ ファイルからのデータをアーカイブします。プロセス データおよびグローバル データは、parm ファイルで指定した間隔で定期的に記録されます。詳細情報は、36 ページの「データ記録間隔の設定」を参照してください。新しいデータのスペースを確保するために、ログ ファイルが最大サイズに達すると最も古いデータから削除されます。ログ ファイル データの保存期間を長くする場合、データのアーカイブ処理を行います。どのような処理を行うかは、ユーザーの必要性によって異なります。次に、いくつかの方法を示します。

- 生ログ ファイルのサイズを非常に大きくして、残りの処理は自動ログ ファイル保守に任せます。これは、最も簡単なアーカイブ方法ですが、数か月後には大量のディスク スペースを消費している可能性があります。
- 生ログ ファイルから削除される前に、データを抽出アーカイブ ファイルに抽出します。このアーカイブ ファイルは、将来必要になるときに備えて、テープなどの長期記憶装置にコピーしてください。
- 生ログ ファイルのサブセットだけを抽出アーカイブ ファイルに抽出します。たとえば、容量が大きくて時間値が低いためにプロセス データがアーカイブされない場合があります。
- 上記の方法を組み合わせて使用する方法もあります。

データのアーカイブは次の方法で行うことをお勧めします。

- オンラインで保持する詳細データの量が収まるように生ログ ファイルのサイズを決定します。
- 週に 1 度、オフライン記憶装置に移すファイルに詳細な生データをコピーします。

アーカイブ処理の管理

前の項の説明に従って、生ログ ファイルのサイズを変更してください。最低 2 週間分のデータを保持できるログ ファイル サイズを選択します (アーカイブ処理を週に 1 度だけ行うと仮定します)。

週に 1 度、extract プログラムを実行するプロセスをスケジュールします。以下に、毎週処理を実行するスクリプト ファイルのサンプルを示します。

```
#extract -gapdt -xm
```

その月の各週のデータが、前の週のデータに追加されます。新しい月になると、extract は新しいアーカイブ ログ ファイルを作成し、その週のデータを該当する月間アーカイブ ログ ファイルに分割します。ログ ファイルには rxmo という名前がつけられ、その後に年を表す 4 桁の数字と、月を表す 2 桁の数字が続きます (たとえば、1999 年 12 月に抽出したデータは、rxmo199912 ファイルに記録されます)。

月が変わると前の月のログ ファイルを終了し、新しいログ ファイルを開始します。したがって、ログ ファイル rxmo が複数存在する場合、最新でないファイルは、必要になるまでオフラインの記憶装置にコピーしておく方が安全です。アーカイブ データにアクセスする必要があるときは、対象となるアーカイブ ファイルを復元して、extract プログラムまたは utility プログラムを使用してアクセスしてください。

システム構成や活動レベルによっては、1 か月のディスク スペースの累積使用量が大きくなることもあります。このような場合は、上記の例に示されている -xm コマンドの代わりに -xw コマンドを使用することで、詳細なアーカイブ ファイルを複数の小さなファイルに細分化できます。

また、詳細なプロセス データをアーカイブしないようにも選択できます。

前記の例で説明した詳細の抽出では、収集したパフォーマンス データがすべて保存されます。ある状況を詳しく調べる必要がある場合、これらのファイルをディスクに復元して分析できます。

extract プログラムを使用すると、転送や分析を容易にするために、複数の抽出ファイルの結合データやデータのサブセットを作成できます。

たとえば、複数年の傾向変動分析を行うために、複数の年間抽出ファイルを結合できます。

HP Operations エージェント で作成されたログ ファイルを、古いバージョンの HP Performance Agent を使用しているシステムに移動することはできません。

3 HP Operations エージェント の操作

データ収集機構を設定した後、そのエージェントを HPOM と共に使用する場合、HPOM のポリシーをノードに配布することにより Operations Monitoring Component の異なるコンポーネントを使用できます。たとえば、Measurement Threshold ポリシーを配布すると、監視エージェント コンポーネントは動作を開始します。ほとんどの監視詳細は HPOM ポリシーに指定できますが、そのノード上でさらに設定を行う必要があるコンポーネントもあります。

監視エージェントの設定

さまざまなソースを監視する監視エージェントを起動 / 設定できます。ノードに Measurement Threshold ポリシーを配布すると、監視エージェント コンポーネントは動作を開始します。エージェントは、ポリシーに指定されている内容に基づいて、以下のソースのオブジェクトの監視を開始します。

- **外部 :** エージェントに数値を送信できる外部プログラム。
- **組み込みパフォーマンス コンポーネント :** エージェントのデータ ストアに存在するデータ。
- **MIB:** 管理情報ベース (MIB) のエントリ。
- **リアルタイム パフォーマンス管理 :** Windows のパフォーマンス ログおよびアラート。
- **プログラム :** HPOM によって起動され、エージェントに数値を送信する外部プログラム。
- **WMI:** WMI データベース。

HPOM ポリシーを使用して上記ソースのオブジェクトを監視する方法については、以下のトピックを参照してください。

- *HPOM for Windows: HPOM for Windows* のオンライン ヘルプの「*Event Policy Editors*」
- *HPOM on UNIX/Linux:* 『*HPOM for UNIX 9.00 Concepts Guide*』の「*Implementing Message Policies* 」

MIB オブジェクトを監視するエージェントの設定

ノードに Measurement Threshold ポリシー (ソース タイプを MIB に設定) を配布すると、監視エージェントは、`public` というコミュニティ文字列でアクセスできる MIB オブジェクトの照会を開始します。デフォルト以外のコミュニティ文字列を使用するように監視エージェントを設定するには、次の手順を実行します。

1 ルート権限または管理権限でノードにログオンします。

2 コマンド プロンプト (シェル) に移動します。

3 以下のディレクトリに移動します。

Windows 環境:

*%ovinstalldir%*bin

HP-UX、*Solaris*、*Linux 環境:*

/opt/OV/bin

AIX 環境:

/usr/lpp/OV/bin

4 次のコマンドを実行します。

- デフォルト以外のコミュニティ文字列を使用するには

  ovconfchg -ns eaagt SNMP_COMMUNITY < *コミュニティ文字列* >

  この < *コミュニティ文字列* > には、デフォルト以外のコミュニティ文字列を指定します。

- 別のコミュニティ文字列を使用するには

  ovconfchg -ns eaagt SNMP_COMMUNITY_LIST < *コミュニティ文字列のリスト* >

  この < *コミュニティ文字列のリスト* > には、使用するコミュニティ文字列のカンマ区切りのリストを指定します。HP Operations エージェントは、上記コマンドに指定された順にコミュニティ文字列のリストを処理します。

  次に例を示します。

  ovconfchg -ns eaagt SNMP_COMMUNITY_LIST "C1,C2,C3"

  HP Operations エージェント は、ノードとの SNMP セッションを確立し、まず、コミュニティ文字列 C1 を使用して OID に対して SNMP Get 処理を試みます。この操作が失敗すると、HP Operations エージェントはコミュニティ文字列 C2 を使って同じ操作を実行し、成功するまで順にこれを繰り返します。

SNMP_COMMUNITY_LIST に指定されているすべてのコミュニティ文字列を使っても HP Operations エージェントが失敗する場合は、SNMP_COMMUNITY に指定されているコミュニティ文字列が使用されます。指定されているすべてのコミュニティ文字列を使ってもエージェントがデータを取得できない場合は、デフォルトのコミュニティ文字列である public が使用されます。

子 OID の監視

MIB Measurement Threshold ポリシーを作成するときは、[MIB ID] フィールドに監視する MIB オブジェクトの OID を指定する必要があります。オブジェクトの下位の子 OID を監視する場合は、OID を以下の形式で [MIB ID] フィールドに入力します。

< *親 OID*>**.***

たとえば、オブジェクト .1.3.6.1.2.1.2.2.1 のすべての子 OID を監視するには、[MIB ID] フィールドに **.1.3.6.1.2.1.2.2.1.*** と指定する必要があります。

子インスタンスの名前または説明の位置を以下の形式で指定することもできます。

< *親 OID*>**.*:**<*n*>

この <*n*> には、子インスタンスの名前または説明の位置を指定します。

たとえば、オブジェクト .1.3.6.1.2.1.2.2.1 のすべての子 OID の中から、.1.3.6.1.2.1.2.2.1 の MIB サブツリーの 2 番目の要素に記述されている子 OID を監視対象とする場合、[MIB ID] フィールドに **.1.3.6.1.2.1.2.2.1.*:2** と指定する必要があります。

監視対象オブジェクトの永続化

このバージョンの HP Operations エージェントでは、中断や障害のイベントに備えて、監視対象オブジェクトとセクション変数の値を定期的に保存します。ただし、監視対象オブジェクトとセッション変数の値を保存しないようにエージェントを設定することもできます。

<h2> 監視対象の値を保存しないようにエージェントを設定するには :</h2>

1. 適切な権限でノードにログオンします。
2. コマンド プロンプトで以下のコマンドを実行します。

   ovconfchg -ns eaagt set OPC_MON_SAVE_STATE FALSE

   これ以降、エージェントは監視対象オブジェクトとセッション変数の値の保存を停止します。

イベント インターセプタの設定

イベント インターセプタは、デフォルトではリモート管理ステーションまたは SNMP 対応デバイスから出力される SNMP トラップを収集し、設定に基づいて適切なイベントを生成できます。イベント インターセプタのデフォルト動作は、以下のプロパティの設定によって変更できます。

- SNMP_TRAP_PORT: デフォルト ポートは **162** です。この値は、HP Operations エージェント ノードの使用可能な任意のポート番号に変更できます。
- SNMP_TRAP_FORWARD_DEST_LIST: このプロパティを使用することで、使用できるすべての SNMP トラップの転送先となるリモート管理ステーションのアドレスを設定できます。カンマで区切ることで、複数のシステム名 (およびポートの詳細) を指定できます。
- SNMP_TRAP_FORWARD_ENABLE: デフォルトでは、このプロパティは **FALSE** に設定されています。このプロパティを **TRUE** に設定した場合、イベント インターセプタは、HP Operations エージェント ノードで使用できる SNMP トラップをリモート マシンまたは管理ステーションに転送できるようになります。
- SNMP_TRAP_FORWARD_COMMUNITY: このプロパティを使用することで、受信トラップのソース マシンと、SNMP トラップの転送先となるターゲット マシンのコミュニティ文字列を指定できます。ソース マシンとターゲット マシンのコミュニティ文字列は、一致している必要があります。
- SNMP_TRAP_FORWARD_FILTER: このプロパティを使用することで、使用できる SNMP トラップを OID でフィルタ処理し、選択されたトラップのみをリモート マシンに転送できます。このフィルタ メカニズムでは、ワイルドカード文字 (*) が有効です。たとえば、このプロパティを **1.2.3.*.*.*** に設定した場合、イベント インターセプタは OID が 1.2.3 から始まるすべての SNMP トラップを転送します。イベント インターセプタによるトラップの転送を有効にした場合、デフォルトでは、使用できるすべてのトラップが転送されます。

  ➤ ソース マシンとターゲット マシンのコミュニティ文字列が一致しない場合、トラップ転送機能は失敗します。

イベント インターセプタのデフォルト動作を変更するには、次の手順を実行します。

1. 適切な権限でノードにログオンします。
2. コマンド プロンプトで以下のコマンドを実行します。

- ポート番号を変更するには、以下のコマンドを実行します。

  ovconfchg -ns eaagt set SNMP_TRAP_PORT < *ポート番号* >

  < *ポート番号* > には整数値を指定する必要があります。指定する < *ポート番号* > が使用可能であることを確認してください。

- イベント インターセプタが SNMP トラップをリモート マシンに転送できるようにするには、以下のコマンドを実行します。

  ovconfchg -ns eaagt set SNMP_TRAP_FORWARD_ENABLE TRUE

- イベント インターセプタが SNMP トラップをリモート マシンに転送できるようにする場合は、以下のコマンドを実行してターゲット マシンの詳細を指定します。

  ovconfchg -ns eaagt set SNMP_TRAP_FORWARD_DEST_LIST "< *マシン名* >:< *ポート* >**"**

  < *マシン名* > には SNMP トラップの転送先となるマシンの完全修飾ドメイン名、< *ポート* > にはそのマシンの HTTPS ポートを指定します。複数のターゲット マシンを指定するには、マシンの詳細をカンマで区切ります。

- ノードで使用できる SNMP トラップのうち、選択したトラップのみをリモート マシンに転送するには、以下のコマンドを実行します。

  ovconfchg -ns eaagt set SNMP_TRAP_FORWARD_FILTER "<*OID フィルタ* >**"**

  <*OID フィルタ* > には、OID とワイルドカード文字を指定します。イベント インターセプタは、使用できるトラップから指定の OID (およびワイルドカード文字) と一致するトラップをフィルタリングし、これらのトラップのみをターゲット マシンに転送します。

RTMA コンポーネントの設定

リアルタイム メトリック アクセス (RTMA) コンポーネントにより、ローカルまたはリモートで、システム パフォーマンス メトリックにリアルタイムでアクセス可能となります。HP Operations エージェントをインストールすると、RTMA コンポーネントの一部である perfd プロセスは、デフォルトの設定でノードで動作を開始します。perfd プロセスの設定は perfd.ini ファイルで変更できます。このファイルは、ノードの以下のディレクトリにあります。

- Windows の場合 : *%ovdatadir%*
- UNIX/Linux:/var/opt/perf

パラメータ	説明	デフォルト値
interval	データ収集の頻度 (単位は秒)。この値は、60 の倍数または約数である必要があります。	10
port	perfd が使用するポート。	5227
depth	perfd キャッシュでグローバル メトリック値を維持する時間。このデータは、データの要約に使用されます。	30
maxrps	perfd が 1 秒間に受け付けるセッション要求の最大数。要求の数がこの制限を超えると、perfd は 1 秒間停止し、このイベントの詳細をログファイルに記録します。このログファイル (status-perfd.< ポート >) は、ノードの以下のディレクトリにあります。 • Windows: *%ovdatadir%* • *UNIX/Linux:* `/var/opt/perf`	20
maxtpc	perfd が受け付ける、クライアントシステムごとのセッションの最大数。使用可能セッションの数がこの制限に達した状態で、追加の要求を受信した場合、perfd はその要求を拒否します。	30
maxcps	perfd が一度に受け付ける同時セッション要求の最大数。要求の数がこの制限を超過すると、サーバーは 3 秒間停止してからセッションを確立します。	2

パラメータ	説明	デフォルト値
lightweight	このパラメータを true に設定すると、perfd はプロセス、アプリケーション、NFS 操作、論理システム、ARM のデータ収集を停止します。また、HP-UX 上の HBA および LVM データも収集されなくなります。	false
localonly	このプロパティを true に設定すると、perfd への接続をローカル マシンのみに限定できます。 true に設定した場合、perfd は、ループバック インターフェイスを経由したホスト システム (ローカル ホスト) からの要求を除く、すべての接続要求を拒否します。拒否された接続要求の詳細は、ステータス ファイルに記録されます。	false
ipv4	このオプションを true に設定すると、perfd は IPv4 接続のみを受け付けるようになります。perfd が IPv6 ソケットを作成できない場合、デフォルトでは、IPv4 のみのソケットに自動的に切り替わります。このオプションは、すべての IPv6 通信を明示的に無効にする場合に使用されます。	false

<h2> **デフォルト設定を変更するには :**</h2>

1. テキスト エディタを使用して、ノードの `perfd.ini` ファイルを開きます。
2. 設定を変更します。
3. ファイルを保存します。
4. HP Operations エージェントを再起動して設定変更を有効にします。

エージェント ユーザーの設定

Operations Monitoring Component は root 以外のユーザーや権限のないユーザーで一も実行できますが、Performance Collection Component は常に root または管理者ユーザーで実行する必要があります。root や管理者権限のないユーザーで HP Operations エージェントを実行するように設定すると、Performance Collection Component を使用することはできなくなります。さらに、もっとも重要な点として、システム パフォーマンス データを収集できなくなります。

インストール後、HP Operations エージェントは Windows ノードではローカル システムのアカウントで、UNIX/Linux ノードでは root アカウントで稼動し始めます。ただし、デフォルト以外のアカウントで実行するようにエージェントを設定できます。

エージェントは、そのエージェントが現在実行されているユーザー アカウントで、自動コマンドまたはオペレータ起動コマンドを開始します。ただし、異なるユーザー アカウントでコマンドを実行するようにエージェントを設定できます。

関連トピック：

Windows でのデフォルト ユーザーの変更

UNIX/Linux でのデフォルト ユーザーの変更

Windows でのデフォルト ユーザーの変更

Windows で使用するエージェント ユーザーは、次の要件を満たしている必要があります。

- エージェントを実行するユーザー権限
  - サービスとしてログオン
  - 監査およびセキュリティ ログの管理
- ノードを管理するユーザー権限
  - システムのシャットダウン

    これにより、エージェントはシステムをシャットダウンできます (ユーザーがコンソールでシャットダウン ツールを開始する場合など)。
  - プログラムのデバッグ

    これにより、 エージェントはプロセスの情報を収集したり、 プロセスを停止したりできます (ユーザーがコンソールで list プロセスや kill プロセス ツールを開始する場合など)。
- エージェント ユーザー以外のユーザーで、エージェントからコマンドとツールを開始できるようにするユーザー権限
  - オペレーティング システムの一部として動作。
  - プロセスのメモリ容量の調節 (Windows の一部のバージョンでは、クォータの増加とも呼ばれる)
  - プロセスレベル トークンの置き換え
- レジストリ エントリの許可
  - `HKEY_LOCAL_MACHINE/Software/Hewlett-Packard/OpenView`

    ユーザーにはこのレジストリ キーとすべての子オブジェクトのフル コントロールが必要です。
  - `HKEY_LOCAL_MACHINE/Software/Microsoft/WindowsNT/CurrentVersion/Perflib`

    エージェントがパフォーマンス データにアクセスするには、ユーザーにこのレジストリ キーの読み取り権限が必要です。

実行する必要がある管理作業の権限を追加で割り当てる必要がある場合があります。次に例を示します。

- ポリシーを使用してログ ファイルを監視できるようにする場合、エージェント ユーザーにはログ ファイルの読み取り権限が必要です。
- 自動コマンド、オペレータ起動コマンド、ツール、予定タスクを使用してプログラムを開始できるようにする場合、エージェント ユーザーにはプログラムを開始する権限が必要です。

  さらに、eaagt 名前空間にパラメータ OPC_PROC_ALWAYS_INTERACTIVE=NEVER を設定する必要があります。このパラメータはエージェント インストールのデフォルトに設定することも、コマンド プロンプトで ovconfchg または ovconfpar を使用して設定することもできます。このパラメータを設定すると、エージェントが起動するプロセスは、デフォルトのデスクトップにアクセスできなくなります。この設定は、ログファイル エンキャプスレータの前処理と、監視エージェントが起動するスクリプトに適用されます。
- 管理者権限のないユーザー アカウントでエージェントを実行する場合、Smart Plug-in の一部では設定やユーザー権限を追加する必要がある場合があります。詳細については、それぞれの Smart Plug-in のドキュメントを参照してください。

<h2>Windows **でデフォルト ユーザーを変更するには :**</h2>

1. *オプション:* エージェントを実行する新しいユーザーを作成します。
2. *オプション:* 新しいグループを作成し、このグループのメンバーとしてユーザーを追加します。
3. ノードでコマンド プロンプトを開き、次のコマンドを実行します。

   cscript "%OvInstallDir%bin\ovswitchuser.vbs"-existinguser < *ドメイン*\ *ユーザー*> **-existinggroup** < *グループ*> **-passwd** < *パスワード*>

   この例では次のようになります。

   < *ドメイン*\ *ユーザー*> はドメインとユーザー名です。

   < *グループ*> はユーザーが属するグループの名前です。たとえば、AgentGroup です。

   < *パスワード*> はユーザーのパスワードです。

   このコマンドは、基本のエージェント機能に必要なユーザー権限を、個人ユーザーにではなくグループ レベルで割り当てます。そのため、使用するグループの選択には注意してください。このエージェント ユーザーのための新しいグループを作成し、エージェント ユーザーをメンバーとして追加することをお勧めします。

4. エージェントを再起動するには、以下のコマンドを実行します。

   a **ovc -kill**

   b **ovc -start**

   これで、指定したユーザーで、制御サービスとエージェント プロセスが実行されました。

関連トピック :

エージェント ユーザーの設定

UNIX/Linux でのデフォルト ユーザーの変更

UNIX/Linux でのデフォルト ユーザーの変更

デフォルトでは、UNIX または Linux のオペレーティング システムのノードでは、エージェントは root アカウントで実行されます。ただし、異なるユーザー アカウントで実行するようにエージェントを設定できます。たとえば、root アカウントよりも権限が少ないアカウントでエージェントを実行することもできます。または、ネットワークを経由してリモート システムにアクセスする権限を持つアカウントでエージェントを実行することもできます。

エージェントを実行してノードを正しく管理するための適切な権限がユーザー アカウントにあるかどうかをテストする必要があります。実行する必要がある管理作業の権限を追加で割り当てる必要がある場合があります。次に例を示します。

- ポリシーを使用してログ ファイルを監視できるようにする場合、エージェント ユーザーにはログ ファイルの読み取り権限が必要です。
- 自動コマンド、オペレータ起動コマンド、ツール、予定タスクを使用してプログラムを開始できるようにする場合、エージェント ユーザーにはプログラムを開始する権限が必要です。
- 別のユーザーでエージェントを実行する場合、Smart Plug-in の一部では設定やユーザー権限を追加する必要がある場合があります。詳細については、それぞれの Smart Plug-in のドキュメントを参照してください。

<h2>UNIX/Linux でデフォルト ユーザーを変更するには :</h2>

1. *オプション:* エージェントを実行する新しいユーザーを作成します。
2. *オプション:* 新しいグループを作成し、このグループのメンバーとしてユーザーを追加します。
3. ノードで root としてログインし、シェル プロンプトを開きます。
4. 以下のディレクトリに移動します。

   HP-UX、Linux、Solaris の場合

   `/opt/OV/bin`

   AIX の場合

   `/usr/lpp/OV/bin`
5. 以下のコマンドを実行して、エージェントを停止します。

   ovc -kill
6. 以下のコマンドを実行して、エージェント ユーザーを変更します。

   ovswitchuser.sh -existinguser < *ユーザー* > **-existinggroup** < *グループ* >

   この例では次のようになります。

   < *ユーザー* > は、エージェントを実行するユーザーの名前です。

   < *グループ* > はユーザーが属するグループの名前です。たとえば、AgentGroup です。このコマンドを実行すると、このグループには、エージェント データ ディレクトリ内のすべてのファイルに対するフル コントロールが付与されます。また、すべてのインストール済みパッケージへのフル コントロールも付与されます。以前にこのコマンドを実行して異なるグループを指定している場合、このコマンドを実行すると、以前のグループのファイルのコントロールは削除されます。

グループ ID フラグがエージェントのデータ ディレクトリに設定されます。このフラグは、指定するグループが、エージェントのベース ディレクトリ内のすべての新しいファイルとサブディレクトリも所有することを示します。

このコマンドは、基本のエージェント機能に必要なユーザー権限を、個人ユーザーにではなくグループ レベルで割り当てます。そのため、使用するグループの選択には注意してください。このエージェント ユーザーのための新しいグループを作成し、エージェント ユーザーをメンバーとして追加することをお勧めします。

7 HP Operations エージェントには、管理サーバーからのインバウンド接続を待ち受ける通信ブローカが用意されています。デフォルトのポートは 383 です。ただし、UNIX ノードおよび Linux ノードでは、root 以外のユーザーは 0 から 1023 の範囲のポートを開くことはできません。そのため、異なるポート (1023 より大きいポート) で待ち受けるように通信ブローカをノードで設定する必要があります。また、アウトバウンド接続が正しい宛先ポートを使用するように、ノードに接続する管理サーバーを設定する必要があります。

通信ブローカのポートを設定するには、bbc.cb.ports 名前空間の PORTS パラメータを設定する必要があります。このパラメータを設定する手順は次のとおりです。

- エージェント インストールのデフォルトで値を設定する。これは、多数のノードに通信ブローカのポートを設定する必要がある場合にお勧めします。ノードを作成および移行する前に、インストールのデフォルトを計画して設定する必要があります。
- コマンド プロンプトで ovconfchg または ovconfpar を使用する。

  この値には、次の形式で 1 つ以上のホスト名または IP アドレスを含める必要があります。

  < *ホスト* **>:<** *ポート* **>[,<** *ホスト* **>:<** *ポート* **>] ...**

  たとえば、node1.emea.example.com というホスト名のノードで通信ブローカのポートを 5000 に設定するには、そのノードとそのノードに接続を開くすべての管理サーバーで次のコマンドを使用します。

  ```
  ovconfchg -ns bbc.cb.ports -set PORTS
  node1.domain.example.com:5000
  ```

8 エージェントを起動するには、以下のコマンドを実行します。

a **su** < ***ユーザー*** >

b **ovc -start**

これで、指定したユーザーで、制御サービスとエージェント プロセスが実行されました。

4 utility プログラムの使い方

utility プログラムは、ログ ファイル、収集パラメータ (parm) ファイル、アラーム定義 (alarmdef) ファイルに関する情報を管理し、レポートするためのツールです。対話型モードまたはバッチ モードで utility プログラムを使用すると、次の作業を行うことができます。

- 生ログ ファイルまたは抽出ログ ファイルの走査と、以下の項目を含んだレポートの作成
  - 扱われている日時
  - scope コレクタを実行していない時間
  - scope パラメータ設定値の変更
  - システム構成の変更
  - ログ ファイル ディスク スペース
  - 収集パラメータ (parm) ファイルでのアプリケーションおよびプロセスの設定値の効果
- 生ログ ファイルのサイズ変更
- parm ファイル内の構文の警告やエラーの確認
- alarmdef ファイル内の構文の警告やエラーの確認
- アラーム定義と比較したログ ファイル データの処理と、履歴データ内のアラーム状態の検出

この章では次の内容について説明します。

- utility プログラムの実行
- 対話型モードの使い方
- utility のコマンド ライン インターフェイスの使い方
- utility 走査レポートの詳細

utility プログラムの各コマンドの詳細は、第 5 章「utility のコマンド」を参照してください。

utility プログラムの実行

utility プログラムを実行するには、次の 3 つの方法があります。

- コマンド ライン モード ― utility プログラムはコマンド ラインでコマンド オプションと引数を使用して制御します。
- 対話型モード ― stdin が対話式の端末またはワークステーションに設定されたプログラムを実行するときに、対話型のコマンドとパラメータを入力します。
  使用経験があるユーザーの場合は、特定の作業に必要なコマンドだけを直ちに指定してもかまいません。初めて使用するユーザーの場合は、utility プログラムの guide コマンドを使用してコマンドの使い方のガイドを表示することをお勧めします。ガイド モードでは、タ

スクを実行するときに、オプション リストからの選択を促すプロンプトが表示されます。また、ガイド モードでは、各作業を行う対話型のコマンドが実行時にリストされるため、それらのコマンドの使い方を確認できます。ガイド モードは、随時終了し、再開始できます。

- バッチ モード ― プログラムを実行し、対話型のコマンドとパラメータを含むファイルに stdin をリダイレクトできます。

コマンド ライン インターフェイスの構文は、その他のプログラムで使用する標準的な UNIX コマンド ライン インターフェイスと類似しています。この構文の詳細はこの章で説明します。

対話型モードとバッチ モードのコマンド構文は同一です。コマンドは、どのような順序で入力してもかまいません。コマンドに関連付けるパラメータがある場合、そのパラメータは対応するコマンドの直後に入力する必要があります。

パラメータには、必須のパラメータ (デフォルトなし) とオプションのパラメータ (デフォルトあり) の 2 種類があります。utility がこれらのパラメータを扱う方法は、実行モードによって異なります。

- 対話型モードでオプションのパラメータが欠落している場合は、プログラムがデフォルトの引数を表示します。ユーザーはその引数を確定するか、無効にする必要があります。必須のパラメータが欠落している場合は、プログラムが引数を入力するように促すプロンプトを表示します。
- バッチ モードでオプションのパラメータが欠落している場合、プログラムはデフォルト値を使用します。必須のパラメータが欠落している場合、プログラムは終了します。

対話型モードの場合とコマンド ラインおよびバッチ モードの場合では、エラーと欠落データが異なる方法で処理されます。対話型モードではデータの追加や間違いの訂正ができますが、コマンド ラインやバッチ モードではできません。

対話型モードの使い方

utility プログラムの対話型モードを使用するには、特定の作業を行うための一連のコマンドを発行する必要があります。

たとえば、今日記録されたデータにアラーム状態が存在しているかどうかをログ ファイルで確認する場合は、utility プログラムを呼び出した後に次のコマンドを発行します。

```
checkdef /var/opt/perf/alarmdef
detail off
start today-1
analyze
```

checkdef コマンドが、alarmdef ファイル内のアラーム定義構文を確認し、analyze コマンドで使用するファイル名を設定して保存します。detail off コマンドを発行すると、analyze コマンドがアラームの要約だけを表示します。start today-1 コマンドは、昨日記録されたデータだけを分析するように指定します。analyze コマンドは、デフォルトの SCOPE データ ソース内にある生ログ ファイルを alarmdef ファイルと比較して分析します。

対話型モードとバッチ モードの使用例

次の例では、バッチ モードと対話型モードにおける utility プログラムの resize コマンドの機能の違いを示します。

resize コマンドでは、パラメータを使用して次の設定を指定できます。

- サイズ変更するログ ファイルの種類
- 新しいファイルのサイズ
- ファイルに残す空きスペースの大きさ
- サイズ変更の実行を指定する動作

次の resize コマンドの例では、グローバル ログ ファイルのサイズを、45 日分の空きスペースを残して最大 120 日分のデータを格納できるサイズに変更します。コマンドとそのパラメータは次のようになります。

```
resize global days=120 empty=45 yes
```

このコマンドを対話式に入力しても、バッチ ジョブから入力しても、同じ結果が得られます。

最初のパラメータ (global) は、サイズを変更するログ ファイルを示しています。このパラメータを指定しないと、対話型モードとバッチ モードでは、次のような動作が行われます。

- バッチ モード ― logfile パラメータにはデフォルトがないため、バッチ ジョブが終了します。
- 対話型モード ― サイズ変更を完了するためにログ ファイルの種類を選択するように促すプロンプトが表示されます。

最後のパラメータ (yes) は、サイズ変更を無条件で実行することを示しています。

パラメータ yes を指定しないと、対話型モードとバッチ モードでは、次のような動作が行われます。

- バッチ モード ― yes がデフォルトの動作であるためサイズ変更が続けられます。
- 対話型モード ― サイズ変更を行う前に動作の指定を促すプロンプトが表示されます。

バッチ モードまたは対話型モードで resize コマンドを使用する前に、データ収集を終了する必要があります。詳細は、第 2 章の 39 ページの「データ収集の終了と再開始」を参照してください。

utility のコマンド ライン インターフェイス

対話型モードとバッチ モードのコマンド構文のほかに、コマンド ライン インターフェイスを使用してもコマンド オプションとそれに関連する引数を utility プログラムに渡すことができます。utility プログラムをシェル スクリプトで簡単に呼び出せるようにし、UNIX のパイプへの入出力のリダイレクトを可能にすることで、コマンド ライン インターフェイスは一般的な UNIX 環境に適合します。

たとえば、コマンド ラインを使用して、前述の「対話型モードの使い方」に示されている例と同様の処理をするには、次のように入力します。

```
utility -xr global days=120 empty=45 yes
```

次の表で、コマンド ラインのオプションと引数を示します。それぞれのコマンドの説明は、第 5 章「utility のコマンド」に記載されています。

表 3　　コマンド ラインの引数

コマンド オプション	引数		説明
-b	date	time	分析または走査関数の開始日時を指定します (第 第 4 章の start コマンドの項を参照)。
-e	date	time	分析または走査関数の終了日時を指定します (第 4 章の stop コマンドの項を参照)。
-l	logfile		オープンするログ ファイルを指定します (第 4 章の logfile コマンドの項を参照)。
-f	listfile		出力リスト ファイルを指定します (第 4 章の list コマンドの項を参照)。
-D			analyze、scan および parm ファイルの確認で詳細を有効にします (第 4 章の detail コマンドの項を参照)。
-d			analyze および parm ファイルの確認で詳細を無効にします (第 4 章の detail コマンドの項を参照)。
-v			コマンド ラインのコマンドを実行時にエコーさせます。
-xp	parmfile		parm ファイルの構文を確認します (第 4 章の parmfile コマンドの項を参照)。
-xc	alarmdef		-xa (または analyze コマンド) で使用する alarmdef ファイルの構文を確認し、そのファイル名を設定します (第 4 章の checkdef コマンドの項を参照)。
-xa			alarmdef ファイルと比較してログ ファイルを分析します (第 4 章の analyze コマンドの項を参照)。
-xs	logfile		ログ ファイルを走査し、レポートを作成します (第 4 章の scan コマンドの項を参照)。
-xr	global application process device transaction ls EMPTY=nnn SPACE=nnn	SIZE=nnn DAYS=nnn YES NO MAYBE	ログ ファイルのサイズを変更します (第 4 章の resize コマンドの項を参照)。
-? または ?			コマンド ライン構文を表示します。

コマンド ライン インターフェイスの使用例

コマンド ラインにコマンド オプションと引数を入力するときには、次の規則が適用されます。

エラーと欠落データは、対応するバッチ モード コマンドとまったく同じように処理されます。すなわち、欠落データには可能な場合デフォルトが使用され、エラーがあるとプログラムは直ちに終了します。

コマンドおよびコマンドの実行結果のエコーは無効に設定されます。Utility はその stdin ファイルから読み込みを行いません。utility はコマンド ラインに指定されている動作を実行してから終了します。

```
utility -xp -d -xs
```

これは、次のように解釈します。

-xp	デフォルトの parm ファイルの構文を確認します。
-d	走査レポートで詳細を無効にします。
-xs	走査を実行します。ログ ファイルが指定されていないため、デフォルトのログ ファイルが走査されます。

utility 走査レポートの詳細

utility プログラムの scan コマンドは、ログ ファイルを読み込み、その内容に関するレポートを作成します。レポートの内容は、scan コマンドの前に発行されたコマンドによって異なります (詳細は、第 5 章「utility のコマンド」の「scan」コマンドの項を参照してください)。

次の表は、すべての走査レポートや、detail on コマンドが scan コマンドと共に使用されたとき (デフォルト) にのみ作成されるレポートに含まれる情報をまとめたものです。

表 4　　走査レポートに含まれる情報

初期値	
初期 parm ファイルのグローバル情報およびシステム構成情報	detail on が指定された場合にのみ出力されます。
初期 parm ファイルのアプリケーション定義	detail on が指定された場合にのみ出力されます。
年代順の詳細	
parm ファイルのグローバル変更内容	detail on が指定された場合にのみ出力されます。
parm ファイルのアプリケーション変更内容	detail on が指定された場合にのみ出力されます。

表 4　　走査レポートに含まれる情報

コレクタのオフタイム通知	detail on が指定された場合にのみ出力されます。
アプリケーション固有の要約レポート	detail on が指定された場合にのみ出力されます。
要約	
プロセス要約レポート	プロセス データが走査された場合、常に出力されます。
コレクタの適用範囲の要約	常に出力されます。
ログ ファイルの内容の要約	常に出力されます。スペースおよび日付を含みます。
ログ ファイルの空きスペースの要約	常に出力されます。

走査レポート情報

utility 走査レポートの情報は、次の 3 つの種類に分類されます。

- 初期値
- 年代順の詳細
- 要約

初期値

この項では、次の初期値について説明します。

- 初期 parm ファイルのグローバル情報
- 初期 parm ファイルのアプリケーション定義

初期 parm ファイルのグローバル情報

このレポートを生成するには、scan コマンドをデフォルトの detail on と共に使用します。

このレポートは、ログ ファイル内で最も古いグローバル レコードの時刻における parm ファイルの構成の設定をリストします。その後のグローバル情報変更通知は、このレポートに含まれる値に基づいて行われます。特定のパラメータについて変更通知が存在しない場合、走査期間中そのパラメータが元の設定値を維持していたことを意味します。

次の例は、parm ファイルの内容をリストするレポートの一部です。

```
06/03/99 15:28 System ID="Homer"
scopeux/UX A.10.00 SAMPLE INTERVAL = 300,300,60 Seconds, Log version=D
Configuration: 9000/855, O/S A.10.00 CPUs=1
Logging Global Process records
        Device= Disk FileSys records
Thresholds: CPU= 10.00%, Disk=10.0/sec, First=5.0 sec, Resp=30.0 sec,
```

```
                Trans=100 Nonew=FALSE, Nokilled=FALSE, Shortlived=FALSE
                (<1 sec)
HP-UX Parms: Buffer Cache Size = 16384KB, NPROC = 532
Wait Thresholds: CPU=100.00%, Memory=100.00%
Impede=100.00%
Memory: Physical = 84.0 MB, Swap = 124304.0 MB, Available to users = 66.5
MB. There are 2 LAN interfaces: 0, 1.
06/03/99 15:28 There are 2 disk devices:
   Disk #1976            = "/dev/hdisk0"
   Disk #1987            = "/dev/hdisk1"
```

最初の行にリストされている日時は、グローバル ログ ファイル内での最初の日時であり、scope の開始時刻を示しています。グローバル ログ ファイルからデータ レコードがロールアウトされている可能性もあるため、このレポートに示されている日時が、必ずしもそのログ ファイルの最初のグローバル レコードであるとは限りません。

初期 parm ファイルのアプリケーション定義

このレポートを生成するには、scan コマンドをデフォルトの detail on と共に使用し、ログ ファイルにアプリケーション データを記録します。

このレポートは、最初のアプリケーション レコードがログ ファイルにリストされた時点の各アプリケーションの名前と定義をリストします。アプリケーションの追加または削除に関する通知は、この初期アプリケーション リストに基づいて行われます。次に例を示します。

```
06/01/99 08:39 Application(1) = "other"
Comment=all processes not in user-defined applications

06/01/99 08:39 Application(2) = "Real_TimeSystem"
 Priority range = 0-127

06/01/99 08:39 Application(3) = "Prog_Development"
 File=vi,ed,sed,xdb,ld,lint,cc,ccom,pc,pascomp
```

走査中に、追加または削除されたアプリケーションが通知されます。追加と削除は、古い名前と新たに記録されたアプリケーション名のスペルと大文字小文字を比較することによって判断されます。アプリケーションの定義の変更は検出されません。新しい名前のアプリケーションが検出された場合、そのアプリケーションが新しい定義と共にリストされます。

このレコードで示される日時は、現在ログ ファイルにある最初のアプリケーション レコードが記録される以前に scope が最後に開始された日時です。

年代順の詳細

この項では、次の年代順の詳細について説明します。

- parm ファイルのグローバル変更通知
- parm ファイルのアプリケーション追加通知および削除通知
- scope のオフタイム通知
- アプリケーション固有の要約レポート

parm ファイルのグローバル変更通知

このレポートを生成するには、scan コマンドをデフォルトの detail on と共に使用します。

このレポートは、scope が開始したことを示すレコードが発見されたときに生成されます。

次の例では、2 つのディスク ドライブがシステムに追加されたときに行われる変更通知を示します。

```
03/13/99 17:30 The number of disk drives changed from 9 to 11
03/13/99 17:30  New disk device scsi-4 = "c4d0s*"
03/13/99 17:30  New disk device scsi-3 = "c3d0s*"
```

parm ファイルのアプリケーション追加通知および削除通知

このレポートを生成するには、scan コマンドをデフォルトの detail on と共に使用し、ログ ファイルにアプリケーション データを記録します。

ユーザー定義のアプリケーションは、scope を開始するたびに追加または削除できます。最後のアプリケーション セットに一致しないアプリケーション名が発見された場合、アプリケーションの追加、削除、変更を示す通知が出力されます。アプリケーション名が変更されていない場合は出力されません。

次の例は、新しいアプリケーションを開始したことを示しています。

```
03/13/99 17:30 Application 4 "Accounting_Users_1" was added
User=ted,rebecca,test*,mark,gene
```

アプリケーションの定義の変更は確認されません。アプリケーションの定義は、アプリケーション名が変更されたときにリストされますが、名前の変更を伴わない既存のアプリケーション定義に対する変更は検出されません。

scope のオフタイム通知

このレポートを生成するには、scan コマンドをデフォルトの detail on と共に使用します。

抽出ファイルに要約情報のみが含まれている場合、時刻の端数が切り捨てられ、最も近い時刻で示されます。次に例を示します。

```
06/03/99 11:00 - 06/03/99 12:34 collector off (01:34:04)
```

最初の日時 (06/03/99 11:00) は、scope を再開始する前のログ ファイルで最後の有効なデータ レコードを示しています。2 番目の日時 (06/03/99 12:34) は、scope を再開始した日時を示しています。

最後のフィールド (括弧内) は、scope を実行していない時間の長さを示します。フォーマットは *ddd*/*hh:mm:ss* です。*ddd* は日数、*hh:mm:ss* は、それぞれ時間、分、秒数です。左側の 0 は削除されます。

この例では、scope が、1999 年 6 月 3 日の 11:00 am から 12:34 pm まで停止していました。要約情報は、データが 1 時間 34 分の間収集されていないことを示しています。

アプリケーション固有の要約レポート

このレポートを生成するには、scan コマンドをデフォルトの detail on と共に使用し、ログ ファイルにアプリケーション データを記録します。

このレポートは、アプリケーションを定義するときに役立ちます。蓄積しているシステム リソースが多すぎるアプリケーションや少なすぎるアプリケーション、他のアプリケーションと統合できるアプリケーションを特定するときには、このレポートを使用してください。蓄積しているシステム リソースが多すぎるアプリケーションを、複数のアプリケーションに分割すると、パフォーマンスが向上します。

アプリケーションの定義は、システム パフォーマンスの調整について判断しやすい方法で行う必要があります。システム リソースがどのアプリケーションにも均等に蓄積することはあまりありません。

アプリケーションの定義が変更されるたびにアプリケーション固有の要約レポートが生成されるため、ユーザーはアプリケーションの定義の変更前と変更後のデータにアクセスできます。

すべてのアプリケーションに関する最終レポートが生成されます。このレポートに含まれる期間は、最後のレポート以降の期間であり、そのログ ファイルに含まれるすべての期間ではありません。次に例を示します。

```
                                  PERCENT OF TOTAL
Application             Records    CPU    DISK   TRANS
------------------- ---------- ------ ------- ------
OTHER                     22385  45.7%  20.9%  63.0%
Resource_Sharing           7531   6.0%   2.2%  17.1%
SPOOLING                  13813   2.4%   0.3%   0.0%
ON_LINE_COMPILES          13119   2.9%   1.7%   0.1%
BATCH_COMPILES             8429   2.9%   0.1%   2.2%
ORDER_ENTRY                 387   0.1%   0.0%   0.0%
ELECTRONIC_MAIL            6251   3.8%   1.3%   9.6%
PROGRAM_DEVELOPMENT        3141   9.1%   2.4%   0.6%
RESEARCH_DEPARMENT         3968   8.7%   2.0%   6.0%
BILL_OF_MATERIALS           336   0.6%   1.5%   0.1%
FINANCIALS                 1080   5.0%   1.5%   0.5%
MARKETING_DEPT             2712  12.9%  67.3%   0.0%
GAMES                       103   0.1%   0.0%   0.0%
------------------- ---------- ------ ------ ------
All user applications       73.1%   54.3%  79.1%  37.0%
```

要約

この項では、次の要約について説明します。

- プロセスの記録理由の要約
- 実際の走査の開始日時および終了日時
- アプリケーションの全体的な要約
- scope の適用範囲の要約
- ログ ファイルの内容の要約
- ログ ファイルの空きスペースの要約

プロセスの記録理由の要約

このレポートを生成するには、ログ ファイルにプロセス データを記録する必要があります。

このレポートは、scope の対象プロセスのしきい値を設定するときに役立ちます。このレポートは、プロセスが対象プロセスとみなされて記録された理由と、各条件を満たして記録されたプロセスの総数をリストします。

次のページの例は、プロセスの記録理由の要約レポートを示しています。

```
Process Summary Report: 04/13/99 3:32 PM to 05/04/99 6:36 PM
There were 93.8 hours of process data
Process records were logged for the following reasons:

Log Reason        Records    Percent    Recs/hr
---------------   -------    -------    -------
New Processes       17619      53.9%       44.7
Killed Processes    16047      49.1%       40.7
CPU Threshold        3169       9.7%        8.0
Disk Threshold       1093       3.3%        2.8
```

注記： プロセスは、一度に複数の理由で記録される場合があります。レコード数と割合が、プロセス レコードの 100% を超えることはありません。

detail on コマンドが発行された場合は、しきい値が変更されるたびにこのレポートが生成されるため、その変更の影響を評価できます。各レポートは、最後のレポート以降の期間をカバーしています。走査終了時に生成される最終レポートは、最後のレポート以降の期間をカバーしています。

detail off コマンドが発行された場合、走査期間全体をカバーしたレポートが 1 つのみ生成されます。

scope で記録するプロセス データの量は、parm ファイルの threshold パラメータを変更し、ほとんどのプロセス ログ レコードを生成する条件となるしきい値を上げると減少します。記録するデータの量を増加するには、対象領域のしきい値を下げてください。

前の例では、CPU しきい値を上げるか、nonew しきい値を設定することで、プロセス データに使用するディスク スペースの量を減らせます (記録される情報量は少なくなります)。

走査の開始および終了

この要約レポートは、有効なデータが走査された場合に出力されます。実際の走査の開始日時および終了日時が示されます。以下に例を示します。

```
Scan started on         03/03/99 12:40 PM
Scan stopped on         03/11/99  1:25 PM
```

アプリケーションの全体的な要約

このレポートを生成するには、ログ ファイルにアプリケーション データを記録する必要があります。

このレポートは、ユーザー定義のアプリケーションで蓄積されたシステム アクティビティの量に関する総合的な指標であり、その他のアプリケーションに関する情報は含まれていません。ユーザー アプリケーションが大量のクリティカル リソースを取り込んでいない場合、ユーザー アプリケーションに含められるプロセスを特定するためにプロセス データの走査を検討する必要があります。

次に例を示します。

```
OVERALL, USER DEFINED APPLICATIONS ACCOUNT FOR
    82534 OUT OF    112355 RECORDS      ( 73.5%)
    218.2 OUT OF     619.4 CPU HOURS    ( 35.2%)
     24.4 OUT OF      31.8 M DISC IOS   ( 76.8%)
      0.2 OUT OF       0.6 M TRANS      ( 27.3%)
```

コレクタの適用範囲の要約

このレポートは、有効なグローバル データまたはアプリケーション データが走査されたときに出力されます。このレポートは、システムの活動状況の収集に scope がどの程度有効に使用されたかを示します。次の例に示されるように、scope の停止時間の割合が高い場合は、scope の開始および終了に関する操作手順を見直す必要があります。

```
The total time covered was          108/16:14:51 out of 128/00:45:02
Time lost when collector was off    19/08:30:11  15.12%
The scopeux collector was started       45 times
```

グローバル詳細データが走査に含まれている場合、このレポートはより完全なものになります。入手できるデータが要約データだけである場合は、scope の開始時刻と終了時刻は直近の正時のみが決定されます (この場合は、適切な警告メッセージがレポートと共に出力されます)。

カバーされた合計時間は、記録データから得られるすべてのインターバルを加算することで決定されます。「out of」時間値は、終了日時から開始日時を差し引くことで算出されます。これは、記録が可能であった合計時間を表しています。「Time lost when collector was off」値は、合計時間からカバーされた時間を差し引いた時間です。

上記の 3 つの時刻を表すフォーマットは次のとおりです。

ddd/hh:mm:ss

ddd は日数、*hh:mm:ss* はそれぞれ時間、分、秒数です。

上記の例では、収集された合計時間は、108 日と 16 時間 14 分 51 秒でした。

ログ ファイルの内容の要約

ログ ファイルの内容の要約は、有効なデータが走査された場合に出力され、ログ ファイルのスペースとカバーされた日付が示されます。この要約は、resize コマンドでログ ファイルのサイズを変更するときに役立ちます。

```
--------Total-------  -----Each Full Day-----  -------Dates--------  Full
Type          Records  MBytes  Records  MBytes  Start          Finish  Days
Global         1376     0.27     288.9   0.057  05/23/99 to 05/28/99  4.8
Application    6931     0.72    1455.0   0.152  05/23/99 to 05/28/99  4.8
Process        7318     1.14    1533.6   0.239  05/23/99 to 05/28/99  4.8
Disk           2748     0.07     567.6   0.014  05/23/99 to 05/28/99  4.8
Transaction  no data found
Overhead                0.29
               -----    ------    ------   -----
TOTAL         18373     2.49    3845.0   0.461
```

各カラムについて、以下に説明します。

カラム	説明
Type	記録されているデータの一般的な型です。Overhead という特殊な型のデータがあります。 Overhead は、ログ ファイルによって占められている (または予約されている) ディスク スペース量と走査されたデータ レコードによって実際に使用されているスペース量との比率です。 ログ ファイル全体が走査されない場合、Overhead には走査されないデータ レコードが含まれます。ファイル全体が走査された場合、Overhead は、ファイル内でデータをブロック化するときの非効率性およびファイル アクセス サポート構造を説明します。抽出ログ ファイルは、位置決めの迅速化を目的とした追加サポート構造を備えているため、通常は生ログ ファイルよりもオーバーヘッドが高くなります。
Total	各データ型について走査された合計レコード数とディスク スペースです。
Each Full Day	scope の実行により 24 時間あたりに使用されるレコードの数とディスク スペースの使用量です。
Dates	走査された各データ型のデータ レコードに関して有効な最初の日付と最後の日付です。
Full Days	このデータ型に関して走査された時間を日数 (24 時間単位) で表した数字です。scope の適用範囲が操作時間の 100 % と一致しない場合、Full Days は開始日から終了日までの日数と等しくなりません。

TOTAL 行 (リストされているデータの最下部) により、ディスク スペースの使用量がわかり、毎日蓄積されるデータ量を予想できます。

ログ ファイルの空きスペースの要約

この要約は、走査された各ログ ファイルについて出力されます。以下に例を示します。

```
The Global      file is now  13.9% full with room for 61 more full days
The Application file is now  15.1% full with room for 56 more full days
The Process     file is now  23.5% full with room for 32 more full days
The Device      file is now   1.4% full with room for 2896 more full days
```

さらに多くのデータを格納するための空き容量は、次の要素に基づいて計算されます。

- ファイル内の未使用スペースの量
- 走査された値 (24 時間あたりの記録データのメガバイト数)

megabytes-scanned-per-day の値 (1 日あたりの走査メガバイト数) が低く、現実的でない場合、この計算にはデフォルト値が使用されます。

抽出ファイルを走査する場合、すべてのデータ型が同じ抽出ファイルを共用するため、レポートに表示されるのは 1 行だけです。

5 utility のコマンド

この章では、utility プログラムのコマンドについて説明します。この章は、構文をまとめた表と、アルファベット順にコマンドを説明しているコマンド リファレンスから構成されています。

utility のコマンドとパラメータは、大文字と小文字を組み合わせて入力できます。コマンド名の最初の 3 文字だけは必ず入力します。たとえば、logfile コマンドは **logfile** と入力するか、**log** または **LOG** と省略して入力できます。

これらのコマンドの使用例については、utility プログラムのオンライン ヘルプを参照してください。

次のページの表では、utility のコマンド構文とパラメータをまとめています。

表 5　　utility のコマンド : 構文とパラメータ

コマンド	パラメータ
analyze	
checkdef	alarmdef ファイル
detail	on off
exit e	
guide	
list	*filename* または *
logfile	logfile
menu ?	
parmfile	parmfile
quit q	

表 5　　utility のコマンド : 構文とパラメータ（続き）

コマンド	パラメータ
resize	global application process device transaction days=maxdays size=max MB empty=days space=MB yes no maybe
scan	*logfile* (list、start、stop、detail の各コマンドも操作に影響します)
show	all
sh !	システム コマンド
start	*date* [*time*] today [-*days*] [*time*] last [-*days*] [*time*] first [+*days*] [*time*]
stop	*date* [*time*] today [-*days*] [*time*] last [-*days*] [*time*] first [+*days*] [*time*]

analyze

analyze コマンドは、アラーム定義 (alarmdef) ファイル内のアラーム定義と比較してログ ファイル内のデータを分析し、結果として生じるアラームのステータスと活動状況をレポートするときに使用します。analyze コマンドを発行する前に、checkdef コマンドを実行してアラーム定義構文を確認する必要があります。また、Checkdef は analyze と共に使用するアラーム定義ファイル名を設定し、保存します。analyze の前に checkdef を実行しないと、アラーム定義ファイル名を入力するように促すプロンプトが表示されます。

コマンド ライン モードを使用している場合は、デフォルトのアラーム定義ファイル /var/opt/perf/alarmdef が使用されます。

アラーム定義の詳細は、151 ページの「パフォーマンス アラーム」を参照してください。

構文

analyze

使い方

analyze コマンドを発行すると、alarmdef ファイルのアラーム定義と比較して、データ ソース構成ファイル datasources で指定されているログ ファイルが分析されます。

analyze コマンドにより、システム上で収集した履歴データとアラーム定義が適合しているかを評価できます。また、アラーム定義が分析用ワークステーション上で非常に多数のアラームを発生させるか、少数のアラームを発生させるかを決定できます。

さらに、条件が満たされると PRINT 文により情報を出力できるため、アラーム定義ファイル内の定義 (IF 文) によりデータ分析を実行できます。アラーム定義での IF 文および PRINT 文の使い方については、第 9 章「パフォーマンス アラーム」を参照してください。

オプションとして、start、stop、detail の各コマンドを analyze と共に実行して、分析プロセスをカスタマイズできます。これらのコマンドは次の順序で指定します。

```
checkdef
start
stop
detail
analyze
```

特定期間に収集されたログ ファイル データを分析する場合は、start コマンドと stop コマンドを使用します (start コマンドと stop コマンドについては、この章で後述します)。

analyze コマンドを実行中に、alarm start、end、repeat status ステータスなどのアラーム イベントと、関連する PRINT 文のテキストがリストされます。また、PRINT 文内のテキストは、IF 文の条件が真になるとリストされます。EXEC 文は実行されませんが、リストされるため、実行内容を参照できます。アラーム要約レポートは、アラームの発生回数と各アラームがアクティブ (オン) であった時間を示します。回数には、alarm starts と repeats が含まれますが、alarm ends は含まれません。

アラーム要約レポートのみを参照する場合は、detail off コマンドを発行します。ただし、コマンド ライン モードを使用している場合は、detail off がデフォルトであるため、アラームのイベントと要約を参照するには、-D を指定する必要があります。

例

checkdef コマンドが、alarmdef ファイル内のアラーム定義構文を確認し、後で analyze コマンドで使用するために alarmdef ファイルの名前を保存します。start today コマンドは、今日記録されたデータのみの分析を指定します。最後に、analyze コマンドが alarmdef ファイルのアラーム定義と比較して、datasources ファイルで指定されているデフォルトの SCOPE データ ソース内のログ ファイルを分析します。

```
utility>
checkdef /var/opt/perf/alarmdef
start today
analyze
```

コマンド ライン引数を使用して上記の操作を実行するには、次のように入力します。

```
utility -xc -D -b today -xa
```

checkdef

checkdef コマンドは、アラーム定義ファイル内のアラーム定義の構文を確認し、発見された警告またはエラーをレポートするときに使用します。また、このコマンドは analyze コマンドと共に使用するアラーム定義ファイル名を設定し、保存します。

アラーム定義構文とアラーム定義の指定方法については、第 9 章「パフォーマンス アラーム」を参照してください。

構文

checkdef [/directorypath/alarmdef]

パラメータ

alarmdef　任意のアラーム定義ファイル名です。ユーザー指定のファイル、またはデフォルトの alarmdef ファイルを指定します。ディレクトリ パスを指定しない場合、現在のディレクトリが検索されます。

使い方

アラーム定義を正しいと判断した場合は、analyze コマンドを使用してログ ファイル内のデータと比較して、アラーム定義を処理できます。

バッチ モードで、アラーム定義ファイルを指定しない場合は、デフォルトの alarmdef ファイルが使用されます。

対話型モードでは、アラーム定義ファイルを指定しない場合、ファイルの指定を促すプロンプトが表示されます。

例

checkdef コマンドが、alarmdef ファイル内のアラーム定義構文を確認し、後で analyze コマンドで使用するために alarmdef ファイルの名前を保存します。

```
utility>
checkdef /var/opt/perf/alarmdef
```

コマンド ライン引数を使用して上記の操作を実行するには、次のように入力します。

utility -xc

detail

detail コマンドは、analyze、parmfile、scan の各レポートに出力する詳細レベルを制御するときに使用します。

デフォルトは、対話型モードおよびバッチ モードでは detail on であり、コマンド ライン モードでは detail off です。

構文

detail　[on]

[off]

パラメータ

on　parm ファイルの有効な内容を parm ファイルのエラーと共に出力します。analyze と scan の完全なレポートを出力します。

off　parm ファイル レポートでは、アプリケーションの定義は出力されません。scan レポートでは、scope の収集時間、初期 parm ファイルのグローバル情報、アプリケーションの定義は出力されません。analyze レポートでは、アラーム イベントとアラーム動作は出力されません。

使い方

analyze、scan、parmfile の各コマンドと共に detail コマンドを使用する方法については、この章の「analyze」、「parmfile」、「scan」のコマンド説明を参照してください。

例

detail コマンドの使用例については、この章の「analyze」、「parmfile」、「scan」のコマンド説明を参照してください。

exit

exit コマンドは、utility プログラムを終了するときに使用します。exit コマンドは、utility プログラムの quit コマンドに相当します。

構文

```
exit
e
```

guide

guide コマンドは、ガイド コマンド モードに移行するときに使用します。ガイド コマンド インターフェイスは utility のさまざまなコマンドについてガイドし、一般的で実行可能ないくつかの作業を行うように促すプロンプトを表示します。

構文

```
guide
```

使い方

- utility の対話型モードからガイド コマンド モードに移行するには、**guide** と入力して **Return** キーを押します。
- パラメータのデフォルト値を適用するには、**Return** キーを押します。
- ガイド コマンド モードを終了し、対話型モードに戻るには、guide> プロンプトで **q** と入力します。

このコマンドでは、パラメータ設定値を組み合わせる必要がありません。このコマンドは、ほとんどのユーザーに有益な結果をもたらす設定値を選択します。

help

help コマンドは、utility プログラムのオンライン ヘルプにアクセスするときに使用します。

構文

help [*keyword*]

使い方

utility のコマンドと作業に関する情報 (ヘルプの情報) を得るには、パラメータを入力します。キーワードを入力すると、別のトピックに移動できます。情報が複数のページにわたる場合は、**Return** を押すと次のページが表示されます。ヘルプ システムを終了し、utility プログラムに戻るには、**q** または **quit** と入力します。

また、特定のトピックに関するヘルプも要求できます。次に例を示します。

```
help tasks
```

または

```
help resize parmfile
```

この形式の help コマンドを使用すると、指定のトピックのヘルプ テキストを受信しますが、utility のコマンド エントリ コンテキストは終了していません。対話型モードでヘルプ サブシステムを指定していないため、次の utility コマンドを入力する前に **quit** を入力する必要はありません。

list

list コマンドは、utility のすべてのレポートの出力ファイルを指定するときに使用します。出力されるレポートの内容は、list コマンドの後に発行される別のコマンドによって異なります。たとえば、list コマンドの後に logfile、detail on、scan の各コマンドを使用すると、ログ ファイルの詳細要約レポートのリスト ファイルが作成されます。

構文

list [*filename*] | *****

***** は stdout に対する出力を設定します。

使い方

レポートのリスト ファイルを指定するには、次の 2 つの方法があります。

- 次のように入力して、utility プログラムを呼び出すときに stdout をリダイレクトします。

  ```
  utility > utilrept
  ```

- 次のように入力して、utility を実行するときに list コマンドを使用します。

  ```
  list utilrept
  ```

どちらの場合も、ユーザーの対話とエラーが stderr に出力され、レポートが指定のファイルに出力されます。

list コマンドの *filename* パラメータは、書き込みアクセス権がある有効なファイル名を表します。既存のファイルがある場合、既存の内容の最後に新しい出力が追加されます。ファイルがない場合は作成されます。

現在の出力ファイルを特定するには、パラメータを使用せずに list コマンドを発行します。

出力ファイルが stdout でない場合、ほとんどのコマンドは入力されたときに出力ファイルにエコーされます。

例

list コマンドは、抽出ログ ファイル rxlog に関する要約レポートを作成します。list utilrept コマンドはディスク ファイルに走査レポート リストを送信します。detail off はレポートの完全な詳細の指定を無効にします。scan コマンドは rxlog を読み取り、レポートを作成します。

list * コマンドはリスト デバイスをデフォルトの stdout に設定します。!lp utilrept はシステム プリンタにディスク ファイルを送信します。

```
utility>
logfile rxlog
list utilrept
detail off
scan
list *
!lp utilrept
```

コマンド ライン引数を使用して上記の操作を実行するには、次のように入力します。

```
utility -l rxlog -f utilrept -d -xs print utilrept
```

logfile

logfile コマンドは、ログ ファイルをオープンするときに使用します。utility プログラムの関数を使用するときは、多くの場合ログ ファイルをオープンする必要があります。これを明示的に行うには logfile コマンドを発行し、暗黙的に行うには別のコマンドを発行します。バッチ モードまたはコマンド ライン モードにあり、ログ ファイル名を指定しない場合は、デフォルトの /var/opt/perf/datafiles/logglob ファイルが使用されます。対話型モードでログ ファイル名を指定しない場合は、ファイル名を指定するか、デフォルトの /var/opt/perf/datafiles/logglob ファイルを適用するように促すプロンプトが表示されます。

構文

logfile [*logfile*]

使い方

生ログ ファイル名または抽出ログ ファイル名を指定できます。抽出ログ ファイル名を指定する場合、すべての情報がこのファイルから得られます。グローバル ログ ファイル logglob 以外の生ログ ファイルを指定する必要はありません。logglob をオープンすると、別のログ ファイルに含まれるすべてのデータにアクセスできます。

生ログ ファイル名は次のとおりです。

logglob	グローバル ログ ファイル
logappl	アプリケーション ログ ファイル
logproc	プロセス ログ ファイル
logdev	デバイス ログ ファイル
logtran	トランザクション ログ ファイル
logls	論理システム ログ ファイル
logindx	索引ログ ファイル

ログ ファイルが正しくオープンされると、ログ ファイル (1 つまたは複数) の内容を示すレポートが印刷または表示されます。次に例を示します。

```
Global      file: /var/opt/perf/datafiles/logglob version D
Application file:/var/opt/perf/datafiles/logappl
Process     file:/var/opt/perf/datafiles/logproc
Device      file:/var/opt/perf/datafiles/logdev
Transaction file: /var/opt/perf/datafiles/logtran
Index       file:/var/opt/perf/datafiles/logindx
System ID:homer
System Type 9000/715 S/N 6667778899 O/S HP-UX B.10.20. A
Data Collector:SCOPE/UX C.02.30
File Created:   06/14/99
Data Covers:    27 days to 7/10/99
Shift is:All Day

Data records available are:
  Global Application Process Disk Volume Transaction

Maximum file sizes:
  Global=10.0 Application=10.0 Process=20.0 Device=10.0 Transaction=10.0 MB
The first GLOBAL      record is on 06/14/99 at 12:00 AM
The first APPLICATION record is on 06/25/99 at 12:00 AM
The first PROCESS     record is on 07/06/99 at 12:01 AM
The first DEVICE      record is on 05/01/99 at 11:50 AM
The first TRANSACTION record is on 05/01/99 at 11:55 AM
The default starting date & time = 05/01/99 11:50 AM   (FIRST +   0)
The default stopping date & time = 07/10/99 11:59 PM   (LAST -    0)
```

show コマンドでオープンしたログ ファイルは、後述のように検証できます。

再度 logfile コマンドを入力すると、随時別のログ ファイルをオープンできます。現在オープンしているログ ファイルは、新しいログ ファイルがオープンされる前にクローズされます。

resize コマンドと scan コマンドを実行するには、ログ ファイルをオープンする必要があります。現在オープンしているログ ファイルがない場合は、暗黙的に logfile コマンドが実行されます。

生ログ ファイル名は変更しないようにしてください。生ログ ファイルへのアクセスは、標準ログ ファイル名が有効であるとみなされて行われます。

複数の生ログ ファイル セットを同じシステムに配置する必要がある場合、各ファイル セットに別々のディレクトリを作成してください。ログ ファイル名は変更できませんが、別々のディレクトリは使用できます。任意の方法でログ ファイルのサイズを変更する場合、すべてのログ ファイルに対する読み取りアクセスおよび書き込みアクセスが必要です。

menu

menu コマンドは、使用可能な utility のコマンドのリストを出力するときに使用します。

構文

menu

例

```
utility> menu
Command  Parameters       Function
HELP     topic]           Get information on commands and options
GUIDE                     Enter guided commands mode for novice users
LOGFILE  [logname]        Specify a log file to be processed
LIST     [filename|*]     Specify the listing file
START    [startdate time] Set starting date & time for SCAN or ANALYZE
STOP     [stopdate  time] Set ending   date & time for SCAN or ANALYZE
DETAIL   [ON|OFF]         Set report detail for SCAN, PARMFILE, or ANALYZE
SHOW     [ALL]            Show the current program settings
PARMFILE [parmfile]        Check parsing of a parameter file
SCAN     [logname]        Read the log file and produce a summary report
RESIZE   [GLOB|APPL|PROC|DEV|TRAN]
         [DAYS=][EMPTY=]  Resize raw log files
CHECKDEF [alarmdef]       Check parsing and set the alarmdef file
ANALYZE                   Analyze the log file using the alarmdef file
! or Sh  [command]        Execute a system command
MENU or ?List the commands menu (This listing)
EXIT or Q                 Terminate the program
utility>
```

parmfile

parmfile コマンドは、データ収集に使用する Performance Collection Component の parm ファイルの設定値を表示し、構文を確認するときに使用します。

構文

parmfile [/*directorypath*/*parmfile*]

使い方

次のいずれかを行うときに、parmfile コマンドを使用します。

- parm ファイルの構文の警告に関する調査と、結果として生じる設定値のレビュー。すべてのパラメータが正しい構文であることが確認され、エラーがレポートされます。構文の確認が完了すると、適用可能な設定値だけがレポートされます
- アプリケーション定義用の空き容量の検索
- detail on を指定したときの parm ファイルの有効な内容、無効にならないデフォルト設定値、アプリケーション定義の出力

バッチ モードでは、parm ファイル名を指定しないと、デフォルトの parm ファイルが使用されます。

対話型モードでは、parm ファイル名を指定しないと、指定するように促すプロンプトが表示されます。

例

parmfile コマンドは、現在の parm ファイルの構文を確認し、警告やエラーをレポートします。detail on は記録しているパラメータの設定をリストします。

```
utility>
detail on
parmfile parm
```

コマンド ライン引数を使用して上記の操作を実行するには、次のように入力します。

```
utility -xp -D
```

quit

quit コマンドは、utility プログラムを終了するときに使用します。quit コマンドは、utility プログラムの exit コマンドに相当します。

構文

quit
q

resize

resize コマンドは、使用している生ログ ファイル セットのスペースを管理するときに使用します。ファイルとファイル内部の制御構造との調整を保存するために生ログ ファイルのサイズを変更するときは、このプログラムを使用する必要があります。別のツールを使用する場合、これらの制御構造の有効性を削除するか、破棄してください。

utility プログラムは、抽出ファイルのサイズを変更するときには使用できません。抽出ファイルのサイズを変更する場合は、extract プログラムを使用し、新しい抽出ログ ファイルを作成します。

構文

```
resize  [global     ] [days=maxdays] [empty=days] [yes  ]
        [application] [size=maxMB  ] [space=MB  ] [no   ]
        [process    ]                             [maybe]
        [device     ]
        [transaction]
```

パラメータ

log file type	サイズを変更する生データ型 (グローバル、アプリケーション、プロセス、デバイス、トランザクションのいずれか) を指定します。生データ型はそれぞれ、生ログ ファイルの logglob、logappl、logproc、logdev、logtran に対応しています。データ型を指定せずに、バッチ モードで utility を実行している場合は、バッチ ジョブが終了します。対話型モードで utility を実行している場合は、現在存在するログ ファイルに基づいてデータ型を指定するように促すプロンプトが表示されます。
days & size	ログ ファイルの最大サイズを指定します。実際のサイズは、ファイル内のデータの量によって異なります。
empty & space	サイズ変更操作の完了後に、ファイルに必要な最小空き容量を指定します。この値は、ログ ファイル内に現在存在するデータをサイズ変更プロセスで削除する必要があるかを決定するときに使用します。

サイズ変更の操作後に、ログ ファイルは指定した日数までいっぱいにならないと予測できます。サイズ変更はログ ファイルがいっぱいになると行われるため、resize コマンドのこの機能を使用すると、scope コレクタによるログ ファイルのサイズ変更の回数を最小にできます。resize を使用して、ログ ファイル内に強制的に一定の空き容量を確保すると、必要なときにログ ファイルのサイズを変更できます。

days および empty には日数で値を入力し、size および space にはメガバイト単位で値を入力します。日数は、ログ ファイルの 1 日あたりの平均メガバイト数を使用して、メガバイト単位に変換されます。この変換要素は、記録するデータ型やシステムの特定の特徴によって異なります。

resize コマンドを発行する前に、既存のログ ファイルに対して scan コマンドを発行すると、より正確な 1 日あたりの平均メガバイト数変換要素が得られます。scan は、システムの累計率を測定します。scan を実行しない場合、または測定した変換要素が妥当でない場合、resize コマンドは各データ型のデフォルトの変換要素を使用します。

yes　サイズ変更を無条件で実行することを指定します。utility を対話型モード以外で実行している場合は、これがデフォルトの動作になります。対話型モードで utility を実行し、動作を指定しない場合は、動作の指定を促すプロンプトが表示されます。

no　サイズ変更を実行しないことを指定します。このパラメータは、サイズ変更レポートを表示し、そのときにサイズ変更を実行しない場合の動作として指定できます。

maybe　ファイルのサイズ変更を実行するか実行しないかを utility により決定することを指定します。このパラメータは、ログ ファイル内の現在の空き容量 (サイズ変更前) と、resize コマンドで指定した容量に基づいて utility がこの決定を行うように強制します。現在のログ ファイルの空き容量が少なくとも指定した容量と同じだけある場合、サイズ変更は*行われません*。現在のログ ファイルの空き容量が指定した容量より小さい場合は、サイズ変更が行われます。

maybe　ログ ファイルからデータを削除せずにサイズ変更が可能な場合 (既存のデータを削除せずに最大ログ ファイル サイズを増減する場合など)、サイズ変更が行われます。
maybe パラメータは、主に定期的なバッチ操作で使用するために用意されています。このように resize コマンドを使用する方法については、後述の「例」を参照してください。

デフォルトのサイズ変更パラメータを、次の表に示します。

表 6　　デフォルトのサイズ変更パラメータ

パラメータ	対話型モードで実行した場合	バッチで実行した場合
log file type	有効なログ ファイル型を入力するように促すプロンプトが表示されます。	デフォルトなし。これは必須パラメータです。
days size	現在のファイル サイズ。	現在のファイル サイズ。
空き容量	現在の空き容量、または現在ファイルにあるすべてのデータを保存するために必要な空き容量のうちどちらか小さい方。	現在の空き容量、または現在ファイルにあるすべてのデータを保存するために必要な空き容量のうちどちらか小さい方。

表 6　　デフォルトのサイズ変更パラメータ (続き)

パラメータ	対話型モードで実行した場合	バッチで実行した場合
yes no maybe	レポートされたディスク スペース結果に従うことを促すプロンプトが表示されます。	Yes で、サイズ変更が行われます。

使い方

ログ ファイルのサイズを変更する前に、第 2 章の 39 ページの「データ収集の終了と再開始」に記載されている手順で、Performance Collection Component を終了する必要があります。

生ログ ファイルはサイズ変更を行う前にオープンする必要があります。resize コマンドを発行する前に、logfile コマンドで生ログ ファイルをオープンします。ファイルは、別のプロセスではオープンできません。

resize コマンドは、元のログ ファイルを削除する前に、TMPDIR 環境変数で設定されているディレクトリに新しいファイル scopelog を作成します。環境変数 TMPDIR が設定されていない場合、一時的な場所として /var/tmp ディレクトリ (IBM AIX 4.1 以降の場合、/tmp) が使用されます。サイズ変更の手順を行う前に、元のログ ファイルを保持するために、TMPDIR で指定したディレクトリまたは /var/tmp ディレクトリ (IBM AIX 4.1 以降の場合、/tmp ディレクトリ) に十分なディスク スペースがあることを確認してください。

サイズ変更後は、ログ ファイルはデータと空きスペースで構成されます。保存されたデータは、最大ファイル サイズから必要な空きスペースを引いた大きさとして計算されます。サイズ変更操作中に削除されたデータは失われます。ログ ファイル データを長期間保存するには、extract を使用して、サイズ変更操作を行う前にこのデータを抽出ファイルにコピーします。

resize コマンドのレポート

生ログ ファイルのサイズを変更すると、標準レポートが作成されます。このレポートは、相互に関係する 3 つのディスク スペース カテゴリ (最大ファイル サイズ、データ レコード、空きスペース) について、サイズ変更の前後の値を示します。以下に例を示します。

```
resize global days=120;empty=10
empty space raised to match file size and data records
final resizing parameters:
file: logglob                          megabytes / day: 0.101199
                 ---currently-----     --after resizing---
maximum size:65 days (  6.6 mb)    120 days ( 12.1 mb)  83% increase
data records:61 days (  6.2 mb)     61 days (  6.2 mb)  no data removed
empty space:4 days (  0.5 mb)     59 days (  6.0 mb)  1225% increase
```

1 日あたりのメガバイト数は、日数とメガバイトの変換に使用されます。これは、走査関数で得られる値またはサイズ変更するデータ型のデフォルトです。

右端のカラムは、ログ ファイル スペースの各カテゴリについてのネット変更の要約です。最大サイズと空きスペースは増加や削減が可能で、変更されないこともあります。サイズ変更中に、データ レコードのデータはいずれのデータも削除されない場合、または指定した量のデータが削除される場合があります。

対話形式でサイズ変更が行われ、1 つ以上のパラメータがデフォルトである場合、予備のサイズ変更レポートを作成できます。このレポートは、現在のログ ファイルの内容と指定されたパラメータを要約し、未指定のパラメータに関する問題を解決するために提示されます。次に例を示します。

```
resize global days=20
file resizing parameters (based on average daily
space estimates and user resizing parameters)
file: logglob                          megabytes / day: 0.101199
                -----currently----    --after resizing--
maximum size:65 days (  6.6 mb)    20 days (  2.0 mb)
data records:61 days (  6.2 mb)    ??
empty space:4 days (  0.5 mb)    ??
```

この例では、最終的なサイズ変更レポートを作成する前に、ファイルの空き容量を指定するように促すプロンプトが表示されます。対話形式のサイズ変更に対して動作パラメータを指定しない場合、最終的なサイズ変更レポートに続いて、ログ ファイルのサイズ変更を指定するように促すプロンプトが表示されます。

例

次のコマンドは、生プロセス ログ ファイルのサイズを変更するときに使用します。日数の計算の精度を向上させるため、サイズ変更の前に走査を実行します。

```
logfile /var/opt/perf/datafiles/logglob
detail off
scan
resize process days=60 empty=30 yes
```

days=60 は、最大 60 日分のデータを保持することを指定します。empty=30 はこのファイルの 30 日分のスペースが現在空きスペースであることを指定します。つまり、合計 60 日分のスペースに 30 日分のデータがある状態で、あと 30 日分の空きスペースを残すために、ファイルのサイズ変更が行われます。yes は、現在の空きスペースにかかわらず、サイズ変更を行うことを指定します。

次の例は、バッチ モードで resize コマンドを使用して、ログ ファイルがこれから 1 週間はいっぱいにならないようにする方法を示します (強制的に scope によりファイルのサイズを変更します)。at コマンドで 7 日分、または多めに 10 日分などと最小容量を指定して、cron スクリプトをスケジュールできます。

次のシェル スクリプトは、以下の操作を行います。

```
echo detail off                          > utilin
echo scan                                >> utilin
echo resize global       empty=10 maybe >> utilin
echo resize application  empty=10 maybe >> utilin
echo resize process      empty=10 maybe >> utilin
echo resize device       empty=10 maybe >> utilin
echo quit                                >> utilin
utility < utilin > utilout 2> utilerr
```

10 日分以上の空きスペースが現在ログ ファイルに存在する場合、yes の代わりに maybe を指定すると、サイズ変更操作が行われません。デフォルトでは、最大ファイル サイズが各ファイルの現在の最大ファイル サイズになることに注意してください。この指定により、このスクリプトに影響を与えずにファイルのサイズを新しい最大サイズに変更できます。

上記のスクリプトを使用する場合、このスクリプトを実行する前に、scope を終了する必要があります。scope の終了と開始については、『*HP Operations エージェント インストール、構成ガイド*』の 「Performance Collection Component の起動方法」の章を参照してください。

scan

scan コマンドは、ログ ファイルを読み取り、その内容に関するレポートを作成するときに使用します (レポートの詳細は、3 章の 59 ページの「utility 走査レポートの詳細」を参照してください)。

構文

scan

使い方

scan コマンドを実行するには、ログ ファイルをオープンする必要があります。走査するログ ファイルは、次のファイルのうち最初に指定されたものです。

- scan コマンドで指定したログ ファイル
- 前のコマンドで最後にオープンしたログ ファイル
- デフォルトのログ ファイル

  この場合、対話型モードでは、必要に応じてデフォルト ログ ファイル名を無効にするように促すプロンプトが表示されます。

次のコマンドが走査関数の操作に影響します。

detail	レポートの詳細レベルを指定します。デフォルトの detail on は完全な詳細を指定します。
list	出力を別のファイルにリダイレクトします。デフォルトでは、標準のリスト デバイスにリストします。
start	走査する最初のログ ファイル レコードの日時を指定します。デフォルトは、ログ ファイルの開始日時です。
stop	走査する最後のログ ファイル レコードの日時を指定します。デフォルトは、ログ ファイルの終了日時です。

detail、list、start、stop の各コマンドの詳細は、この章のそれぞれの説明を参照してください。

scan コマンドのレポートは 12 のセクションから構成されます。scan を発行する前に detail コマンドを発行すると、レポートに入れるセクションを制御できます。

次の 4 つのセクションは、(detail off を指定している場合も) 必ず出力されます。

- 実際の走査の開始日時および終了日時
- コレクタの適用範囲の要約
- ログ ファイルの内容の要約
- ログ ファイルの空きスペースの要約

detail on (デフォルト) を指定している場合は、次のセクションが出力されます。

- 初期 parm ファイルのグローバル情報およびシステム構成情報
- 初期 parm ファイルのアプリケーション定義
- parm ファイルのグローバル変更内容
- parm ファイルのアプリケーション追加通知および削除通知
- コレクタのオフタイム通知
- アプリケーション固有の要約レポート

アプリケーション データが走査された場合は、(detail off を指定している場合も) 次のセクションが必ず出力されます。

- アプリケーションの全体的な要約

プロセス データが走査された場合は、(detail off を指定している場合も) 次のセクションが必ず出力されます。

- プロセスの記録理由の要約

例

現在のデフォルト グローバル ログ ファイルの走査を開始します。対象は、1999 年 6 月 1 日 7:00 AM から現在の日時までに記録されたレコードです。

```
utility>
logfile /var/opt/perf/datafiles/logglob
detail on
start 6/1/99 7:00 am
scan
```

コマンド ライン引数を使用して上記の操作を実行するには、次のように入力します。

```
utility -D -b 6/1/99 7:00 am -xs
```

sh

sh は、utility を終了せずにシェル コマンドを入力するときに使用します。**sh** またはエクスクラメーション マーク (**!**) の後にシェル コマンドを指定します。

構文

sh または **![shell command]**

パラメータ

sh ls	ls コマンドを実行し、utility に戻ります。
!ls	上記と同じです。

使い方

1 つのコマンドを実行すると、自動的に utility に戻ります。複数のシェル コマンドを発行するときに、1 つのコマンドを実行するたびに utility に戻らないようにする場合は、次のように新しいシェルを開始します。次に例を示します。

```
sh ksh
```

または

```
!ksh
```

show

show コマンドは、オープンしたファイルの名前と設定可能な utility パラメータのステータスのリストを作成するときに使用します。

構文

```
show [all]
```

例

show を使用すると、次のようなリストが作成されます。

```
Logfile:/var/opt/perf/datafiles/logglob
List:"stdout"
Detail:ON for ANALYZE, PARMFILE and SCAN functions
The default starting date & time = 10/08/99 08:17 AM (FIRST + 0)
The default stopping date & time = 11/20/99 11:59 PM (LAST  - 0)
The default shift = 12:00 AM - 12:00 AM
```

デフォルトのシフト時間が情報として示されます。utility ではシフト時間は変更できません。

show all を使用すると、次の例のようにより詳細なリストが作成されます。

```
Logfile:/var/opt/perf/datafiles/logglob
    Global      file:/var/opt/perf/datafiles/logglob
    Application file:/var/opt/perf/datafiles/logappl
    Process     file:/var/opt/perf/datafiles/logproc
    Device      file:/var/opt/perf/datafiles/logdev
    Transaction file: /var/opt/perf/datafiles/logtran
    Index       file:/var/opt/perf/datafiles/logindx
    System ID:homer
    System Type 9000/715 S/N 66677789   OS/ HP-UX B.10.20 A
    Data Collector:SCOPE/UX C.02.30
    File created:   10/08/99
    Data Covers:44 days to 11/20/99
```

```
    Shift is:All Day
Data records available are:
  Global Application Process Disk Volume Transaction
Maximum file sizes:
  Global=10.0 Application=10.0 Process=20.0 Device=10.0 Transaction 10.0
MB
List    "stdout"
Detail  ON for ANALYZE, PARMFILE and SCAN functions
The default starting date & time = 10/08/99 11:50 AM (FIRST + 0)
The default stopping date & time = 11/20/99 11:59 PM (LAST  - 0)
The default shift = 12:00 AM - 12:00 AM
```

start

start コマンドは、走査または分析するログ ファイルのサブセットの開始日時を指定するときに使用します。start により、特定の日時に記録されたデータから scan プロセスまたは analyze プロセスを開始します。

デフォルトの開始日時は、logfile コマンドで現在オープンしているログ ファイルに含まれる最初のレコード (任意の型) の日時に設定されています。

構文

	[date	*[time]]*	
start	**[today**	*[-days]*	*[time]]*
	[last	*[-days]*	*[time]]*
	[first	*[+days]*	*[time]]*

パラメータ

date　日付フォーマットは、使用しているシステムに構成されている母国語によって異なります。母国語を使用しない場合やデフォルトの言語を C に設定している場合、日付フォーマットは *mm/dd/yy* (月 / 日 / 年) になります。たとえば、1999 年 6 月 30 日は 06/30/99 になります。

time　時刻フォーマットも、使用している母国語によって異なります。C の場合、フォーマットは *hh:mm AM* または *hh:mm PM* (12 時間制の時 : 分の後に AM または PM を付ける形式) になります。たとえば、午前 7 時は 07:00 AM になります。24 時間制の時刻もすべての言語で使用できます。たとえば、11:30 PM を 23:30 とすることもできます。
日付または時刻を不適切なフォーマットで入力すると、正しいフォーマットの例が表示されます。
開始時刻を指定しないと、午前 0 時 (12:00 AM) とみなされます。指定日の午前 0 時が開始時刻の場合は、その日の始まり (24 時間制の場合 00:00) に開始されます。

today　現在の日付を指定します。today-*days* パラメータは、現在の日付からさかのぼる日数を指定します。たとえば、today-1 は前日を示し、today-2 は前前日を示します。

last　ログ ファイルに含まれる最終日を表すときに使用します。last-*days* パラメータは、ログ ファイル内の最終日からさかのぼる日数を指定します。

first　ログ ファイルに含まれる最初の日を表すときに使用します。first+*days* パラメータは、ログ ファイル内の最初の日から経過した日数を指定します。

使い方

start コマンドは、非常に大きなログ ファイルに対して、ファイル全体の走査や分析を行わないときに役立ちます。また、このコマンドを使用して走査するデータの期間を指定できます。たとえば、**today-14** と指定すると、現在の日付の 14 日前から記録されたデータのログ ファイルを走査できます。

stop コマンドを使用することで、走査するログ ファイル レコードをさらに制限できます。

母国語のサポートがシステムにインストールされているかわからない場合は、utility を実行する前に次の文を発行して、次のように utility が C の日時フォーマットを使用するように指定できます。

```
LANG=C; export LANG
```

C シェルでは、次の文を実行します。

```
setenv LANG C
```

例

デフォルト グローバル ログ ファイルの走査を開始します。対象は、1999 年 8 月 5 日 8:00 AM から現在の日時までに記録されたレコードです。

```
utility>
logfile /var/opt/perf/datafiles/logglob
detail on
start 8/5/99 8:00 AM
scan
```

コマンド ライン引数を使用して上記の操作を実行するには、次のように入力します。

```
utility -D -b 8/5/99 8:00 AM-xs
```

stop

stop コマンドは、走査または分析するログ ファイルのサブセットの終了日時を指定するときに使用します。stop により、特定の日時に記録されたデータで走査プロセスまたは分析プロセスを終了できます。

デフォルトの終了日時は、logfile コマンドで現在オープンしているログ ファイルに含まれる最後のレコード (任意の型) の日時に設定されています。

構文

```
                [date       [time]]
stop            [today      [-days]     [time]]
                [last       [-days]     [time]]
                [first      [+days]     [time]]
```

パラメータ

date　日付フォーマットは、使用しているシステムに構成されている母国語によって異なります。母国語を使用しない場合やデフォルトの言語を C に設定している場合、日付フォーマットは *mm*/*dd*/*yy* (月 / 日 / 年) になります。たとえば、1999 年 6 月 30 日は 06/30/99 になります。

time　時刻フォーマットも、使用している母国語によって異なります。C の場合、フォーマットは *hh:mm AM* または *hh:mm PM* (12 時間制の時 : 分の後に AM または PM を付ける形式) になります。たとえば、午前 7 時は 07:00 AM になります。24 時間制の時刻もすべての言語で使用できます。たとえば、11:30 PM を 23:30 とすることもできます。
日付または時刻を不適切なフォーマットで入力すると、正しいフォーマットの例が表示されます。
終了時刻を指定しないと、午前 0 時の 1 分前 (11:59 PM) とみなされます。指定日の午前 0 時 (12:00 AM) が終了時刻の場合、その前日の終わり (24 時間制の場合 23:59) に終了します。

today　現在の日付を指定します。today-*days* パラメータは、現在の日付からさかのぼる日数を指定します。たとえば、today-1 は前日を示し、today-2 は前前日を示します。

last　ログ ファイルに含まれる最終日を表すときに使用します。last-*days* パラメータは、ログ ファイル内の最終日からさかのぼる日数を指定します。

first　ログ ファイルに含まれる最初の日を表すときに使用します。first+*days* パラメータは、ログ ファイル内の最初の日から経過した日数を指定します。

使い方

stop コマンドは、非常に大きなログ ファイルに対して、ファイル全体の走査を行わないときに役立ちます。また、このコマンドを使用して走査するデータの期間を指定できます。たとえば、現在の日付の 1 か月前から記録された 7 日分のデータのログ ファイルを走査できます。

母国語のサポートがシステムにインストールされているかわからない場合は、utility を実行する前に次の文を発行して、次のように utility が C の日時フォーマットを使用するように指定できます。

```
LANG=C; export LANG
```

C シェルでは、次の文を実行します。

```
setenv LANG C
```

例

1999 年 7 月 5 日 8:00 AM から記録されたレコードについて、14 日分のデータの走査を開始し、1999 年 7 月 18 日 11:59 PM に記録された最後のレコードで終了します。

```
utility>
logfile /var/opt/perf/datafiles/logglob
detail on
start 7/5/99 8:00 AM
stop 7/18/99 11:59 PM
scan
```

コマンド ライン引数を使用して上記の操作を実行するには、次のように入力します。

```
utility -D -b 7/5/99 8:00 am -e 7/18/99 11:59pm -xs
```

6 extract プログラムの使い方

extract プログラムには 2 つの主要な関数が含まれており、生ログ ファイルからデータを抽出して、抽出ログ ファイルに書き込むことができます。extract では、表計算ソフトなどの分析用製品で使用できるように、ログ ファイル データをエクスポートすることもできます。

初めて Performance Collection Component をインストールした場合は、extract を機能させるには、ファイルのインストールを完了するために各サービスを起動する必要があります。

抽出関数およびエクスポート関数は、ログ ファイルからデータをコピーします。すなわち、データは削除されません。

Performance Collection Component では、次の 3 種類のログ ファイルを使用します。

- scope ログ ファイル ― scope コレクタにより Performance Collection Component 内で収集されるデータが含まれます。
- 抽出ログ ファイル ― scope 生ログ ファイルから抽出されるデータが含まれます。
- データ ソース統合 (DSI: data source integration) ログ ファイル ― アプリケーションやデータベースなどの外部ソースにより収集されるユーザー定義のデータが含まれます。このデータは、後で Performance Collection Component の DSI プログラムにより記録されます。

extract プログラムを使用すると、次の作業を行うことができます。

- scope 生ログ ファイルからデータのサブセットを抽出し、抽出ログ ファイル フォーマットにします。抽出ログ ファイル フォーマットは、アーカイブ、システム間の転送、Performance Manager での分析に適したフォーマットです。データは、DSI ログ ファイルからは抽出できません。
- 抽出フォーマット ファイルからデータを抽出またはエクスポートし、そのデータを既存の抽出ログ ファイルに追加し、データ型や日付、シフト (時刻) によって細分することで、アーカイブ ログ ファイル データを管理します。
- 生 scope ログ ファイルまたは抽出 scope ログ ファイルおよび DSI ログ ファイルからデータをエクスポートし、レポートや分析、表計算ソフトや同様の分析ソフトへのインポートに適した ASCII、binary、datafile、WK1 (表計算シート) のいずれかのフォーマットにします。

- 抽出関数では要約データは作成できません。要約データはエクスポート関数でのみ作成することができます。
- 抽出関数またはエクスポート関数のは以下のように制限されます。
  - ― 抽出関数からの出力ファイルは、最大 3.5 ギガバイトです。
  - ― エクスポート関数からの出力ファイルは、最大 4 ギガバイトです。

実行される多様な作業例と extract コマンドの使用例については、extract プログラムのオンライン ヘルプを参照してください。

この章では次の内容について説明します。

- extract プログラムの実行
- 対話型モードの使い方
- extract のコマンド ライン インターフェイス
- エクスポート関数の概要

extract プログラムの実行

extract プログラムを実行するには、次の 3 つの方法があります。

コマンド ライン モード

extract プログラムをコマンド ラインでコマンド オプションと引数を使用して制御します。

対話型モード

stdin が対話式の端末またはワークステーションに設定されたプログラムを実行するときに、対話型のコマンドとパラメータを入力します。
使用経験があるユーザーの場合は、特定の作業に必要なコマンドだけを直ちに指定してもかまいません。初めて使用するユーザーの場合は、extract の使い方に関するヘルプが表示されるガイド モードを指定することをお勧めします。ガイド モードでは、作業を行うときに、オプション リストからの選択を促すプロンプトが表示されます。また、ガイド モードでは、各作業を行う対話型のコマンドが実行時にリストされるため、それらのコマンドの使い方を確認できます。ガイド モードは、随時終了および再開始できます。

バッチ モード

プログラムを実行し、対話型のコマンドとパラメータを含むファイルに stdin をリダイレクトできます。

構文

コマンド ライン インターフェイスの構文は、その他のプログラムで使用される標準的な UNIX コマンド ライン インターフェイスと同様であり、この章で詳細を説明します。

対話型モードとバッチ モードのコマンド構文は同一です。つまり、コマンドの後に 1 つ以上のパラメータを指定します。コマンドは、どのような順序で入力してもかまいません。コマンドに関連付けるパラメータがある場合、そのパラメータは対応するコマンドの直後に入力する必要があります。

パラメータには、必須のパラメータ (デフォルトなし) とオプションのパラメータ (デフォルトあり) の 2 種類があります。extract プログラムがこれらのパラメータを扱う方法は、extract プログラムの実行モードによって異なります。

- 対話型モードでオプションのパラメータが欠落している場合は、プログラムがデフォルトのパラメータを表示します。ユーザーはその引数を確定するか、無効にする必要があります。必須のパラメータが欠落している場合、プログラムがパラメータを入力するように促すプロンプトを表示します。

- バッチ モードでオプションのパラメータが欠落している場合、プログラムはデフォルト値を使用します。
  必須のパラメータが欠落している場合、プログラムは終了します。

対話型モードの場合とコマンド ラインおよびバッチ モードの場合では、エラーと欠落データが異なる方法で処理されます。対話型モードではデータの追加や間違いの訂正ができますが、コマンド ラインやバッチ モードではできません。

対話型モードの使い方

extract プログラムの対話型モードを使用するには、特定の作業を行うための一連のコマンドを発行する必要があります。

たとえば、2003 年の 5 月 15 日から収集されたアプリケーション データをデフォルトのグローバル ログ ファイルからエクスポートする場合、extract プログラムを呼び出した後に次のコマンドを発行します。

```
logfile /var/opt/perf/datafiles/logglob
application detail
start 5/15/2003
export
```

logfile コマンドがデフォルトのグローバル ログ ファイル /var/opt/perf/datafiles/logglob をオープンします。start コマンドが 2003 年 5 月 15 日以降に記録されたデータだけをエクスポートすることを指定します。export コマンドがデータのエクスポートを開始します。

extract のコマンド ライン インターフェイス

対話型モードとバッチ モードのコマンド構文のほかに、コマンド ライン インターフェイスを使用してもコマンド オプションと引数を extract プログラムに渡すことができます。extract プログラムをシェル スクリプトで簡単に呼び出せるようにし、UNIX のパイプへの入出力のリダイレクトを可能にすることで、コマンド ライン インターフェイスは一般的な UNIX 環境に適合します。

たとえば、コマンド ラインを使用して、前述の 91 ページの「対話型モードの使い方」に示されている例と同様の処理をするには、次のように入力します。

```
extract -l -a -b 5/15/02 -xp
```

コマンド ライン モードでは、グローバル ログ ファイル /var/opt/perf/datafiles/logglob がデフォルトです。このファイルを指定する必要はありません。

次の表で、コマンド ラインのオプションと引数を示します。それぞれのコマンドの説明は、第 7 章「extract のコマンド」に記載されています。

表 7　　コマンド ラインの引数

コマンド オプション	引数		説明
-b	date	time	抽出関数やエクスポート関数の開始日時を指定します (第 6 章の start コマンドの項を参照)。
-B		UNIX *start time*	抽出関数やエクスポート関数の UNIX フォーマットでの開始時刻を指定します。
-e	date	time	抽出関数やエクスポート関数の終了日時を指定します (第 6 章の stop コマンドの項を参照)。
-E		UNIX *stop time*	抽出関数やエクスポート関数の UNIX フォーマットでの終了時刻を指定します。
-s	time-time	noweekends	週末を除く特定期間の開始時刻と終了時刻を指定します (第 6 章の shift コマンドの項を参照)。
-l	logfile		入力ログ ファイルを指定します (第 6 章の logfile コマンドの項を参照)。デフォルトは /var/opt/perf/datafiles /logglob です。
-r	export template file		エクスポート関数で使用するエクスポート テンプレート ファイルを指定します (第 6 章の report コマンドの項を参照)。
-C	classname	opt	抽出またはエクスポートする scope データ、またはエクスポートする自己記述 (DSI) データを指定します (第 6 章の class コマンドの項を参照)。 opt = detail （**デフォルト**） summary both off
-i			論理システムから抽出またはエクスポートする scope データを指定します
-k			抹消されたプロセスのみを抽出します。このオプションを使用する場合は、reptfile に PROC_INTEREST メトリックを加えます。

表 7　　コマンド ラインの引数（続き）

コマンド オプション	引数		説明
gapkcdzntuy iGADZNTUYI			次のように、抽出またはエクスポートするデータ型を指定します。 g = グローバル詳細 (第 6 章の global コマンドの項を参照)。デフォルトではグローバル詳細はオフです。 a = アプリケーション詳細 (第 6 章の application コマンドの項を参照)。 p = プロセス詳細 (第 6 章の process コマンドの項を参照)。 k = プロセス抹消 (第 6 章の process コマンドの項を参照)。 c = 構成詳細 (第 6 章の configuration コマンドの項を参照)。 d = ディスク デバイス詳細 (第 6 章 disk コマンドの項を参照)。 z = lvolume 詳細 (第 6 章の lvolume コマンドの項を参照)。 n = netif 詳細 (第 6 章の netif コマンドの項を参照)。

表 7　　コマンド ラインの引数（続き）

コマンド オプション	引数		説明
gapkcdzntuy iGADZNTUYI			t = トランザクション詳細 u = CPU 詳細 y = ファイルシステム詳細 i = 論理システム詳細 **注記 :** 以下の要約オプションはエクスポート専用です。抽出関数はデータの要約をサポートしていません。 G = グローバル要約 デフォルトでは、グローバル要約はオフです。 A = アプリケーション要約 D = ディスク デバイス要約 (第 6 章の disk コマンドの項を参照)。 Z = lvolume 要約 (第 6 章の lvolume コマンドの項を参照)。 N = netif 要約 (第 6 章の netif コマンドの項を参照)。 i = 論理システム要約
gapkcdzntuy GADZNTUY （続き）			T = トランザクション要約 U = CPU 要約 (第 6 章の cpu コマンドの項を参照)。 Y = ファイルシステム要約 (第 6 章の filesystem コマンドの項を参照)。
-v			冗長出力レポート フォーマットを生成します。
-f	filename	,new ,append ,purge	抽出データまたはエクスポート データをファイルに送信します。filename が指定されていない場合は、データをデフォルトの出力ファイルに送信します (第 6 章の output コマンドの項を参照)。
-ut			エクスポートされた DSI ログ ファイル データの日付と時刻を UNIX フォーマットで表示します。

表 7　　コマンド ラインの引数（続き）

コマンド オプション	引数		説明
-we	1.....7		エクスポートから除外する曜日を指定します。1 は日曜日です （weekdays コマンドの項を参照）。
-xp	xopt		データを外部フォーマット ファイルにエクスポートします （第 6 章の export コマンドの項を参照）。
-xt	xopt		データをシステム内部フォーマットに抽出します（第 6 章の extract コマンドの項を参照）。 xopt = *dwmy*（日、週、月、年） *dwmy*-[オフセット値] *dwmy* [絶対値]
-xw	week		月曜日から日曜日までの 1 週間のデータを抽出します（第 6 章の weekly コマンドの項を参照）。
-xm	month		暦月のデータを抽出します（第 6 章の monthly コマンドの項を参照）。
-xy	year		暦年のデータを抽出します（第 6 章の monthly コマンドの項を参照）。
-? または ?			コマンド ライン構文を表示します。

コマンド ラインの引数を評価するときや、コマンド ラインにコマンド オプションを入力するときには、次の規則が適用されます。

- エラーと欠落データは、対応するバッチ モード コマンドとまったく同じように処理されます。すなわち、欠落データには可能な場合デフォルトが使用され、エラーがあるとプログラムは直ちに終了します。
- 引数 -v を使用して冗長モードを有効にしていない場合、コマンドおよびコマンドの実行結果のエコーは無効に設定されます。
- 有効な動作を指定していない場合 (-xp、-xw、-xm、-xy、-xt のいずれか)、すべてのパラメータが処理された後、extract ステータスはその stdin ファイルからコマンドの読み込みを開始します。
- 動作を指定している場合 (-xp、-xw、-xm、-xy、-xt のいずれか)、その他のすべてのパラメータが評価された後、プログラムはコマンド オプションを実行します。この場合、パラメータがリストのどの部分に配置されているかは関係ありません。
- コマンド ラインで動作を指定した場合、extract プログラムはその stdin ファイルから読み込みを行いません。代わりに、次の動作に続いてプログラムが終了します。

  ```
  extract -f rxdata -r /var/opt/perf/rept1 -xp d-1 -G
  ```

これは、次のように解釈します。

-f rxdata	現在のディレクトリの指定された rxdata ファイルに出力します。
-r rept1	/var/opt/perf/rept1 ファイルには対象となるエクスポート フォーマットが含まれます。
-xp d-1	現在の日付から 1 日引いた日 (前日) のデータをエクスポートします。
-G	グローバル要約データをエクスポートします。

実際のエクスポートは最後に行われるため、-G パラメータはエクスポートが行われる前に処理されることに注意してください。

ログ ファイルが指定されていないため、デフォルトの logglob ファイルを使用していることに注意してください。

動作が指定されているため (-xp)、エクスポートが終了すると、extract プログラムはその stdin ファイルから読み込みを行わずに終了します。さらに、冗長モードが -v コマンド オプションで設定されていないため、stdout に対して無関係なすべての出力が削除されます。

エクスポート関数の概要

extract プログラムの export コマンドは Performance Collection Component の生ログ ファイル、抽出ログ ファイル、DSI ログ ファイルの各データを、エクスポート ファイルに変換します。export コマンドは、ASCII、データファイル、バイナリ、WK1 (表計算シート) のうちいずれかのフォーマットでファイルを書き込みます。エクスポート ファイルは、レポート、カスタム グラフィックス ソフト、データベース、ユーザーが記述した分析プログラムなど、さまざまな方法で使用できます。

データのエクスポート方法

データをエクスポートするには、次の方法が最も簡単です。

- データをエクスポートするデフォルトのグローバル ログ ファイル /var/opt/perf/datafiles/logglob を指定します。
- エクスポート データのフォーマットを定義するデフォルトのエクスポート テンプレート ファイル /var/opt/perf/reptfile を指定します。
- エクスポート関数を開始します。

エクスポート データは、現在のディレクトリにあるデフォルトの出力ファイル xfrdGLOBAL.asc に配置されます。出力ファイルでは、ASCII フォーマットが印刷に適しています。

このデフォルトのデータ セット以外のものをエクスポートする場合は、export コマンドと共にその他のコマンドとファイルを使用します。

次のデータ型をエクスポートできます。

global	5 分ごとの要約および 1 時間ごとの要約
application	5 分ごとの要約および 1 時間ごとの要約
process	1 分間の詳細
disk device	5 分ごとの要約および 1 時間ごとの要約
lvolume	5 分ごとの要約および 1 時間ごとの要約
transaction	5 分ごとの要約および 1 時間ごとの要約
configuration	データ コレクタの開始時の parm ファイル情報を含む 1 レコードとシステム構成情報
DSI クラス	DSI ログ ファイルのインターバルと要約
netif	5 分ごとの要約および 1 時間ごとの要約
cpu	5 分ごとの要約および 1 時間ごとの要約
filesystem	5 分ごとの要約および 1 時間ごとの要約

- 各データ型に必要なデータ項目 (メトリック) を指定できます。
- データ収集期間の開始日と終了日を、シフトや週末除外フィルタと共に指定できます。
- エクスポート テンプレート ファイルにエクスポート データの対象となるフォーマットを指定します。このファイルは、ASCII (テキスト) フォーマットでファイルを保存できる任意のテキスト エディタまたはワード プロセッサを使用して作成できます。
- デフォルトのエクスポート テンプレート ファイル `/var/opt/perf/reptfile` も使用できます。このファイルは次の出力フォーマットの設定値を指定します。
  - ASCII ファイル フォーマット
  - 欠落値に対する 0 (ゼロ)
  - フィールド区切り記号の空白
  - 60 分ごとの要約
  - カラムのヘッダーを含める
  - 指定のデータ型に対応する標準のメトリック セットをエクスポートに含める

特定のプラットフォームで作成したログ ファイルからデータを抽出またはエクスポートする場合は、同一のプラットフォームの `reptall` ファイルを使用してください。これは、サポートされるメトリック クラスのリストがプラットフォームによって異なるためです。

エクスポート タスクのサンプル

Performance Collection Component には、エクスポート テンプレート ファイルの 2 つのサンプル `repthist` と `reptall` が用意されています。これらのファイルは `/var/opt/perf/` ディレクトリにあります。`repthist` と `reptall` を使用すると、共通のエクスポート タスクを実行できます。また、これらのファイルは次に説明するタスクのようなカスタム タスクの開始点としても使用できます。

印刷可能な CPU レポートの生成

エクスポート テンプレート ファイル repthist には、ある期間のシステムの CPU とディスク使用量の文字グラフを生成するための指定が含まれます。このグラフは印刷可能な文字から構成され、132 カラムを印刷可能なデバイスに印刷できます。たとえば、次の extract プログラムのコマンドを使用して、最近 7 日間のグラフを生成できます。これは、約 2 ページ分になります (1 時間ごとの要約の代わりに、5 分間の詳細を指定すると、34 ページ分のデータが作成されます)。

```
logfile /var/opt/perf/datafiles/logglob
report /var/opt/perf/repthist
global summary
start today-7
export
```

エクスポート データは、エクスポート ファイル xfrsGLOBAL.asc に書き込まれます。このファイルを印刷するには、次のように入力します。

```
lp xfrsGLOBAL.asc
```

カスタマイズしたエクスポート ファイルの作成

エクスポート テンプレート ファイルを新規作成する場合は、extract プログラムの guide コマンドを使用してエクスポート テンプレート ファイルをコピーしてカスタマイズします。ガイド モードでは、reptall ファイルを /var/opt/perf/ ディレクトリからコピーし、指定されている scope または DSI のログ ファイルを読み込み、動的にデータ型とメトリック名のリストを作成します。

reptall ファイルには、scope ログ ファイル データの各データ型に使用可能なメトリックが含まれているため、必要な作業は使用するメトリックをコメントから外すだけです。この方法は、エクスポート テンプレート ファイル全体を入力し直すより簡単です。

データ ファイルのエクスポート

出力ファイル名を指定するために、export コマンドを発行する前に output コマンドを使用した場合は、すべてのエクスポート データがこのファイルに追加されます。対話型モードで、extract プログラムを実行し、直接ワークステーション (標準出力ファイル) にデータをエクスポートする場合は、export コマンドを発行する前に、extract コマンドで output stdout を指定します。

出力ファイルがデフォルトに設定されると、エクスポートするデータ型によって、エクスポート データが 14 種類のデフォルトの出力ファイルに分割されます。

デフォルトのエクスポート ログ ファイル名は次のとおりです。

xfrdGLOBAL.ext	グローバル詳細データ ファイル
xfrsGLOBAL.ext	1 時間ごとのグローバル要約データ ファイル
xfrdAPPLICATION.ext	アプリケーション詳細データ ファイル
xfrsAPPLICATION.ext	1 時間ごとのアプリケーション要約データ ファイル
xfrdPROCESS.ext	プロセス詳細データ ファイル
xfrdDISK.ext	ディスク デバイス詳細データ ファイル

xfrsDISK.ext	1 時間ごとのディスク デバイス要約データ ファイル
xfrdVOLUME.ext	論理ボリューム詳細データ ファイル
xfrsVOLUME.ext	論理ボリューム要約データ ファイル
xfrdNETIF.ext	netif 詳細データ ファイル
xfrsNETIF.ext	netif 要約データ ファイル
xfrdCPU.ext	CPU 詳細データ ファイル
xfrsCPU.ext	CPU 要約データ ファイル
xfrdFILESYSTEM.ext	ファイルシステム詳細データ ファイル
xfrsFILESYSTEM.ext	ファイルシステム要約データ ファイル
xfrdTRANSACTION.ext	トランザクション詳細データ ファイル
xfrsTRANSACTION.ext	トランザクション要約データ ファイル
xfrdCONFIGURATION.ext	構成データ ファイル

ここで ext は、asc (ASCII)、bin (バイナリ)、dat (データ ファイル)、wk1 (表計算シート) のいずれかです。

export コマンドを発行する前に、エクスポート テンプレート ファイルでデータに適合するデータ型と関連する項目を指定しないと、出力ファイルは作成されません。

エクスポート テンプレート ファイルの構文

エクスポート テンプレート ファイルには、エクスポート データのフォーマット方法およびエクスポート ファイルに含める内容に応じて、次の情報のすべてまたは一部が含まれます。

```
report      "export file title"
format      [ASCII]
            [datafile]
            [binary]
            [WK1] or
            [spreadsheet]
headings    [on]
            [off]
separator= "char"
summary=value
missing=value
layout=single | multiple
output=filename
data type datatype
items
```

パラメータ

report　エクスポート ファイルのタイトルを指定します。次の項の 102 ページの「エクスポート ファイルのタイトル」を参照してください。

format　エクスポート データのフォーマットを指定します。

ASCII

ASCII (またはテキスト) フォーマットは、プリンタや端末にファイルをコピーする最適なフォーマットです。フィールドは二重引用符 (") で囲みません。

datafile

datafile フォーマットは、ASCII フォーマットに類似していますが、数値以外のフィールドはすべて二重引用符で囲みます。二重引用符により、カラムの位置合わせが厳密でなくなるため、datafile フォーマットのファイルは直接印刷には適していません。datafile フォーマットは、ほとんどの表計算ソフトおよびグラフィックス ソフトにインポートするための最も簡単なフォーマットです。

binary

binary フォーマットは、数値が 2 進整数として表されるため、よりコンパクトです。変換が最小限で済み、メトリックの精度が最高であるため、このフォーマットはユーザーが記述した分析プログラムへの入力に最も適しています。直接印刷には適しません。

WK1 (表計算シート)

WK1 (表計算シート) フォーマットは、Microsoft Excel、その他の表計算ソフトおよびグラフィックス プログラムと互換性があります。

headings　エクスポート ファイルにリストされているメトリックのカラム ヘッダーを書き込むかどうかを指定します。headings off を指定すると、カラム ヘッダーはファイルに書き込まれません。ファイルの最初のレコードはエクスポート データです。headings on を指定すると、ASCII フォーマットと datafile フォーマットでは、最初のデータ レコードの前にエクスポート タイトルと、メトリックの各カラムのカラム ヘッダーが書き込まれます。binary フォーマット ファイルのカラム ヘッダーには、ファイル内のメトリックの説明が含まれます。WK1 フォーマットには、必ずカラム ヘッダーが書き込まれます。

separator　ASCII フォーマットまたは datafile フォーマットのデータの各フィールド間に印刷される文字を指定します。デフォルトの区切り記号は空白ですが、多くのプログラムではフィールドの区切り記号としてコンマが使用されています。区切り記号は、印刷される文字を指定しても印刷されない文字を指定してもかまいません。

summary　各要約インターバルを分単位で指定します。この値は、要約レコードの各レコードに収められる時間を決定します。デフォルトのインターバルは 60 分です。要約値は 5 ～ 1440 分 (1 日) に設定できます。

missing　欠落データの代わりに使用するデータ値を指定します。欠落データのデフォルト値は 0 です。欠落データと 0 データを区別するために、別の値を指定できます。次の場合は、データ項目が欠落します。

- scope コレクタの特定のバージョンで記録されなかった場合
- データ項目が属するインスタンス (アプリケーション、ディスク、トランザクション、netif) がそのインターバルにアクティブでなかったために、データ項目が scope で記録されなかった場合
- DSI ログ ファイルの場合、あるインターバルで dsilog プログラムにデータが提供されず、「欠落レコード」となったとき

デフォルトでは、欠落レコードはエクスポート データから除外されます。

layout　アプリケーション、ディスク、トランザクション、lvolume、netif などのデータ型に対して、単一のレイアウトか複数のレイアウト (レコード出力ごとの出力) かを指定します。

- 単一のレイアウトは、特定のインターバルでアクティブであった各アプリケーションについて 1 レコードにつき 1 つのインスタンスを書き込みます。たとえば、1 レコードには、エクスポートされたディスクが 1 つ書き込まれます。
- 複数のレイアウトは、各インターバルで 1 レコードに複数のインスタンスを書き込みます。このレコードの一部は、構成されている各アプリケーションについて保存されます。たとえば、1 レコードには、エクスポートされたすべてのディスクが書き込まれます。

output　エクスポートするデータが書き込まれる出力ファイル名を指定します。エクスポート データ項目のリストを開始するデータ型を示す行の直後に output *filename* を置くことで、エクスポートする各クラスまたは各 data type について output を指定できます。任意の有効なファイル名を output に指定できます。

また、output コマンドを使用すれば名前を対話式に指定できます。名前を指定すると、デフォルトの出力ファイル名がオーバーライドされます。

data type　エクスポート可能なデータ型 (グローバル、アプリケーション、プロセス、ディスク、トランザクション、lvolume、netif、構成、DSI クラス名) の 1 つを指定します。このパラメータは、このデータ型がエクスポートされるときに、コピーされるデータ項目をリストするエクスポート テンプレート ファイルのセクションを開始します。

items　エクスポート ファイルに書き込むメトリックを指定します。結果ファイルにメトリック名をリストする順序で、1 行に 1 つずつメトリック名をリストします。items をリストする前に、適切な data type を指定する必要があります。エクスポート テンプレート ファイルには、対象となるデータ型の数だけ項目リストを書き込めます。各 data type は、そのデータ型のエクスポートを選択した場合にのみ参照されます。

output パラメータおよび layout パラメータは、エクスポート テンプレート ファイル内で複数回使用できます。次に例を示します。

```
data type global
   output=myglobal
   gbl_cpu_total_util

data type application
```

```
output=myapp
layout=multiple
app_cpu_total_util
```

エクスポート テンプレート ファイルは、システム上に複数あってもかまいません。各ファイルが、特定の用途に合うエクスポート ファイル フォーマットのセットを定義できます。report コマンドを使用すると、export 関数で使用するエクスポート テンプレート ファイルを指定できます。

1 つのデータ型で異なるレイアウトは指定できません。たとえば、同じエクスポート ファイル内で、**data type disk** を **layout = multiple** で指定した後に、**layout = single** で指定することはできません。

エクスポート ファイルのタイトル

export file title の文字列は、次の項目で置き換えることができます。

!date	エクスポート関数を実行した日付
!time	エクスポート関数を実行した時刻
!logfile	ソース ログ ファイルの完全修飾名
!class	必要なデータ型
!collector	コレクタ プログラムの名前とバージョン (DSI ログ ファイルでは無効)
!system_id	データを収集したシステムの識別子 (DSI ログ ファイルでは無効)

次に例を示します。

```
report "!system_id data from !logfile on !date !time"
```

この文字列は次のタイトルのようなエクスポート ファイル タイトルを生成します。

```
barkley data from logglob on 02/02/99 08:30 AM
```

カスタム グラフまたはレポートの作成

エクスポートしたグローバル データとアプリケーション データを含むカスタム グラフまたはレポートを作成するには、次の手順に従います。

1. 各データ型から必要なデータ項目 (メトリック) を選択し、それらにアクセスするときのフォーマットを決定します。

   この例では、ヘッダーがなく、フィールドがコンマで区切られる ASCII ファイルを指定します。

2. 次の ASCII エクスポート テンプレート ファイルを作成し、/var/opt/perf/ ディレクトリに保存します。ファイル名は report1 にします。

```
REPORT "sample export template file (report1)"
FORMAT ASCII
HEADINGS OFF

DATA TYPE GLOBAL
   GBL_CPU_TOTAL_UTIL
   GBL_DISK_PHYS_IO_RATE
```

```
DATA TYPE APPLICATION
   APP_CPU_TOTAL_UTIL
   APP_DISK_PHYS_IO_RATE
   APP_ALIVE_PROCESSES
```

3 extract プログラムを実行します。

4 report コマンドを発行して、作成したエクスポート テンプレート ファイルを指定します。

```
report /var/opt/perf/report1
```

5 global コマンドと application コマンドを使用して、グローバル要約データとアプリケーション要約データを指定します。

```
global summary
application summary
```

6 export コマンドを発行して、エクスポートを開始します。

```
export
```

7 パフォーマンス データの取得元をプログラムに指定していないため、指定するように促すプロンプトが表示されます。この例では、デフォルトのログ ファイルが正しいため、**Enter** キーを押すだけです。

8 次のように出力されます。

```
exporting global data .........50%......100%
exporting application data .....50%......100%
The exported file contains 31 days of data from 01/01/99 to 01/31/99
                            examined  exported
data type                    records   records     space
-----------------------  ---------- --------- ---------
global      summaries                    736   0.20 Mb
application summaries                   2560   0.71 Mb
                                              ---------
                                                0.91 Mb
```

作成した 2 つのファイル (xfrsGLOBAL.asc および xfrsAPPLICATION.asc) に、指定のフォーマットでグローバル要約データとアプリケーション要約データが書き込まれます。

エクスポート ファイルの出力

各エクスポート ファイルの内容は、次のとおりです。

レポート ファイルのタイトル行	export title と headings on を指定した場合
名前 (application、netif、lvolume、transaction のいずれか)	複数レイアウト ファイルで headings on を指定した場合
見出し行 1	headings on を指定した場合
見出し行 2	headings on を指定した場合

最初のデータ レコード

2 番目のデータ レコード

...

最後のデータ レコード

レポート タイトルとヘッダーの行はファイル内で繰り返されません。

ASCII フォーマットと datafile フォーマットに関する注意

これらのフォーマット ファイルのデータは、印刷可能な ASCII フォーマットです。datafile フォーマットで数値以外のすべてのフィールドが二重引用符で囲まれることを除いて、ASCII フォーマットと datafile フォーマットは同一です。datafile では、ヘッダー情報も二重引用符で囲まれます。

ASCII ファイル フォーマットでは、二重引用符でフィールドを囲みません。したがって、ASCII ファイルのデータは、印刷したときにフィールドの位置がそろっています。

数値は、その範囲と内部的な精度に基づいてフォーマットされます。すべてのフィールドの長さが同じとは限らないため、各フィールドを開始するときに使用する区切り記号を指定する必要があります。

ユーザーが指定した区切り記号 (またはデフォルトの空白) は、ASCII フォーマットおよび datafile フォーマットで各フィールドを区切ります。レポートを印刷する場合、区切り記号として空白を使用すると目立ちます。別のプログラムでエクスポート テンプレート ファイルを読み込む場合、別の文字を区切り記号として使用する方が便利です。

コンマは多くのアプリケーションで区切り記号として適用されますが、区切り記号ではないコンマを含むデータ項目もあります。このようなコンマにより、分析プログラムは混乱します。日付と時刻のフォーマットには、extract プログラムを実行するときに指定する母国語に基づき、異なる特殊文字が含まれます。

印刷されない特殊文字を区切り記号として使用するには、エクスポート テンプレート ファイルで separator パラメータの最初の二重引用符の直後にその特殊文字を入力します。

- ほとんどの場合、separator="," を使用した datafile フォーマットのファイルは、表計算ソフトで使用できます。
- 多くの表計算ソフトでは、1 シートで最大 256 カラムを使用できます。複数レイアウトのデータ型をエクスポートするときには、合計で 256 を超える項目が生成される可能性が高いので注意してください。report reportfile, show コマンドの出力を使用すると、このような問題が生じるかを特定できます。
- 下線を引けるプリンタの場合、ASCII フォーマットと縦線 (separator=|) を指定し、下線を有効に設定してファイルを印刷すると、見やすくなります。

binary フォーマットに関する注意

binary フォーマット ファイルでは、数値が 32 ビット整数として書き込まれます。これにより、ファイル全体のサイズを小さくしてスペースを節約できますが、binary ファイルを読み取れるプログラムが必要になります。binary フォーマット ファイルはプリンタや端末にコピーしないようにしてください。

binary フォーマットでは、数値以外のデータが ASCII フォーマットの場合と同じように書き込まれます。ただし、区切り記号は使用されません。binary フォーマット ファイルを正しく使用するには、**report** *reportfile*,**show** を指定するときに、extract で出力されるレコード レイアウト レポートを使用する必要があります。このレポートは、指定した各項目の開始バイトを提供します。

最大精度を維持し、非標準の binary 浮動小数点表現を避けるために、すべての数値が 32 ビット整数として書き込まれます。項目によっては、整数フォーマットに切り捨てる前に定数を掛けます。

たとえば、CPU が使用された秒数は、切り捨てる前に 1000 を掛けます。エクスポート ファイルの値を実際の秒数に変換するには、その値を 1000 で割ります。簡単に変換できるように、**headings on** を指定して、エクスポート ファイルに倍率を書き込みます。レポート タイトルと特別なヘッダー レコードは binary フォーマット ファイルに書き込まれ、プログラムによる解釈に役立てられます。

2 進整数は、extract プログラムを実行するシステムに固有のフォーマットで書き込まれます。たとえば、Intel のシステムは、「little endian」整数を書き込みますが、HP-UX、IBM AIX、Sun の各システムは、「big endian」整数を書き込みます。異なる「endian」を使用するシステムにバイナリ エクスポート ファイルを送信するときは、注意してください。

バイナリ ヘッダー レコード レイアウト

binary フォーマット エクスポート ファイルの各レコードには、ユーザーが指定したデータの前に特殊な 16 バイトのレコード ヘッダーが含まれます。report *reportfile*,show コマンドには、このレコード ヘッダーを構成する次の 4 つのフィールドが含まれます。

バイナリ レコード ヘッダー メトリック

Record Length	16 バイトのレコード ヘッダーを含むレコードのバイト数
Record ID	レコードの種類を識別するための数字 (下記参照)
Date_Seconds	1970 年 1 月 1 日からの時間 (秒単位)
Number_of_vars	このレコードで繰り返しているエントリの数

Record ID メトリックは、レコードに含まれるデータ型を独自に識別します。現在の Record ID 値は次のとおりです。

-1	Title Record
-2	First header Record (Contains Item Numbers)
-3	Second header Record (Contains Item Scale Factors)
-4	Application Name Record (for Multiple Instance Application Files)

-1 Title Record

-5 Transaction Name Record (for Multiple Instance Transaction Files)

-7 Disk Device Name Record (for Multiple Instance Disk Device Files)

-8 Logical Volume Name Record (for Multiple Instance Lvolume Files)

-9 Netif Name Record (for Multiple Instance Netif Files)

-10 Filesystem Name Record (for Multiple Instance Netif Files)

-11 CPU Name Record (for Multiple Instance Netif Files)

```
    1   Global Data Record      ( 5 minute detail  record)
  101   Global Data Record      (60 minute summary record)
    2   Application Data Record ( 5 minute detail  record)
  102   Application Data Record (60 minute summary record)
    3   Process Data Record     ( 1 minute detail record)
    4   Configuration Data Record
    7   Disk Device Data Record ( 5 minute detail record)
  107   Disk Device Data Record (60 minute summary record)
    8   Logical Volume Data Record ( 5 minute detail record)
  108   Logical Volume Data Record (60 minute summary record)
    9   Filesystem Data Record ( 5 minute detail record)
  109   Filesystem  Data Record (60 minute summary record)
   11   Netif Data Record       ( 5 minute detail record)
  111   Netif Data Record       (60 minute summary record)
   12   Transaction Data Record ( 5 minute detail record)
  112   Transaction Data Record (60 minute summary record)
   13   CPU Data Record ( 5 minute detail record)
  113   CPU Data Record ( 60 minute summary record)
 ClassID +1,000,000 Class Data Record ( 5 minute detail record)
 ClassID +1,000,000+100 Class Data Record (60 minute summary record)
```

Date_Seconds メトリックは、ユーザーが選択可能な Date_Seconds メトリックと等しく、対象となる日時に簡単にレコードを走査できるように用意されています。

Number_of_vars メトリックは、レコード内で繰り返されているフィールドのグループの数を示します。単一インスタンスのデータ型の場合、この値は 0 です。

複数インスタンスのアプリケーション レコードの場合、Number_of_vars メトリックは構成されるアプリケーションの数です。複数インスタンスのディスク デバイス レコードの場合、Number_of_vars メトリックは構成されるディスク デバイスの数です。すべてのヘッダー レコードで、このメトリックは繰り返し可能なグループの最大数です。

binary フォーマット ファイルには、タイトル レコードとヘッダー レコードに特別なフォーマットがあります。これらのレコードには、ファイル内容の説明に必要な情報が含まれているため、プログラムはそれを正確に解釈できます。headings off を指定すると、データ レコードのみがファイルに含まれます。headings on を指定すると、次のレコードがすべてのデータ レコードに先行します。

バイナリ ヘッダー レコード

Title Record	このレコード (Record ID -1) は、headings on を指定していると、ユーザーによるレポート タイトルの指定の有無にかかわらず書き込まれます。このレコードには、ログ ファイルとそのソースに関する情報が含まれます。
First Header Record	最初のヘッダー レコード (Record ID -2) には、ログ ファイルに含まれる項目に対応する固有の項目識別番号のリストが含まれます。項目 ID 番号の位置は、ファイル内の各エクスポート項目の位置とサイズを決定するために使用されます。
Second Header Record	2 番目のヘッダー レコード (Record ID -3) には、エクスポート項目に対応する倍率のリストが含まれます。詳細は、この項で後述している「倍率」を参照してください。
Application Name Record	このレコード (Record ID -4) は、Multiple Layout フォーマットを使用するアプリケーション データ ファイルのみに含まれます。これは、ファイルの残りの部分にあるアプリケーション メトリックのグループに対応するアプリケーション名をリストします。
Transaction Name Record	このレコード (Record ID -5) は、Multiple Layout フォーマットを使用するトランザクション追跡データ ファイルのみに含まれます。これは、ファイルの残りの部分にあるトランザクション メトリックのグループに対応するトランザクション名をリストします。
Disk Device Name Record	このレコード (Record ID -7) は、Multiple Layout フォーマットを使用するディスク デバイス データ ファイルのみに含まれます。これは、ファイルの残りの部分にあるディスク デバイス メトリックのグループに対応するディスク デバイス名をリストします。
Logical Volume Name Record	このレコード (Record ID -8) は、Multiple Layout フォーマットを使用する論理ボリューム データ ファイルのみに含まれます。これは、ファイルの残りの部分にある論理ボリューム メトリックのグループに対応する論理ボリューム名をリストします。
Netif Name Record	このレコード (Record ID -9) は、Multiple Layout フォーマットを使用する netif (LAN) データ ファイルのみに含まれます。これは、ファイルの残りの部分にある `netif` デバイス メトリックのグループに対応する `netif` デバイス名をリストします。
ファイルシステム Name Record	このレコード (Record ID -12) は、Multiple Layout フォーマットを使用するファイルシステムデータ ファイルのみに含まれます。これは、ファイルの残りの部分にある filesystems メトリックのグループに対応する filesystems 名をリストします。
Cpu Name Record	このレコード (Record ID -13) は、Multiple Layout フォーマットを使用する CPU データ ファイルのみに含まれます。これは、ファイルの残りの部分にある CPU メトリックのグループに対応する CPU 名をリストします。

Binary Title Record

BINARY ファイルのタイトル レコードには、エクスポート ファイルの内容をプログラムが解釈するための情報が含まれます。このレコードは、headings on を指定している場合、エクスポート ファイルに書き込まれます。

Binary Title Record の内容は、次のとおりです。

```
Record Length          4 byte Int    Length of Title Record
Record ID              4 byte Int     -1
Date_Seconds           4 byte Int     Date exported file was created
Number_of_vars         4 byte Int     Maximum number of repeating
                                      variables
Size of Fixed Area     4 byte Int     Bytes in nonvariable part of
                                      record
Size of Variable Entry 4 byte Int     Bytes in each variable entry
GMT Time Offset        4 byte Int     Seconds offset from Greenwich
                                      Mean Time
Daylight Savings Time  4 byte Int     =1 indicates Daylight Savings
                                      Time
System ID             40 Characters,  System Identification
Collector Version     16 Characters,  Name & version of the data
                                      collector
Log File Name         72 Characters,  Name of the source log file
Report Title          100 Characters, User specified report title
```

Binary Title Record の Date_Seconds、GMT Time Offset、Daylight Savings Time の各メトリックは、システムおよびエクスポート ファイルが作成された時刻に適用されます。このシステムがデータを記録したシステムと同じでない場合、これらのフィールドはファイル内のデータを正しく反映できません。

Binary Item Identification Record

バイナリ ファイルの最初のヘッダー レコード (Record ID -2) には、エクスポートされた各項目に固有の項目番号が含まれます。各項目 ID は 4 バイトの整数で、この製品に用意されている rxitemid ファイルを使用して相互に参照できます。項目 ID のフィールドは、ファイルの残りの部分でそれらが示されるデータ フィールドにそろえられます。すべてのバイナリ エクスポート データ項目が、エクスポート ファイルで 4 バイトを複数回使用し、それぞれが 4 バイトの境界から始まります。データ項目に 4 バイト以上のスペースが必要な場合、対応する項目 ID フィールドの左側が 0 で埋められます。

たとえば、プログラム プロセス メトリックには 16 バイトが必要です。そのデータ レコードと項目 ID レコードは次のようになります。

```
Header 1 (Item Id Record) ...|   0|   0|   0|12012|
Process Data Record          |Prog|ram_|name| _aaa|
```

Binary Scale Factor Record

バイナリ ファイルの 2 番目のヘッダー レコード (Record ID -3) には、各エクスポート項目の倍率が含まれます。別のコンピュータ アーキテクチャが浮動小数点を実装している場合に問題を最小にするために、数値項目は 32 ビット (4 バイト) 整数としてバイナリ ファイルにエクスポートされます。ほとんどの項目は、固定倍率を掛けてから、整数フォーマットに適合させるために切り捨てられます。これにより、倍数を分母として使用して、浮動小数点数は小数部として表現できます。

各倍率は、ほとんどのデータ項目と一致する 32 ビット (4 バイト) 整数です。数値以外のメトリックおよびその他の特殊な値のメトリックを示すために、倍率に対して特殊な値が使用されます。

特殊な倍率

ASCII フィールドなど、数値以外のメトリックには、0 という倍率があります。-1 という倍率は発生しないはずですが、発生した場合、extract プログラムの内部エラーが示され、レポートされます。

DATE フォーマットは、フィールドの中で最も影響の小さい 16 ビット (最も右寄りの 16 ビット) の MPE CALENDAR フォーマットです。日付の倍率は 512 です。これを 32 ビット整数にする (512 で割る) と、年が日付の整数部分として、その年の日 (512 で割る) が小数部として分離されます。

TIME は 4 バイトのバイナリ フィールドです (時、分、秒、十分の一秒)。時間の倍率は 65536 です。時間を 65536 で割ると、整数部分が (時 * 256) + 分の数字になります。

Date_Seconds 値は、バイナリ ファイルで処理する方が簡単です。

Application Name Record

アプリケーション データを Multiple Layout フォーマットでエクスポートすると、特殊な Application Name Record が書き込まれ、アプリケーション グループを識別します。バイナリ フォーマット ファイルの場合、このレコードには Record ID -4 が含まれます。このレコードは、バイナリ レコードのヘッダー (16 バイト) と、エクスポート データの開始日に定義された各アプリケーション名 (20 バイト) から構成されます。

データ抽出時間にアプリケーションが追加または削除される場合、Application Name Record は新しいアプリケーション名で繰り返されます。

Transaction Name Record

トランザクション データを Multiple Layout フォーマットでエクスポートすると、特殊な Transaction Name Record が書き込まれ、トランザクション名を識別します。バイナリ フォーマット ファイルの場合、このレコードには Record ID -5 が含まれます。このレコードは、バイナリ レコードのヘッダー (16 バイト) と、エクスポート データの開始日に構成された各トランザクションのアプリケーション トランザクション名 (60 バイトに切り捨て) から構成されます。データ抽出時間にトランザクションが追加される場合、Transaction Name Record は、元のリストの最後に新しいアプリケーション トランザクション名が追加されて繰り返されます。データ抽出開始後に削除されるトランザクションは、Multiple Layout データ レコードから削除されません (詳細は、『HP Operations エージェント & Glance Plus for UNIX トランザクション追跡』を参照してください)。

Disk Device Name Record

ディスク デバイス データを Multiple Layout フォーマットでエクスポートすると、特殊な Disk Device Name Record が書き込まれ、ディスク デバイス名を識別します。バイナリ フォーマット ファイルの場合、このレコードには Record ID -7 が含まれます。このレコードは、バイナリ レコードのヘッダー (16 バイト) と、エクスポート データの開始日に構成された各ディスク デバイス名 (20 バイト) から構成されます。

データ抽出時間にディスク デバイスが追加される場合、Disk Device Name Record は、元のリストの最後に新しいディスク デバイス名が追加されて繰り返されます。データ抽出開始後に削除されるディスク デバイスは、Multiple Layout データ レコードから削除されません。

Logical Volume Name Record

論理ボリューム データを Multiple Layout フォーマットでエクスポートすると、特殊な Logical Volume Name Record が書き込まれ、論理ボリューム名を識別します。バイナリ フォーマット ファイルの場合、このレコードには Record ID -8 が含まれます。このレコードは、バイナリ レコードのヘッダー (16 バイト) と、エクスポート データの開始日に構成された各論理ボリュームのディスク デバイス名 (20 バイト) から構成されます。

データ抽出時間に論理ボリュームが追加される場合、Logical Volume Name Record は、元のリストの最後に新しい論理ボリューム名が追加されて繰り返されます。データ抽出開始後に削除される論理ボリュームは、Multiple Layout データ レコードから削除されません。

Netif Name Record

`netif` データを Multiple Layout フォーマットでエクスポートすると、特殊な Netif Name Record が書き込まれ、netif デバイス名を識別します。バイナリ フォーマット ファイルの場合、このレコードには Record ID -11 が含まれます。このレコードは、バイナリ レコードのヘッダー (16 バイト) と、エクスポート データの開始日に構成された各 `netif` デバイスの `netif` デバイス名 (20 バイト) から構成されます。

データ抽出時間に `netif` デバイスが追加される場合、Netif Name Record は、元のリストの最後に新しいデバイス名が追加されて繰り返されます。データ抽出開始後に削除される netif デバイスは、Multiple Layout データ レコードから削除されません。

7 extract のコマンド

この章では、extract プログラムのコマンドについて説明します。この章は、コマンド構文をまとめた表、データの抽出およびエクスポートを行うコマンドの表、アルファベット順にコマンドを説明しているコマンド リファレンスから構成されています。

extract のコマンドとパラメータは、大文字と小文字を組み合わせて入力できます。コマンド名の最初の 3 文字だけは必ず入力します。ただし、weekdays コマンドと weekly コマンドだけは、コマンド名全体を入力する必要があります。たとえば、application detail コマンドは app det と省略できます。

これらのコマンドの使用例については、extract プログラムのオンライン ヘルプを参照してください。

次のページの表では、extract コマンドとパラメータの構文をまとめています。

extract 関数では要約データは作成できません。要約データは export 関数でのみ作成することができます。

表 8　　extract のコマンド：構文とパラメータ

コマンド	パラメータ
application	on detail summary（export のみ） both（export のみ） off（デフォルト）
class	detail（デフォルト） summary（export のみ） both（export のみ） off
cpu	detail summary（export のみ） both　（export のみ） off（デフォルト）
configuration	on detail off（デフォルト）

表 8　　extract のコマンド：構文とパラメータ（続き）

コマンド	パラメータ
disk	on detail summary（export のみ） both　（export のみ） off（デフォルト）
exit e	
export	day[*ddd*] [-*days*] week [*ww*] [-*weeks*] month[*mm*] [-*months*] year [*yy*] [-*years*]
extract	day[*ddd*] [-*days*] week [*ww*] [-*weeks*] month[*mm*] [-*months*] year [*yy*] [-*years*]
filesystem	detail summary（export のみ） both　（export のみ） off（デフォルト）
global	on detail（デフォルト） summary（export のみ） both（export のみ） off
guide	
help	
list	filename *
logfile	logfile
lvolume	on detail summary（export のみ） both（export のみ） off（デフォルト）
menu	
monthl	
y	*yyymm* *mm*

表 8　　extract のコマンド：構文とパラメータ（続き）

コマンド	パラメータ
netif	on detail summary（export のみ） both（export のみ） off（デフォルト）
output	*outputfile* ,new ,purgeboth ,append
process	on detail [app#[-#],...] off（デフォルト） killed
quit q	
report	[*export template file*], show
shift	*starttime - stoptime* all day noweekends
sh !	シェル コマンド
show	all
start	*date* [*time*] today [-*days*] [*time*] last [-*days*] [*time*] first [+*days*] [*time*]
stop	*date* [*time*] today [-*days*] [*time*] last [-*days*] [*time*] first [+*days*] [*time*]
transaction	on detail summary（export のみ） both　（export のみ） off（デフォルト）
weekdays	1.....7
weekly	*yywww* *ww*
yearly	*yyyy* *yy*

次の表は、データの抽出およびエクスポートに使用するコマンドと、使用されるログ ファイル (scope ログ ファイルまたは DSI ログ ファイル) の種類のリストです。

表 9　　extract のコマンド : データの抽出とエクスポート

コマンド	データの抽出	データのエクスポート	scope ログファイル	DSI ログファイル
application	x	x	x	
class	x	x	x	x
configuration		x	x	
cpu	x	x	x	
disk	x	x	x	
export		x	x	x
extract	x		x	
filesystem	x	x	x	
global	x	x	x	
logfile	x	x	x	x
lvolume	x	x	x	
monthly	x		x	
netif		x	x	
output	x	x	x	x
process	x	x	x	
report		x	x	x
shift	x		x	x
start	x	x	x	x
stop	x	x	x	x
transaction	x	x	x	x
weekdays		x	x	x
weekly	x		x	
yearly	x		x	
logicalsystems	x	x	x	

application

application コマンドは、抽出またはエクスポートするアプリケーション データの種類を指定するときに使用します。

デフォルトは、application off です。

構文

```
                    [on]
                    [detail]
application         [summary]
                    [both]
                    [off]
```

パラメータ

on または detail	生の 5 分間詳細データの抽出またはエクスポートを指定します。
summary (export のみ)	次の基準によるデータの要約を指定します。 • 指定したエクスポート テンプレート ファイルの summary パラメータで指定される時間 (分単位) (export のみ) • デフォルトの要約インターバル (1 時間) (export または extract) 使用する要約インターバルによりますが、抽出またはエクスポートしたデータのサイズは要約によって大幅に減少できます。たとえば、1 時間ごとの要約データは、5 分間の詳細データのサイズの約 10 分の 1 になります。
both (export のみ)	詳細データおよび要約データの抽出またはエクスポートを指定します。
off	このタイプのデータを抽出またはエクスポートしないことを指定します。

Performance Manager を使用している場合、5 分間隔のポイントでアプリケーション グラフを作成する前に、抽出ファイルに詳細データを含める必要があります。

例

この例では、application コマンドにより、詳細アプリケーション ログ ファイル データがエクスポートされます。output export ファイルには、myrept エクスポート テンプレート ファイルで指定されているアプリケーション メトリックが含まれます。

```
logfile /var/opt/perf/datafiles/logglob
global off
application detail
report /var/opt/perf/myrept
export
```

コマンド ライン引数を使用して上記の操作を実行するには、次のように入力します。

```
extract -a -r /var/opt/perf/myrept -xp
```

class

class コマンドは、エクスポートする DSI データのクラスや、抽出またはエクスポートする scope データのクラスを指定するときに使用します。

デフォルトは、class detail です。

構文

		[detail]
class	[*classname*]	**[summary]**
		[both]
		[off]

パラメータ

classname	同類をまとめたメトリックのグループ名です。
detail	DSI ログ ファイルの場合は、DSI ログ ファイルに設定されている時刻に従ってエクスポートする詳細データの量を指定します。詳細情報は、247 ページの「データ ソース統合の概要」を参照してください。 scope ログ ファイルの場合は、生の 5 分間の詳細を抽出またはエクスポートすることを指定します。
summary bothoff	115 ページの「application」コマンドの説明にある「パラメータ」を参照してください。summary および both はエクスポートのみ可能です。

例

クラス acctg_info を含む DSI ログ ファイルに要約データをエクスポートするには、次のコマンドを発行します。

```
class acctg_info summary
```

ユーザーがログ ファイルを指定し、extract プログラムによりこのファイルをオープンすると、acctg_info クラスがログ ファイルにあることが確認され、このクラスがエクスポートされます。

このコマンドの別の使用例を示します。

```
CLASS ACCTG_INFO SUMMARY
class ACCTG_INFO summary
class acctg_info sum
```

コマンドは大文字でも小文字でもかまいません。クラス名は常に大文字に変換され、比較されます。

次の例では、クラス fin_info の要約データをエクスポートします。

```
extract>
class fin_info summary
export
```

コマンド ライン引数を使用して上記の操作を実行するには、次のように入力します。

```
extract -l dsi.log -C fin_info summary -xp
```

configuration

configuration コマンドは、システム構成情報をエクスポートするかどうかを指定するときに使用します。

デフォルトは、configuration off です。

構文

```
                [on]
configuration   [detail]
                [off]
```

パラメータ

on または detail すべての構成レコードのエクスポートを指定します。

off　　構成データをエクスポートしないことを指定します。

ログ ファイルに含まれるすべての構成情報がエクスポートされます。configuration コマンドと共に使用される begin、end、shift、start、stop、noweekends の各コマンドはすべて無視されます。

configuration コマンドは、export 関数のみに影響します。extract 関数は必ずシステム構成情報を抽出するため、extract 関数には影響しません。

例

この例では、configuration コマンドにより、システム構成情報がエクスポートされます。出力エクスポート ファイルには、myrept エクスポート テンプレート ファイルで指定されている構成メトリックが含まれます。

```
logfile /var/opt/perf/datafiles/logglob
configuration on
report /var/opt/perf/myrept
export
```

コマンド ライン引数を使用して上記の操作を実行するには、次のように入力します。

```
extract -c -r /var/opt/perf/myrept -xp
```

cpu

cpu コマンドは、CPU の要約レベルを指定するときに使用します。

デフォルトは、cpu off です。

構文

```
            [detail]
cpu         [summary]
            [both][off]
```

パラメータ

detail	5 分間の詳細レコードを抽出またはエクスポートします。
summary	要約レコードをエクスポートします。
both	詳細レコードと要約レコードの両方をエクスポートします。
off	CPU データを抽出またはエクスポートしません。

例

この例では、cpu コマンドにより、2001 年 7 月 26 日から収集された CPU detail データがエクスポートされます。エクスポート テンプレート ファイルを指定していないため、デフォルトのエクスポート テンプレート ファイル reptfile が使用されます。reptfile で指定されているように、すべてのディスク メトリックが出力ファイルに含まれます。

```
logfile /var/opt/perf/datafiles/logglob
global off
cpu detail
start 7/26/01
export
```

コマンド ライン引数を使用して上記の操作を実行するには、次のように入力します。

```
extract -u -b 7/26/01 -xp
```

disk

disk コマンドは、抽出またはエクスポートするディスク デバイス データの種類を指定するときに使用します。

デフォルトは、disk off です。

構文

```
          [on]
          [detail]
disk      [summary]
          [both]
          [off]
```

パラメータ

on または detail summary bothoff	この章の最初に説明されている application コマンドの説明にある「パラメータ」を参照してください。summary および both はエクスポートのみ可能です。

例

この例では、disk コマンドにより、1999 年 7 月 5 日から収集された disk detail データがエクスポートされます。エクスポート テンプレート ファイルを指定していないため、デフォルトのエクスポート テンプレート ファイル reptfile が使用されます。reptfile で指定されているように、すべてのディスク メトリックが出力ファイルに含まれます。

```
logfile /var/opt/perf/datafiles/logglob
global off
```

```
disk detail
start 7/5/99
export
```

コマンド ライン引数を使用して上記の操作を実行するには、次のように入力します。

```
extract -D -b 7/5/99 -xp
```

exit

exit コマンドは、extract プログラムを終了するときに使用します。exit コマンドは、extract プログラムの quit コマンドに相当します。

構文

```
exit
e
```

export

export コマンドは、エクスポート ファイル フォーマットにデータをコピーするプロセスを開始するときに使用します。

構文

```
                [day          [ddd] [yyddd] [-days]]
export          [week         [ww] [yyww] [-weeks]]
                [month        [mm] [yymm] [-months]]
                [year         [yy] [yyyy] [-years]]
```

パラメータ

特定のインターバルでデータをエクスポートするときには、次のパラメータの 1 つを使用します。

day	1 日を表します
week	月曜日から 日曜日までの 1 週間を表します
month	月の最初の日から最後の日までの 暦月を表します
year	年の最初の日から最後の日までの 暦年を表します

export コマンドと共にパラメータを使用しない場合は、エクスポート データに対して使用するインターバルが、start コマンドおよび stop コマンドにより設定されます。

使い方

特定のインターバル (day、week、month、year) を指定するには、次の 4 つの方法があります。

- 現在のインターバル ― パラメータのみを指定します。たとえば、month は当月を意味します。

- 過去のインターバル ― パラメータ、マイナス符号、現在からさかのぼるインターバルの数を指定します。たとえば、day-1 は前日を示し、week-2 は今週から 2 週間前の週を示します。
- 絶対インターバル ― パラメータと正数を指定します。数字は、現在の年の絶対インターバルを示します。たとえば、day 2 は今年の 1 月 2 日を示します。
- 絶対インターバルと年 ― パラメータと大きな整数を指定します。数字は、年の最後の 2 桁とその年の絶対インターバルを示す数から構成する必要があります。このフォーマットでは、絶対日付には 5 桁を使用し (99002 は 1999 年 1 月 2 日を示します)、その他のすべてのパラメータでは 4 桁を使用します (month 9904 は 1999 年 4 月を示します)。

事前にログ ファイルまたはエクスポート テンプレート ファイルを指定していない場合、logfile コマンドはデフォルトのグローバル ログ ファイル logglob ファイルを使用し、report コマンドはデフォルトのエクスポート テンプレート ファイル reptfile を使用します。

その他のすべてのパラメータでは、設定値またはデフォルトが使用されます。これらの動作の詳細は、「application」、「configuration」、「global」、「process」、「disk」、「lvolume」、「netif」、「CPU」、「filesystem」、「transaction」、「output」、「shift」、「start」、「stop」のコマンド説明を参照してください。

export コマンドは、指定したデータ型および要約レベルに基づき、最大 16 種類のデフォルトの出力ファイルを作成します。

xfrdGLOBAL.ext	グローバル詳細データ ファイル
xfrsGLOBAL.ext	1 時間ごとのグローバル要約データ ファイル
xfrdAPPLICATION.ext	アプリケーション詳細データ ファイル
xfrsAPPLICATION.ext	1 時間ごとのアプリケーション要約データ ファイル
xfrdPROCESS.ext	プロセス詳細データ ファイル
xfrdDISK.ext	ディスク デバイス詳細データ ファイル
xfrsDISK.ext	ディスク デバイス要約データ ファイル
xfrdVOLUME.ext	論理ボリューム詳細データ ファイル
xfrsVOLUME.ext	論理ボリューム要約データ ファイル
xfrdNETIF.ext	netif 詳細データ ファイル
xfrsNETIF.ext	netif 要約データ ファイル
xfrdCPU.ext	CPU 詳細データ型
xfrsCPU.ext	CPU 要約データ型
xfrdFILESYSTEM.ext	ファイルシステム詳細データ型
xfrsFILESYSTEM.ext	ファイルシステム要約データ型
xfrdTRANSACTION.ext	トランザクション詳細データ ファイル
xfrsTRANSACTION.ext	トランザクション要約データ ファイル
xfrdCONFIGURATION.ext	構成詳細データ ファイル

ここで、ext は asc、dat、bin、wk1 のいずれかです

デフォルトのファイル名は、データ型名から作成されます。データが詳細データであるか要約データであるかによって、接頭辞が xfrd または xfrs になります。拡張子は、指定されているデータ フォーマットによって、asc (ASCII)、bin (バイナリ)、dat (データ ファイル)、wk1 (表計算シート) のいずれかになります。

たとえば、classname = ACCTG_INFO の場合は、次の export ファイル名になります。

xfrdACCTG_INFO.wk1　ACCT_INFO の詳細表計算シート データ

xfrsACCTG_INFO.asc　ACCT_INFO の要約 ASCII データ

データのエクスポートに関する詳細は、第 5 章の 96 ページの「エクスポート関数の概要」を参照してください。

例

この例では、export コマンドにより、前日の 8:00 AM から 5:00 PM までに収集されたログ ファイル データをエクスポートします。エクスポート テンプレート ファイルを指定していないため、デフォルトのエクスポート テンプレート ファイル reptfile が使用されます。reptfile で指定されているように、すべてのグローバル メトリックが出力ファイルに含まれます。

```
logfile /var/opt/perf/datafiles/logglob
start today-1 8:00 am
stop today-1 5:00 pm
global both
export
```

コマンド ライン引数を使用して上記の操作を実行するには、次のように入力します。

```
extract -gG -b today-1 8:00 am -e today-1 5:00 pm -xp
```

extract

extract コマンドは、生ログ ファイルから抽出ファイル フォーマットにデータをコピーするプロセスを開始するときに使用します。抽出ファイルは、アーカイブするときや、Performance Manager などの分析プログラムで分析するときに使用できます。データを生ログ ファイルと抽出ファイルから抽出できます。

extract コマンドは、DSI ログ ファイルのデータを処理するときには使用できません。

構文

```
          [day       [ddd] [yyddd] [-days]]
extract   [week      [ww] [yyww] [-weeks]]
          [month     [mm] [yymm][-months]]
          [year      [yy] [yyyy] [-years]]
```

パラメータ

特定のインターバルでデータを抽出するときには、次のパラメータの 1 つを使用します。

day	1 日を表します
week	月曜日から日曜日までの 1 週間を表します
month	月の最初の日から最後の日までの暦月を表します
year	年の最初の日から最後の日までの暦年を表します

extract コマンドと共にパラメータを使用しない場合は、データ抽出に対して使用するインターバルが、start コマンドおよび stop コマンドにより設定されます。

使い方

特定のインターバル (day、week、month、year) を指定するには、次の 4 つの方法があります。

- 現在のインターバル ― パラメータのみを指定します。たとえば、month は当月を意味します。
- 過去のインターバル ― パラメータ、マイナス符号、現在からさかのぼるインターバルの数を指定します。たとえば、day-1 は前日を示し、week-2 は今週から 2 週間前の週を示します。
- 絶対インターバル ― パラメータと正数を指定します。数字は、現在の年の絶対インターバルを示します。たとえば、day 2 は今年の 1 月 2 日を示します。
- 絶対インターバルと年 ― パラメータと大きな整数を指定します。数字は、年の最後の 2 桁とその年の絶対インターバルを示す数から構成する必要があります。このフォーマットでは、日付には 5 桁を使用し (99002 は 1999 年 1 月 2 日を示します)、その他のすべてのパラメータでは 4 桁を使用します (month 99904 は 1999 年 4 月を示します)。

extract コマンドがデータの抽出を開始します。事前に指定していない場合、logfile コマンドと output コマンドは、extract コマンドが実行されるときに次のデフォルトを仮定します。

```
log file = /var/opt/perf/datafiles/logglob
output file = rxlog,new
```

その他のすべてのパラメータでは、設定値またはデフォルトが使用されます。これらの動作の詳細は、「application」、「global」、「process」、「disk」、「lvolume」、「netif」、「CPU」、「filesystem」、「transaction」、「shift」、「start」、「stop」 のコマンド説明を参照してください。

抽出するログ ファイルのサイズは、最大 3.5 ギガバイトです。

例

最初の例では、2000 年 3 月 1 日から 2000 年 6 月 30 日の平日の 8:00 AM から 5:00 PM までに収集されたデータが抽出されます。グローバル要約データとアプリケーション詳細データだけが抽出されます。

```
logfile /var/opt/perf/datafiles/logglob
start 03/01/00
stop 06/30/00
shift 8:00 am - 5:00 pm noweekends
global detail
application detail
extract
```

コマンド ライン引数を使用して上記の操作を実行するには、次のように入力します。

```
extract -ga -b 03/01/00 -e 6/30/00 -s 8:00 am - 5:00 noweekends -xt
```

2 番目の例では、新しい抽出ログ ファイル rxjan00 が作成されます。この名前が付いている既存のファイルは削除されます。2000 年 1 月 1 日から 2000 年 1 月 31 日までに収集されたすべての生ログ ファイル データが抽出されます。

```
logfile /var/opt/perf/datafiles/logglob
output rxjan00,purge
start 01/01/00
stop 01/31/00
global detail
application detail
transaction detail
process detail
disk detail
lvolume detail
netif detail
filesystem detail
cpu detail
extract
```

コマンド ライン引数を使用して上記の操作を実行するには、次のように入力します。

```
extract -f rxjan00,purge -gatpdznyu -b 01/01/00 -e 01/31/00 -xt
```

filesystem

filesystem コマンドは、extract または export を実行する filesystem データの要約レベルを指定するときに使用します。

デフォルトは、filesystem off です。

構文

```
             [detail]
filesystem   [summary]
             [both]
             [off]
```

パラメータ

detail	5 分間の詳細レコードを抽出またはエクスポートします。
summary	要約レコードをエクスポートします。
both	詳細レコードと要約レコードの両方をエクスポートします。
off	ファイルシステム データを抽出またはエクスポートしません。

例

この例では、filesystem コマンドにより、2001 年 7 月 26 日から収集されたファイルシステム詳細データがエクスポートされます。エクスポート テンプレート ファイルを指定していないため、デフォルトのエクスポート テンプレート ファイル reptfile が使用されます。reptfile で指定されているように、すべての filesystem メトリックが出力ファイルに含まれます。

```
logfile /var/opt/perf/datafiles/logglob
global off
filesystem detail
start 7/26/01
export
```

コマンド ライン引数を使用して上記の操作を実行するには、次のように入力します。

```
extract -y -b 7/26/01 -xp
```

global

global コマンドは、抽出またはエクスポートするグローバル データの量を指定するときに使用します。

デフォルトは、global detail です (コマンド ライン モードでのデフォルトは global off です)。

構文

```
            [on]
            [detail]
global      [summary]
            [both]
            [off]
```

パラメータ

detail または on summary both off	この章の最初に説明されている application コマンドの説明にある「パラメータ」を参照してください。summary および both はエクスポートのみ可能です。

使い方

5 分間隔のポイントで Performance Manager グローバル グラフを作成する場合、詳細データを抽出する必要があります。

グラフの作成に必要なデータ レコードが少ないため、要約データのグラフは Performance Manager によって迅速に作成されます。グローバル要約データのみが抽出される場合、Performance Manager のグローバル グラフは 5 分間隔のデータ ポイントでは作成できません。

both オプションは 1 時間ごとの要約レコードにより得られるアクセス速度を維持します。また、5 分間隔のポイントで Performance Manager グローバル グラフを作成できます。

Performance Manager を使用している場合、システム全体の動作を正しく理解するにはグローバル データが必要なため、off パラメータを指定しないようにしてください。抽出ファイルに少なくとも 1 種類のグローバル データが含まれていない場合、Performance Manager グローバル グラフは作成できません。

例

この例では、グローバル データをエクスポートしないことを指定するために、global コマンドが使用されています (global detail がデフォルトです)。詳細トランザクション データのみがエクスポートされます。output export ファイルには、myrept エクスポート テンプレート ファイルで指定されているトランザクション メトリックが含まれます。

```
extract>
logfile /var/opt/perf/datafiles/logglob
global off
transaction detail
report /var/opt/perf/myrept
```

export

コマンド ライン引数を使用して上記の操作を実行するには、次のように入力します。

extract -l -t -r /var/opt/perf/myrept -xp

guide

guide コマンドは、ガイド コマンド モードに移行するときに使用します。ガイド コマンド インターフェイスは extract のさまざまなコマンドについてガイドし、一般的で実行可能ないくつかの作業を行うように促すプロンプトを表示します。

構文

guide

使い方

- extract の対話型モードからガイド コマンド モードに移行するには、**guide** と入力します。
- パラメータのデフォルト値を適用するには、**Return** キーを押します。
- ガイド コマンド モードを終了し、対話型モードに戻るには、guide> プロンプトで **q** と入力します。

このコマンドでは、パラメータ設定値を組み合わせる必要がありません。このコマンドは、ほとんどのユーザーに有益な結果をもたらす設定値を選択します。extract の対話型コマンド モードからは、extract の関数を完全に制御できます。

DSI ログ ファイル データをエクスポートしている場合、ガイド コマンド モードを使用して、カスタマイズしたエクスポート テンプレート ファイルを作成し、データをエクスポートすることをお勧めします。

help

help コマンドは、オンライン ヘルプにアクセスするときに使用します。

構文

help [*keyword***]**

使い方

extract のコマンドと作業に関する情報 (ヘルプの情報) を得るには、パラメータを入力します。キーワードを入力すると、別のトピックに移動できます。情報が複数のページにわたる場合は、**Return** を押すと次のページが表示されます。ヘルプ システムを終了し、extract プログラムに戻るには、**q** または **quit** と入力します。

また、特定のトピックに関するヘルプも要求できます。次に例を示します。

help tasks

または

help resize parms

この形式の help コマンドを使用すると、指定のトピックのヘルプ テキストを受信しますが、extract のコマンド エントリ コンテキストは終了していません。対話型モードでヘルプ サブシステムを指定していないため、次の extract コマンドを入力する前に **quit** を入力する必要はありません。

list

list コマンドは、extract プログラムのすべてのレポートのリスト ファイルを指定するときに使用します。

構文

list [*file*]
[*]

使い方

extract を使用中にリスト デバイスを指定する場合は、随時 list を使用できます。このコマンドは、ファイル名またはリスト デバイス名を使用して、ユーザー指定の設定値を出力します。リスト ファイルがすでに存在する場合、出力はそのファイルに追加されます。

リスト デバイスに送信されるデータも画面上に表示されます。

extract の実行中に、次のように入力します。

list outfilename

リストされているデバイスをユーザー端末に返すには、次のいずれかを入力します。

list stdout

または

list *

現在のリスト デバイスを特定するには、次のようにパラメータを使用せずに list コマンドを入力します。

list

リスト ファイルが stdout でない場合、ほとんどのコマンドは入力されたときにリスト ファイルにエコーされます。

例

次の例では、リスト デバイスが mylist に設定されています。次のコマンドの結果は、mylist に出力され、画面に表示されます。

```
extract>
logfile /var/opt/perf/datafiles/logglob
list mylist
global detail
shift 8:00 AM - 5:00 PM
extract
```

logfile

logfile コマンドは、ログ ファイルをオープンするときに使用します。ログ ファイルは、extract プログラムのすべての関数に対してオープンする必要があります。これを明示的に行うには logfile コマンドを発行し、暗黙的に行うには extract コマンドまたは export コマンドを発行します。ログ ファイル名を指定しない場合は、extract プログラムがログ ファイル名の入力を促し、デフォルトのグローバル ログ ファイル /var/opt/perf/datafiles/logglob が表示されます。デフォルトを使用しても、別のログ ファイルを指定してもかまいません。

構文

logfile [*logfile***]**

使い方

ログ ファイルをオープンするには、生ログ ファイル名または抽出ログ ファイル名を指定します。export コマンドにより作成されるファイルの名前は指定できません。抽出ログ ファイル名を指定する場合、すべての情報がこのファイルから得られます。生ログ ファイル名を指定する場合、生ログ ファイルにアクセスするために、グローバル ログ ファイル名を指定する必要があります。この場合、指定するのは生ログ ファイル名のみです。

ログ ファイルが現在の作業ディレクトリにない場合、そのパスを入力する必要があります。

グローバル ログ ファイルとその他の生ログ ファイルは同じディレクトリにある必要があります。これらのファイル名は次のとおりです。

logglob	グローバル ログ ファイル
logappl	アプリケーション ログ ファイル
logproc	プロセス ログ ファイル
logdev	デバイス ログ ファイル
logtran	トランザクション ログ ファイル
logindx	索引ログ ファイル

ログ ファイルをオープンすると、そのログ ファイルの目次が表示されます。

生ログ ファイル名は変更しないようにしてください。生ログ ファイルにアクセスするときに、プログラムは標準ログ ファイル名が有効であると仮定します。複数のシステムのログ ファイルを同じシステムに配置して分析を行うために、ログ ファイル名を変更する必要がある場合、まずデータを抽出してから抽出ログ ファイルの名前を変更してください。

例

これは、典型的なデフォルト グローバル ログ ファイルのリストです。

```
Global      file:/var/opt/perf/datafiles/logglob, version D
Application file:/var/opt/perf/datafiles/logappl
Process     file:/var/opt/perf/datafiles/logproc
Device      file:/var/opt/perf/datafiles/logdev
Transaction file:/var/opt/perf/datafiles/logdev
Index       file:/var/opt/perf/datafiles/logindx
System ID:homer
System Type 9000/715/ S/N 2223334442 O/S HP-UX B.10.20 A
Data collector:SCOPE/UX C.02.30
File Created:  10/08/99
Data Covers:44 days to 11/20/99
Shift is:All Day
Data records available are:
```

```
Global Application Process Disk Volume Transaction
Maximum file sizes:
Global=10.0 Application=10.0 Process=20.0 Device=10.0 Transaction=10.0 MB
The first GLOBAL      record is on 10/08/99 at 08:17 AM
The first NETIF       record is on 10/08/99 at 08:17 AM
The first APPLICATION record is on 11/17/99 at 12:15 PM
The first PROCESS     record is on 10/08/99 at 08:17 AM
The first DEVICE      record is on 10/31/99 at 10:45 AM
The Transaction data file is empty
The default starting date & time = 10/08/99 11:50 AM   (LAST -30)
The default stopping date & time  = 11/20/99 11:59 PM   (LAST -0)
```

lvolume

lvolume コマンドは、抽出またはエクスポートする論理ボリューム データの種類を指定するときに使用します (このコマンドは、HP-UX システム上でのみ使用されます)。

デフォルトは、lvolume off です。

構文

```
                [on]
                [detail]
lvolume         [summary]
                [both]
                [off]
```

パラメータ

on または detail summary both off	この章の最初に説明されている application コマンドの説明にある「パラメータ」を参照してください。summary および both はエクスポートのみ可能です。

例

この例では、新しい抽出ログ ファイル rx899 が作成され、同じ名前の既存のファイルが削除されます。8 月 1 日から 8 月 31 日までに収集されたすべての論理ボリューム データが抽出されます。

```
logfile /var/opt/perf/datafiles/logglob
output rx899,purge
start 08/01/99
stop 08/31/99
global detail
lvolume detail
month 9908
```

コマンド ライン引数を使用して上記の操作を実行するには、次のように入力します。

```
extract -f rx899,purge -gz -xm 9908
```

menu

menu コマンドは、使用可能な extract のコマンドのリストを出力するときに使用します。

構文

menu

例

```
Command     Parameters   Function
HELP        [topic]      Get information on commands and options
GUIDE       Enter guided commands mode for novice users
LOGFILE     [logname]    Specify a log file to be processed
LIST        [filename|*] Specify the listing file
OUTPUT      [filename]
            [,NEW/PURGE/APPEND] Specify a destination file
REPORT      [filename][,SHOW]   Specify an Export Format file for EXPORT"
GLOBAL      [DETAIL/SUMMARY/BOTH/OFF] Extract GLOBAL records
APPLICATION [DETAIL/SUMMARY/BOTH/OFF] Extract APPLICATION records
PROCESS  [DETAIL/OFF/KILLED][APP=] Extract PROCESS records
DISK        [DETAIL/SUMMARY/BOTH/OFF] Extract DISK DEVICE records
LVOLUME     [DETAIL/SUMMARY/BOTH/OFF] Extract Logical VOLUME records
NETIF       [DETAIL/SUMMARY/BOTH/OFF] Extract Logical NETIF records
CPU         [DETAIL/SUMMARY/BOTH/OFF] Extract CPU records
FILESYSTEM  [DETAIL/SUMMARY/BOTH/OFF] Extract FILESYSTEM records
CONFIG      [DETAIL/OFF]    Export CONFIGURATION records
CLASS       classname[DETAIL/SUMMARY/BOTH/OFF]  Export classname records
TRANSACTION [DETAIL/SUMMARY/BOTH/OFF] Extract TRANSACTION records
START       [startdate time]  Specify a starting date and time for SCAN
STOP        [stopdate  time]  Specify an ending  date and time for SCAN
SHIFT       [starttime - stoptime] [NOWEEKENDS] Specify daily shift times
SHOW        [ALL]  Show the current program settings
EXPORT      [d/w/m/y][-offset] Copy log file records to HOST format files
EXTRACT     [d/w/m/y][-offset] Copy selected records to output (or append)
file
WEEKLY      [ww/yyww]  Extract one calendar  week's data with auto file names
MONTHLY     [mm/yymm]  Extract one calendar month's data with auto file names
YEARLY      [yy/yyyy]  Extract one calendar  year's data with auto file names
WEEKDAYS   [1...7]  Set days to exclude from export 1=Sunday ! or SH
            [command] Execute a system command
MENU or ?List the command menu (this listing)
EXIT or Q Terminate the program
```

monthly

monthly コマンドは、暦月に基づいてデータの抽出を指定するときに使用します。実行中、このコマンドは抽出するデータの月と年に基づいて、開始日と終了日を適切な日付に設定します。

出力ファイル名は、文字 rxmo の後に、抽出する年を示す 4 桁の数字と月を示す 2 桁の数字が続きます。たとえば、1999 年 3 月に抽出したデータは、rxmo199903 ファイルに出力されます。

構文

monthly [*yymm*]
[*mm*]

パラメータ

monthly	当月 (デフォルト) からデータを抽出します。
monthly *mm*	現在の年のデータから特定の月のデータを抽出します (*mm* は 01 ～ 12 の数字です)。
monthly *yymm*	特定の月と年のデータを抽出します (*yymm* は年の最後の 2 桁を示す数字と月を示す 2 桁の数字から構成されます)。たとえば、1999 年 2 月のデータを抽出するには、**monthly 9902** と指定します。

monthly コマンドを実行する前にログ ファイルを指定しない場合は、デフォルトの logglob ファイルが使用されます。

使い方

アーカイブするデータを月単位で抽出するときは、monthly コマンドを使用します。

抽出して要約するデータ型は、extract コマンドの通常の規則に従い、monthly コマンドを実行する前に設定できます。月間出力ファイルがない場合、これらの設定値が適用されます。月間出力ファイルがある場合、データは最初に指定したデータ型に基づいてそのファイルに追加されます。

monthly コマンドには、前月の抽出ファイルをオープンし、そのファイルが完成しているか、つまりその月の最終日までの抽出データが含まれているかを確認する機能があります。ファイルが完成していない場合、monthly コマンドはデータをこのファイルに追加し、前月の抽出を完了します。

たとえば、monthly コマンドを 1999 年 5 月 7 日に実行すると、5 月 1 日から現在の日付 (5 月 7 日) までのデータを含むログ ファイル rxmo199905 が作成されます。

1999 年 6 月 4 日に、再度 monthly コマンドを実行すると、当月の rxmo199906 ファイルが作成される前に、前月の rxmo199905 ファイルがオープンして確認されます。このファイルが完成していない場合は、データが追加され、1999 年 5 月 31 日までの抽出を完了します。次に、rxmo199906 ファイルが作成され、1999 年 6 月 1 日から現在の日付 (6 月 4 日) までのデータを保持します。

少なくとも月 1 回 monthly コマンドを実行すると、この機能が翌月のファイルを作成する前に、各月のファイルを完成します。rxmo199905 (5 月) と rxmo199906 (6 月) など、2 つの連続した月間ファイルがあるときには、最初のファイルが対応する月に対して完成しているとみなされるため、そのファイルをアーカイブして削除してもかまいません。

monthly コマンドと extract month コマンドは、どちらも暦月のデータを抽出する点で類似しています。monthly コマンドは、output コマンドの設定値を無視し、事前に定義されている出力ファイル名を代わりに使用します。また、このコマンドは、前月の抽出ログ ファイルがシステム上に残っている場合、欠落データをこのファイルに追加しようとします。一方、extract month コマンドは、output コマンドの設定値を使用します。このコマンドは、前月の抽出ファイル名を認識しないため、このファイルにデータを追加できません。

例

この例では、1999 年 5 月に記録された詳細アプリケーション データが抽出されます。

```
logfile /var/opt/perf/datafiles/logglob
global off
application detail
monthly 9905
```

コマンド ライン引数を使用して上記の操作を実行するには、次のように入力します。

```
extract -a -xm 9905
```

netif

netif コマンドは、extract または export を実行する論理ネットワーク インターフェイス (LAN) データの種類を指定するときに使用します。netif データは logdev ファイルに記録されます。

デフォルトは、netif off です。

構文

```
          [on]
          [detail]
netif     [summary]
          [both]
          [off]
```

パラメータ

on または detail summary both off	この章の最初に説明されている application コマンドの説明にある「パラメータ」を参照してください。summary および both はエクスポートのみ可能です。

例

この例では、2000 年 3 月 1 日から 2000 年 6 月 30 日の平日午前 8 時から午後 5 時の間に収集された詳細 netif データが抽出されます。

```
logfile /var/opt/perf/datafiles/logglob
start 03/01/00
stop 06/30/00
shift 8:00 AM - 5:00 PM noweekends
netif detail
extract
```

コマンド ライン引数を使用して上記の操作を実行するには、次のように入力します。

```
extract -n -b 03/01/00 -e 6/30/00 -s 8:00 am - 5:00 noweekends -xt
```

output

output コマンドは、extract 関数または export 関数用の出力ファイル名を指定するときに使用します。

2 つ目のパラメータはオプションであり、同じ名前の出力ファイルが存在する場合の動作を指定します。

構文

```
                          [,new]
output    [filename]      [,purge]
                          [,append]
```

パラメータ

,new	出力ファイルを新しいファイルにすることを指定します。これはバッチ モードのデフォルトの動作です。同じ名前のファイルが存在する場合、バッチ ジョブが終了します。
,purge	既存のファイルを削除して、新しい出力ファイル用に空きスペースを作ることを指定します。
,append	既存の抽出ファイルにデータを追加することを指定します。指定した名前の出力ファイルが存在しない場合、新しいファイルが作成されます。

使い方

バッチ モードで動作を指定しない場合は、デフォルトの動作である ,new が使用されます。対話型モードでは、重複するファイルが見つかると動作の入力を促すプロンプトが表示されます。

出力ファイルを指定しない場合、デフォルトの出力ファイルが作成されます。デフォルトの出力ファイル名は次のとおりです。

extract の場合：rxlog

export の場合：

```
xfrdGLOBAL.ext
xfrsGLOBAL.ext
xfrdAPPLICATION.ext
xfrsAPPLICATION.ext
xfrdPROCESS.ext
xfrdDISK.ext
xfrsDISK.ext
xfrdLVOLUME.ext
xfrsLVOLUME.ext
xfrdNETIF.ext
xfrsNETIF.ext
xfrdCPU.ext
xfrsCPU.ext
xfrdFILESYSTEM.ext
xfrsFILESYSTEM.ext
```

```
xfrdTRANSACTION.ext
xfrsTRANSACTION.ext
xfrdCONFIGURATION.ext
```

ここで、ext は asc (ASCII)、dat (データ ファイル)、bin (バイナリ)、wk1 (表計算シート) のいずれかです。

エクスポートの操作と共に次のように stdout (または *) という特別なファイル名を使用すると、出力を直接 stdout ファイルに送信できます (stdout は、通常は端末またはワークステーションですが、シェル コマンドを使用してリダイレクト可能です)。

```
output stdout
```

または

```
output *
```

出力をデフォルトの設定値に返すには、次のように入力します。

```
output default
```

または

```
output -
```

エクスポート テンプレート ファイルで output パラメータを使用すると、エクスポート ファイルのデフォルトの出力ファイル名を無効にできます。

抽出操作のファイルは ASCII デバイスに対応していないため、stdout に出力できません。また、同じ理由によりエクスポート操作のバイナリ フォーマットまたは WK1 フォーマットも stdout ファイルに出力できません。

既存のエクスポート データ ファイルへの抽出データの追加や、既存の抽出ファイルへのエクスポート データの追加を行わないようにしてください。間違ったデータ型を追加しようとすると、エラーが生じます。

例

この例では、出力ファイルが指定されていないため、抽出するアプリケーション要約データには、デフォルトの出力ファイル rxlog が使用されます。,purge オプションは、既存の出力ファイルを削除することを指定しています。

```
extract>
logfile /var/opt/perf/datafiles/logglob
output rxlog,purge
global off
application detail
extract month 9905
```

コマンド ライン引数を使用して上記の操作を実行するには、次のように入力します。

```
extract -f rxlog,purge -a -xm 9905
```

process

process コマンドは、プロセス データを抽出またはエクスポートするかどうかを指定するときに使用します。

デフォルトは、process off です。

構文

```
            [on]
process     [detail]        [application=#[-#] ,...]
            [off]
            [killed]
```

パラメータ

on	プロセス データの抽出またはエクスポートを指定します。
detail	process detail の指定は、process on の指定と同じです。
off	プロセス データを抽出またはエクスポートしないことを指定します。
killed	対象となる理由に killed が含まれるプロセスのみを指定します (測定インターバル中に終了されたプロセス)。
application	選択したアプリケーションに属するプロセスのみを指定します。アプリケーションは 1 つの番号またはアプリケーション番号の範囲として入力できます (7 - 9 はアプリケーション 7、8、9 を意味します)。アプリケーション番号は、データ収集時に parm ファイルのアプリケーション定義の順序により決定されます。複数のアプリケーションを指定している場合、それぞれのアプリケーションをコンマで区切ります。

プロセス データにより、抽出ログ ファイルのサイズが大幅に増大する可能性があります。分析を行うためにワークステーションにログ ファイルをコピーする場合、抽出するプロセス データの量を制限できます。

例

この例では、process コマンドがインターバル中に終了したプロセスのうち、アプリケーション 1、4、6、7、8、10 に属するプロセスを指定します。utility プログラムの scan コマンドを使用して、特定のアプリケーションのアプリケーション番号を検索します。

```
process killed applications=1,4,6-8,10
```

quit

quit コマンドは、extract プログラムを終了するときに使用します。quit コマンドは、extract プログラムの exit コマンドに相当します。

構文

```
quit
q
```

report

report コマンドは、export 関数で使用するエクスポート テンプレート ファイルを指定するときに使用します。エクスポート テンプレート ファイルが指定されていない場合は、デフォルトのエクスポート テンプレート ファイル reptfile が使用されます。エクスポート テンプレート ファイルは、export 関数で使用する出力フォーマットのさまざまな属性を指定するときに使用します。また、エクスポートするメトリックも指定します。

対話型モードでエクスポート テンプレート ファイルを指定しない場合、必要なデータ型のすべてのメトリックは ASCII フォーマットでエクスポートされます。

構文

report [*exporttemplatefile***] [,show]**

パラメータ

,show　エクスポート テンプレート ファイルで指定されたすべてのメトリックについて、フィールドの位置と開始カラムをリストすることを指定します。このリストは、エクスポート ファイルを別のプログラムで処理するときに使用できます。

使い方

このコマンドを発行すると、事前に指定しているログ ファイルでエクスポート テンプレートのメトリックを検証するかしないかを確認するメッセージが表示され、入力を促すプロンプトが表示されます。この検証により、エクスポート テンプレート ファイルで指定されているメトリックがログ ファイルに存在することが確認されます。これにより、エクスポート テンプレート ファイルのエラーを確認できます。検証を実行しないと、エクスポートを実行するまでこの動作が延期されます。

ここで説明した report コマンドの ,show パラメータは、後述の show コマンドとは異なります。

sh

sh は、extract を終了せずにシェル コマンドを入力するときに使用します。**sh** またはエクスクラメーション マーク（**!**）の後に UNIX シェル コマンドを指定します。

構文

sh または **![shell command]**

パラメータ

sh ls　ls コマンドを実行し、extract に戻ります。このシェル コマンドは任意のシステム コマンドです。

!ls　上記と同じです。

!ksh　Korn シェルを開始します。直ちに extract には戻りません。extract プログラムに戻るときは、**exit** と入力するか、**CTRL-d Return** を押します。

使い方

1 つのコマンドを実行すると、自動的に extract に戻ります。コマンドを実行するたびに extract に戻らないで、複数のシェル コマンドを発行する場合は、次のように新しいシェルを開始します。

シェル コマンドの名前を指定せずに sh コマンドを発行すると、次のようにシェル コマンドを指定するように促すプロンプトが表示されます。次に例を示します。

```
sh

enter SYSTEM command:ls
```

shift

shift コマンドは、データの抽出を 1 日のうちの作業シフトに相当する時間や、週末 (土曜日および日曜日) 以外の日に限定するときに使用します。

デフォルトは shift all day で、週末を含む毎日のデータを抽出します。

構文

```
          [starttime-stoptime]
shift     [all day]
          [noweekends]
```

パラメータ

starttime パラメータと stoptime パラメータは、start コマンドの時間と同じフォーマットで入力します。午前 0 時をまたがるシフトも指定できます。starttime を stoptime より後の時刻にスケジュールすると、シフトは starttime に開始し、一晩中処理が行われ次の日の *stoptime* に終了します。

all day	デフォルトのシフトである 12:00 AM - 12:00 AM (または 24 時間制で示す場合 00:00 - 00:00) を指定します。
noweekends	土曜日と日曜日に記録されたデータを除外することを指定します。午前 0 時をまたがるシフトと共に noweekends を入力すると、土曜日または日曜日に開始するシフトで週末が構成されます。

例

この例では、1999 年 6 月 15 日から毎日 10:00 AM から 4:00 PM までに収集されたディスク詳細データが抽出されます。

```
extract>
logfile /var/opt/perf/datafiles/logglob
global off
disk detail
shift 10:00 am - 4:00 PM
start 6/15/99
extract
```

コマンド ライン引数を使用して上記の操作を実行するには、次のように入力します。

```
extract -d -b 6/15/99 -s 10:00 AM-4:00 PM -xt
```

show

show コマンドは、オープンしたファイルの名前と設定可能な extract パラメータのステータスのリストを作成するときに使用します。

構文

show [all]

ここで説明する show コマンドは、前述の **report** コマンドの , show パラメータとは異なります。

例

show のみを使用すると、次のようなリストが作成されます。

```
Logfile:/var/opt/perf/datafiles/logglob
Output:Default
Report:Default
List:"stdout"
The default starting date & time = 10/08/99 12:00 AM (LAST -30)
The default stopping date & time = 11/20/99 11:59 PM (LAST -0)
The default shift = 12:00 AM - 12:00 PM
GLOBAL           DETAIL           records will be processed
APPLICATION.. . . . . NO records will be processed
PROCESS .. . . . . NO records will be processed
DISK DEVICE.. . . . . NO records will be processed
LVOLUME.. . . . . NO records will be processed
TRANSACTION.. . . . . NO records will be processed
NETIF .. . . . . .NO records will be processed
CPU .. . . . . .NO records will be processed
FILESYSTEM.. . . . . .NO records will be processed
Configuration .. . . . . .NO records will be processed
```

show all を使用すると、次の例のようにより詳細なリストが作成されます。

```
Logfile:/var/opt/perf/datafiles/logglob
Global       file:/var/opt/perf/datafiles/logglob,version D
Application file:/var/opt/perf/datafiles/logappl
Process     file:/var/opt/perf/datafiles/logproc
Device      file:/var/opt/perf/datafiles/logdev
Transaction file:/var/opt/perf/datafiles/logdev
Index       file:/var/opt/perf/datafiles/logindx
System ID:homer
System Type 9000/715/ S/N 2223334442 O/S HP-UX B.10.20 A
Data collector:SCOPE/UX C.02.30
File Created:   10/08/99
Data Covers:44 days to 11/20/99
Shift is:All Day
Data records available are:
  Global Application Process Disk Volume Transaction
Maximum file sizes:
  Global=10.0 Application=10.0 Process=20.0 Device=10.0
  Transaction=10.0 MB
Output:Default
Report:Default
List:"stdout"
```

```
The default starting date & time = 10/08/99 11:50 AM (LAST -30)
The default stopping date & time = 11/20/99 11:59 PM(LAST - 0)
The default shift = 12:00 AM - 12:00 PM
GLOBAL..........DETAIL..........records will be processed
APPLICATION.....................NO records will be processed
PROCESS.........................NO records will be processed
DISK DEVICE.....................NO records will be processed
LVOLUME.........................NO records will be processed
TRANSACTION.....................NO records will be processed
NETIF...........................NO records will be exported
CPU.............................NO records will be processed
FILESYSTEM......................NO records will be processed
Configuration ..................NO records will be exported
Export Report Specifications:
   Interval = 3600, Separator = " "
   Missing data will not be displayed
   Headings will be displayed
   Date/time will be formatted
   Days to exclude:None
```

start

start コマンドは、extract 関数と export 関数の開始日時を設定するときに使用します。デフォルトの開始日は、ログ ファイル内の最終日より 30 日前の日付です。また、30 日未満のレコードしかない場合、デフォルトの開始日はログファイル内の最初のレコードの日付になります。

構文

```
          [date [time]]
start     [today [-day][time]]
          [last [-days][time]]
          [first [+days][time]]
```

パラメータ

date	date フォーマットは、使用しているシステムに構成されている母国語によって異なります。母国語を使用しない場合やデフォルトの言語を C に設定している場合、データ フォーマットは *mm*/*dd*/*yy* (月 / 日 / 年) になります。たとえば、extract 関数または export 関数の場合、1999 年 9 月 30 日は 09/30/99 になります。
time	time フォーマットも、使用している母国語によって異なります。C 言語の場合、フォーマットは hh:mm AM または hh:mm PM (12 時間制の時 : 分の後に AM または PM を付ける形式) になります。たとえば、午前 7 時は 07:00 AM になります。 24 時間制の時刻もすべての言語で使用できます。たとえば、11:30 PM を 23:30 とすることもできます。 日付または時刻を不適切なフォーマットで入力すると、正しいフォーマットの例が表示されます。 開始時刻を指定しないと、午前 0 時 (12:00 AM) とみなされます。指定日の午前 0 時が開始時刻の場合、その日の始まり (24 時間制の場合 00:00) に開始されます。
today	現在の日付を指定します。today-days などのパラメータの修飾は、現在の日付からさかのぼる日数を指定します。たとえば、today-1 は前日を示し、today-2 は前前日を示します。
last	ログ ファイルに含まれる最終日を表すときに使用します。last-days パラメータは、ログ ファイル内の最終日からさかのぼる日数を指定します。
first	ログ ファイルに含まれる最初の日を表すときに使用します。first+days パラメータは、ログ ファイル内の最初の日から経過した日数を指定します。

使い方

次のコマンドは、start コマンドで設定されている開始日を無効にします。

- weekly
- monthly
- yearly
- extract (day、week、month、year のいずれかのパラメータを使用する場合)
- export (day、week、month、year のいずれかのパラメータを使用する場合)

例

この例の start コマンドは、抽出する最初のインターバルの開始時刻として 1999 年 6 月 5 日 8:00 AM を指定します。output コマンドは出力ファイル myout を指定します。

```
logfile /var/opt/perf/datafiles/logglob
start 6/5/99 8:00 am
output myout
global detail
extract
```

コマンド ライン引数を使用して上記の操作を実行するには、次のように入力します。

```
extract -g -b 06/05/99 8:00 AM -f myout -xt
```

stop

stop コマンドは、extract 関数または export 関数の終了日時を指定するときに使用します。

デフォルトの終了日時は、ログ ファイルに記録されている最後の日時になります。

構文

```
          [date [time]]
start     [today [-day][time]]
          [last [-days][time]]
          [first [+days][time]]
```

パラメータ

date　　date フォーマットは、使用しているシステムに構成されている母国語によって異なります。母国語を使用しない場合やデフォルトの言語を C に設定している場合、データ フォーマットは *mm*/*dd*/*yy* (月 / 日 / 年) になります。たとえば、抽出関数またはエクスポート関数の場合、1999 年 9 月 30 日は 09/30/99 になります。

time　　time フォーマットも、使用している母国語によって異なります。C 言語の場合、フォーマットは *hh:mm* AM または *hh:mm* PM (12 時間制の時 : 分の後に AM または PM を付ける形式) になります。たとえば、午前 7 時は 07:00 AM になります。

24 時間制の時刻もすべての言語で使用できます。たとえば、11:30 PM を 23:30 とすることもできます。

日時を不適切なフォーマットで入力すると、正しいフォーマットの例が表示されます。

終了時刻を指定しないと、午前 0 時の 1 分前 (11:59 PM) とみなされます。指定日の午前 0 時 (12:00 AM) が終了時刻の場合、その日の終わり (24 時間制の場合 23:59) に終了されます。

today　　現在の日付を指定します。today-days などのパラメータの修飾は、現在の日付からさかのぼる日数を指定します。たとえば、today-1 は前日を示し、today-2 は前前日を示します。

last　　ログ ファイルに含まれる最終日を表すときに使用します。last-days パラメータは、ログ ファイル内の最終日からさかのぼる日数を指定します。

first　　ログ ファイルに含まれる最初の日を表すときに使用します。first+days パラメータは、ログ ファイル内の最初の日から経過した日数を指定します。

使い方

次のコマンドは、*stop* コマンドによって設定される終了日を無効にします。

- weekly
- monthly
- yearly
- extract (day、week、month、year のいずれかのパラメータを使用する場合)
- export (day、week、month、year のいずれかのパラメータを使用する場合)

例

この例の stop コマンドは、抽出する最後のインターバルの終了時間として 1999 年 6 月 5 日 5:00 PM を指定します。output コマンドは出力ファイル myout を指定します。

```
extract>
logfile /var/opt/perf/datafiles/logglob
start 6/5/99 8:00 AM
stop 6/5/99 5:00 PM
output myout
global detail
extract
```

コマンド ライン引数を使用して上記の操作を実行するには、次のように入力します。

```
extract -g -b 6/5/99 8:00 AM -e 6/5/99 5:00 PM -f myout -xt
```

transaction

transaction コマンドは、抽出またはエクスポートするトランザクション データの種類を指定するときに使用します。

構文

```
             [on]
             [detail]
transaction  [summary]
             [both]
             [off]
```

パラメータ

on または detail summary both off	この章の最初に説明されている application コマンドの説明にある「パラメータ」を参照してください。summary および both はエクスポートのみ可能です。

例

rxmay99 という新しい抽出ログ ファイルが 1999 年 1 月 1 日に作成されます。この名前が付いている既存のファイルは削除されます。1999 年 5 月 1 日から 1999 年 5 月 31 日までに収集された生のトランザクション ログ ファイル データがすべて抽出されます。

```
extract>
logfile /var/opt/perf/datafiles/logglob
output rxmay99,purge
global detail
transaction detail
month 9905
```

コマンド ライン引数を使用して上記の操作を実行するには、次のように入力します。

```
extract -gt -f rxmay99,purge -xm 9905
```

weekdays

weekdays コマンドは、特定の曜日のデータをエクスポートから除くときに使用します (曜日 1 は日曜日を示します)。

構文

weekdays [1|2.....7]

使い方

1 週間のうち、特定の曜日のデータだけをエクスポートする場合は、このコマンドを使用してデータが不要な曜日を除きます。各曜日には、次の値が割り当てられています。

```
Sunday      =1
Monday      =2
Tuesday     =3
Wednesday   =4
Thursday    =5
Friday      =6
Saturday    =7
```

たとえば、月曜日から木曜日までに記録されたデータだけをエクスポートする場合は、金曜日、土曜日、日曜日のデータをエクスポートから除きます。

例

この例では、火曜日と木曜日に記録された詳細グローバル データがエクスポートから除外されます。出力エクスポート ファイルには、myrept エクスポート テンプレート ファイルで指定されているグローバル メトリックが含まれます。

```
extract>
logfile /var/opt/perf/datafiles/logglob
global detail
report myrept
weekdays 35
export
```

weekly

weekly コマンドは、1 週間のデータ抽出を指定するときに使用します。1 週間は月曜日から日曜日までの 7 日間と定義されます。

実行中、このコマンドは抽出されたデータの週と年に基づき、開始日と終了日を適切な日付に設定します。

構文

weekly [*yyww*]
[*ww*]

パラメータ

weekly	現在の週のデータを抽出します (デフォルト)。
weekly *ww*	現在の年のデータから特定の週のデータを抽出します (*ww* は、01 ～ 53 の任意の数字です)。
weekly *yyww*	特定の週と年のデータを抽出します (*yyww* は、年の最後の 2 桁を示す数字と、年間を通した 2 桁の週数を示す数字から構成されます)。たとえば、1999 年 20 週目は、weekly 9920 になります。

weekly コマンドを実行する前にログ ファイルを指定しない場合は、datafiles ディレクトリのデフォルトの logglob ファイルが使用されます。

使い方

アーカイブするデータを週単位で抽出するときは、weekly コマンドを使用します。

出力ファイル名は、文字 rxwe の後に、年の最後の 2 桁と抽出する週を示す 2 桁の週数が続きます。たとえば、1999 年の 12 週目 (3 月 22 日 月曜日 ～ 3 月 29 日 日曜日) は、rxwe9912 という名前のファイルに出力されます。

抽出して要約するデータ型は、extract コマンドの通常の規則に従い、weekly コマンドを実行する前に設定できます。週間出力ファイルがない場合、これらの設定が適用されます。週間出力ファイルがある場合、最初に選択したデータの型に基づいてデータがそのファイルに追加されます。

weekly コマンドには、前の週の抽出ファイルをオープンし、そのファイルが完成しているか、つまりその週の最終日までの抽出データが含まれているかを確認する機能があります。ファイルが完成していない場合、weekly コマンドはデータをこのファイルに追加し、前の週の抽出を完了します。

たとえば、weekly コマンドを 1999 年 5 月 20 日 (木) に実行すると、5 月 17 日 (月) から現在の日付 (5 月 20 日) までのデータを含むログ ファイル rxwe199920 が作成されます。

1999 年 5 月 26 日 (水) に、再度 weekly コマンドを実行すると、現在の週に対して rxwe199921 ファイルが作成される前に、前の週の rxwe199920 ファイルがオープンして確認されます。このファイルが完成していない場合は、データが追加され、1999 年 5 月 23 日 (日) までの抽出を完了します。次に、rxwe199921 ファイルが作成され、1999 年 5 月 24 日 (月) から現在の日付 (5 月 26 日) までのデータを保持します。

少なくとも週 1 回 weekly コマンドを実行すると、この機能が翌週のファイルを作成する前に、各週のファイルを完成します。2 つの連続した週間ファイル (rxwe199920 と rxwe199921 など) があるときには、最初のファイルが対応する週に対して完成しているとみなされるため、そのファイルをアーカイブして削除してもかまいません。

週数は、その開始日に基づいて数えられます。したがって、年の最初の週 (週 01) は、その年の最初の月曜日から始まる週になります。その月曜日より前の日は、前年の最後の週に属します。weekly コマンドと extract week コマンドは、どちらも月曜日から日曜日までの 1 週間のデータを抽出する点で類似しています。weekly コマンドは、output コマンドの設定値を無視し、事前に定義されている出力ファイル名を代わりに使用します。また、このコマンドは、前の週の抽出ログ ファイルがシステム上に残っている場合、欠落データをこのファイルに追加しようとします。一方、extract week コマンドは、output コマンドの設定値を使用します。このコマンドは、前の週の抽出ファイル名を認識しないため、このファイルにデータを追加できません。出力ファイル名は、文字 rxwe の後に、現在の年 (*yyyy*) と週数 (*ww*) が続きます。

例

この例では、weekly コマンドにより現在の週のデータが抽出され、前の週の抽出ファイルがある場合は、このファイルが完成します。

```
extract>
logfile /var/opt/perf/datafiles/logglob
global detail
application detail
process detail
weekly
```

コマンド ライン引数を使用して上記の操作を実行するには、次のように入力します。

```
extract -gap -xw
```

yearly

yearly コマンドは、暦年に基づいてデータの抽出を指定するときに使用します。

実行中、このコマンドは抽出する年に基づいて、開始日と終了日を適切な日付に設定します。

構文

yearly [*yyyy*]
[*yy*]

パラメータ

yearly	現在の年のデータを抽出します (デフォルト)。
yearly *yy*	特定の年のデータを抽出します (yy は、00 ～ 99 の数字です)。 00 ～ 27 の指定は、2000 ～ 2027 年とみなされ、71 ～ 99 の指定は、1971 ～ 1999 年とみなされます。
yearly *yyyy*	特定の年のデータを抽出します (*yyyy* は、年を示す 1971 ～ 2027 の数字です)。

yearly コマンドを実行する前にログ ファイルを指定しない場合は、デフォルトの logglob ファイルが使用されます。

使い方

アーカイブするデータを年単位で抽出するときは、yearly コマンドを使用します。

出力ファイル名は、文字 rxyr の後に、抽出する年を示す 4 桁の数字が続きます。したがって、1999 年のデータは rxyr1999 という名前のファイルに出力されます。

抽出して要約するデータ型は、extract コマンドの通常の規則に従い、yearly コマンドを実行する前に設定できます。年間出力ファイルがない場合、これらの設定が適用されます。年間出力ファイルがある場合、データは最初に選択したデータ型に基づいてそのファイルに追加されます。

yearly コマンドには、前年の抽出ファイルをオープンし、そのファイルが完成しているか、つまりその年の最終日までの抽出データが含まれているかを確認する機能があります。ファイルが完成していない場合、yearly コマンドはデータをこのファイルに追加し、前年の抽出を完了します。

たとえば、yearly コマンドを 1998 年 12 月 15 日に実行すると、1998 年 1 月 1 日から現在の日付 (12 月 15 日) までのデータを含むログ ファイル rxyr1998 が作成されます。

1999 年 1 月 5 日に、再度 yearly コマンドを実行すると、現在の年に対して rxyr1999 ファイルが作成される前に、前年の rxyr1998 ファイルがオープンして確認されます。このファイルが完成していない場合は、データが追加され、1998 年 12 月 31 日までの抽出を完了します。次に、rxyr1999 ファイルが作成され、1999 年 1 月 1 日から現在の日付 (1 月 5 日) までのデータを保持します。

少なくとも年 1 回 yearly コマンドを実行すると、この機能が翌年のファイルを作成する前に各年のファイルを完成します。2 つの連続した年間ファイル (rxyr1998 と rxyr1999 など) があるときには、最初のファイルが対応する年に対して完成しているとみなされるため、そのファイルをアーカイブして削除してもかまいません。

前のパラグラフの説明は、生ログ ファイルが 1 年分のデータを保持できるサイズである場合にのみ当てはまります。生ログ ファイルのサイズを小さくして、より頻繁に (月 1 回など) yearly コマンドを実行する方が一般的です。

yearly コマンドと extract year コマンドは、どちらも暦年のデータを抽出する点で類似しています。yearly コマンドは、output コマンドの設定値を無視し、事前に定義されている出力ファイル名を代わりに使用します。また、このコマンドは、前年の抽出ログ ファイルがシステム上に残っている場合、欠落データをこのファイルに追加しようとします。一方、extract year コマンドは、output コマンドの設定値を使用します。このコマンドは、前年の抽出ファイル名を認識しないため、このファイルにデータを追加できません。

例

この例では、アプリケーション詳細データとグローバル要約データが既存の年間要約ファイルに追加されます (必要に応じて、このファイルが作成されます)。出力ファイルは rxyr*yyyy* (*yyyy* は現在の年を表す) です。

```
extract>
logfile /var/opt/perf/datafiles/logglob
global detail
application detail
process off
yearly
```

コマンド ライン引数を使用して上記の操作を実行するには、次のように入力します。

```
extract -ga -xy
```

8 cpsh プログラムの使用

cpsh プログラムを使用できるのは、HP Ops OS Inst to Realtime Inst LTU または Glance Pak Software LTU を有効にしている場合のみです。

cpsh プログラムは、監視対象システムから収集されたリアルタイムのメトリック データを確認できる、新しいコマンド ライン プロンプトを表示します。

インタラクティブ モードの使用

cpsh プログラムは、インタラクティブ モードで使用できます。オプションを指定せずに cpsh コマンドを実行すると、新しいコマンド プロンプトが表示されます。このプロンプトでは、リアルタイム メトリックの詳細を表示するためのさまざまなタスクを実行できます。

cpsh プロンプトを表示する手順は以下のとおりです。

1. HP Operations エージェント がインストールされているシステムに、ルートまたは管理権限でログオンします。
2. ローカル システムの cpsh プロンプトを表示するには、以下のコマンドを実行します。

   cpsh

   リモート システムの cpsh プロンプトを表示するには、以下のコマンドを実行します。

   cpsh -n < *システム名* >

   この < *システム名* > には、リモート システムの完全修飾ドメイン名を指定します。

   または

   cpsh -n <*IP アドレス*>

   この <*IP アドレス*> には、リモート システムの IP アドレスを指定します。

   リモート システムの cpsh プロンプトを表示するときは、リモート システムで perfd プロセスが実行されている必要があります。

cpsh プロンプトが表示されます。

見やすい表形式でメトリック データを表示するには、-t オプションを指定して cpsh コマンドを実行します。

次に例を示します。

cpsh -t

または

cpsh -n < *システム名* > **-t**

3 cpsh プロンプトで使用できるコマンドの詳細を表示するには、**help** と入力します。

リアルタイム メトリックの表示

cpsh プロンプトを使用することで、利用できるメトリックのリアルタイムの値を表示できます。cpsh プロンプトで何らかの操作を実行するには、事前にメトリック コンテキストを設定する必要があります。perfd デーモンとその関連ユーティリティは、メトリック クラスに基づいて利用可能データを処理します。このため、cpsh ユーティリティでリアルタイム データを表示する場合は、利用可能データの表示操作を実行する前に、常にメトリック クラスを設定する必要があります。

メトリック クラスのメトリックのリアルタイム値を表示する手順は以下のとおりです。

1 cpsh プロンプトに **class** < *メトリック クラス* > と入力します。

2 指定したクラスに現在設定されているメトリックのリストを表示するには、**list** と入力します。指定したメトリック クラスのすべてのデフォルト メトリックが表示されます。

3 指定したクラスに属すメトリックの値を表示するには、cpsh プロンプトに **push** と入力します。cpsh プログラムは、該当メトリックのリアルタイム値を表形式で表示します。

4 cpsh プロンプトに戻るには、**Ctrl** キーを押しながら **C** キーを押します。

メトリック クラスの変更

メトリック クラスのデフォルト メトリックのリストには、別の使用可能メトリックを追加できます。cpsh プロンプトでメトリック クラスにメトリックを追加したり、削除する手順は以下のとおりです。

1 cpsh プロンプトを表示します。

2 cpsh プロンプトに **class** < *メトリック クラス* > と入力します。

3 **list** と入力します。指定したメトリック クラスのすべてのデフォルト メトリックが表示されます。

4 メトリックを削除する手順は以下のとおりです。

a cpsh プロンプトに **delete** < *メトリック名* > と入力します。

b **list** と入力します。指定したメトリック クラスのメトリック リストには、削除したメトリックは表示されなくなります。

5 メトリック クラスにメトリックを追加する手順は以下のとおりです。

a cpsh プロンプトに **add** < *メトリック名* > と入力します。

b **list** と入力します。指定したメトリック クラスのメトリック リストには、新たに追加したメトリックが表示されます。

すべての使用可能メトリックの表示

メトリック クラスに属すすべての使用可能メトリックを表示する手順は以下のとおりです。

1 cpsh プロンプトを表示します。

2 cpsh プロンプトに **class** < *メトリック クラス* > と入力します。

3 **list all** と入力します。指定したメトリック クラスに属すすべての使用可能メトリックが表示されます。

メトリック クラスの整理

メトリック クラスに対してメトリックの追加 / 削除処理を繰り返し実行する代わりに、メトリック クラスをまとめて整理することができます。メトリック クラスを整理して指定のメトリックを追加する手順は以下のとおりです。

1 cpsh プロンプトを表示します。

2 cpsh プロンプトに **class** < *メトリック クラス* > と入力します。

3 **init** < *メトリック名* > < *メトリック名* > < *メトリック名* > ... と入力します。

指定したメトリック クラスには、init で指定したメトリックのみが含まれるようになります。

メトリックのヘルプの表示

cpsh プロンプトでは、すべてのリアルタイム メトリックの説明を表示できます。メトリックの説明を表示する手順は以下のとおりです。

1 cpsh プロンプトを表示します。

2 class < *メトリック クラス* > と入力します。

3 cpsh プロンプトに **help** < *メトリック名* > と入力します。指定したメトリックの詳細な説明が表示されます。

要約メトリック データの表示

GLOBAL クラスと TABLE クラスのメトリックについては、cpsh プロンプトで要約データを表示できます。要約データを表示する手順は以下のとおりです。

1 cpsh プロンプトを表示します。

2 cpsh プロンプトに **class gbl** または **class tbl** と入力します。

3 **summ** < *間隔* > と入力します。この < *間隔* > には、要約の間隔を秒単位で指定します。< 間隔 > は、cpsh が接続されている perfd サーバーの収集間隔の倍数である必要があります。

cpsh ユーティリティは、指定したメトリック クラスに属すメトリックの以下の測定値を表示します。

- Maximum (最大)
- Minimum (最小)

- Average (平均)
- Standard deviation (標準偏差)

9 パフォーマンス アラーム

アラームを定義するときには、Performance Collection Component を使用します。アラームは、scope または DSI のメトリックが、定義しておいた条件を満たしたときや超えたときにユーザーに通知します。

アラームを定義するには、アラートまたは動作をトリガする条件を各監視対象システムに指定します。アラームは、アラーム定義ファイルである alarmdef で定義します。

データが scope または他のコレクタにより記録されると、そのデータはアラーム定義と比較され、条件を満たしているかどうかが確認されます。条件を満たしている場合は、アラートまたは動作がトリガされます。

リアルタイム アラーム ジェネレータを使用すると、以下のタスクを実行できます。

- HPOM コンソールへのアラート通知の送信
- アラーム通知が生成された際に、SNMP トラップを作成
- SNMP トラップを SNMP トラップ リスナーに転送
- 監視対象システム上のローカル アクションを実行

履歴ログ ファイル データをアラーム定義と比較して分析し、その結果を utility プログラムの analyze コマンドを使用してレポートできます。

アラームの処理

Performance Collection Component により収集されたパフォーマンス データは、alarmdef ファイルで定義されたアラーム条件と比較され、その条件を満たしているかどうかが確認されます。条件を満たしている場合、アラームが発生し、アラームに対して定義された動作 (ALERT、PRINT、EXEC) が実行されます。

ログ ファイルに収集されたデータは、alarmdef ファイルのアラーム定義と比較されます。アラーム条件が満たされると、アラーム定義で定義された動作が実行されます。ただし、データがログ ファイルに記録されない場合 (たとえば、しきい値パラメータが高位置に設定されている場合) は、alarmdef ファイルのアラーム条件が満たされても、アラームは生成されません。それぞれのメトリックのクラスのしきい値については、25 ページの「Thresholds」を参照してください。

アラーム定義には、以下のアクションを含めることができます。

- UNIX コマンドを使用して実行されるローカル アクション
- Network Node Manager または Operations Manager に送信されるメッセージ

アラーム ジェネレータ

Performance Collection Component アラーム ジェネレータは、アラーム通知の通信を処理します。アラーム ジェネレータは、アラーム ジェネレータ サーバー (perfalarm)、アラーム ジェネレータ データベース (agdb)、utility プログラム (agsysdb) で構成されます。

agdb には、SNMP ノードのリストが含まれます。agsysdb プログラムは、アラーム イベントにより実行される動作を表示するときや変更するときに使用します。

Performance Collection Component を起動すると、perfalarm が開始され、agdb を読み込み、アラーム通知の送信の有無およびその送信先を決定します。また、システム上の Operations Manager エージェントの有無を調べます。

アラート通知の送信先のリストを表示するには、次のコマンド ライン オプションを使用します。

agsysdb -l

Network Node Manager への SNMP トラップの送信

SNMP トラップを Network Node Manager に送信するには、次のコマンドを使用して Performance Collection Component の agdb にシステム名を追加する必要があります。

agsysdb -add *systemname*

ALERT が生成されると、定義したシステムに SNMP トラップが送信されます。トラップ テキストには、ALERT と同じメッセージが含まれます。

システムへの SNMP トラップの送信を停止するには、次のコマンドを使用して agdb からシステム名を削除する必要があります。

agsysdb -delete *systemname*

Operations Manager へのメッセージの送信

Performance Collection Component と同じシステム上に Operations Manager エージェントがある場合、Operations Manager にアラート通知を送信できます。Operations Manager エージェントは、中央 Operations Manager システムと通信します。

Operations Manager エージェントが Performance Collection Component システム上で動作している場合、デフォルトにより、アラーム ジェネレータは EXEC 文のアラームに定義されているローカル アクションを実行しません。代わりに、アラーム ジェネレータは Operations Manager のイベント ブラウザにメッセージを送信します。Operations Manager エージェントが Performance Collection Component システム上で動作していない場合、アラーム ジェネレータは Operations Manager にアラート通知を送信せず、ローカル アクションを実行します。

Operations Manager エージェントが Performance Collection Component システム上で動作している場合でも、次のコマンドを使用してデフォルトを変更すると、Operations Manager への情報の送信を停止できます。

agsysdb -ovo OFF

Performance Collection Component トラップを他のノードに送信するには、/etc/services ファイルに以下のエントリを追加します。

```
snmp-trap    162/tcp   # SNMPTRAP
snmp-trap    162/udp   # SNMPTRAP
```

この例では、162 がポート番号を指定しています。Performance Collection Component からトラップを別のノードに送信する場合は、/etc/services ファイルの snmp-trap 文字列がチェックされます。このエントリがないと、トラップは別のノードに送信されません。

ローカル アクションの実行

EXEC 文のローカル アクションは、Operations Manager エージェントが Performance Collection Component システム上で動作していない状態で実行されます。

次のようにデフォルトを変更すると、ローカル アクションを停止できます。

```
agsysdb -actions off
```

Operations Manager エージェントが動作していても、常にローカル アクションを実行する場合は、次のように入力します。

```
agsysdb -actions always
```

次の表は、Operations Manager への情報の送信とローカル アクションの実行に関する設定をリストしたものです。

表 10　Operations Manager への情報の送信とローカル アクションの実行に関する設定

フラグ	Operations Manager エージェントが動作している場合	Operations Manager エージェントが動作していない場合
Operations Manager フラグ		
off	アラート通知が Operations Manager に送信されません。	アラート通知が Operations Manager に送信されません。
on	アラート通知が Operations Manager に送信されます。	アラート通知が Operations Manager に送信されません。
ローカル アクション フラグ		
off	ローカル アクションが実行されません。	ローカル アクションが実行されません。
always	Operations Manager エージェントが動作していても、ローカル アクションが実行されます。	ローカル アクションが実行されます。
on	ローカル アクションが Operations Manager に送信されます。	ローカル アクションが実行されます。

アラーム処理のエラー

アラーム送信時に起きた最後のエラーは、agdb に記録されます。agdb の内容を参照するには、次のように入力します。

```
agsysdb -l
```

次の情報が表示されます。

```
PA alarming status:

OVO messages : on      Last Error :none
Exec Actions :on
Analysis system: <hostname>, Key=<ip address>
PerfView : no Last Error : <error number>
SNMP : yes Last Error : <error number>
```

アラームの履歴データの分析

ログ ファイル データ内のアラーム状態を検索するには、utility プログラムの analyze コマンドを使用します (第 5 章の「utility のコマンド」を参照してください)。ログ ファイル内の履歴データを alarmdef ファイル内のアラーム定義と比較して、どのアラーム状態がトリガされているかを調べるため、この処理は前述のリアルタイム アラームの処理とは異なります。

履歴データのアラーム情報の例

次の例は、履歴データのアラーム状態を分析するときにレポートされる内容を示しています。

最初の例では、START 文、END 文、REPEAT 文がアラーム定義で定義されています。アラーム開始イベントは、指定の継続時間中、アラームがそのすべての条件を満たすたびにリストされます。これらの条件が満たされなくなると、アラーム終了イベントがリストされます。最初のアラームが終了する前に別のアラームが発生するほど長い期間アラーム条件が満たされている場合、繰り返しイベントがリストされます。

リストされる各イベントは、日時、アラーム番号、アラーム イベントを示します。EXEC の動作は実行されませんが、要求されたすべてのパラメータ置換と共に所定の位置にリストされます。

```
05/10/99 11:15  ALARM [1] START
CRITICAL: CPU test 99.97%
05/10/99 11:20  ALARM [1] REPEAT
WARNING: CPU test 99.997%
05/10/99 11:25  ALARM [1] END
RESET: CPU test  22.86%
EXEC: end.script
```

カラー ワークステーションを使用している場合、次の出力が強調表示されます。

CRITICAL **文は赤色で表示されます。**

WARNING **文はシアン（緑がかった青）で表示されます。**

次の例は、アラーム イベントがリストされた後に表示されるアラームの要約を示しています。最初のカラムにはアラーム番号、2 番目のカラムにはアラーム状態が起きた回数、3 番目のカラムにはアラーム状態の全継続時間がリストされます。

```
Performance Alarm Summary:

  Alarm   Count   Minutes
      1     574      2865
      2       0         0

Analysis coverage using "alarmdef":
Start: 05/04/99 08:00    Stop: 05/06/99 23:59
```

```
Total time analyzed:  Days: 2  Hours: 15  Minutes: 59
```

アラーム定義の構成要素

定義した 1 つ以上の条件が指定した継続時間以上続くと、アラームが発生します。アラーム定義には、アラームの開始時または終了時に実行する動作を含めることができます。

条件とは、複数の項目間での比較のことです。比較される項目は、メトリック名、定数、変数のいずれかです。次に例を示します。

```
ALARM gbl_cpu_total_util > 95 FOR 5 MINUTES
```

アラームの開始時、終了時、繰り返し時に動作を実行するように指定できます。次のいずれかの動作を指定できます。

- Operations Manager にメッセージを送信するか、Network Node Manager に SNMP トラップを送信する ALERT
- UNIX コマンドを実行する EXEC
- utility プログラムを使用して処理したときに、stdout にメッセージを送信する PRINT

次に例を示します。

```
ALARM gbl_swap_space_util > 95 FOR 5 MINUTES
   START
      RED ALERT "Global swap space is nearly full"
   END
      RESET ALERT "End of global swap space full condition"
```

ブール論理、アプリケーションなどの複数インスタンス データによるループ、変数を使用すると、より複雑な動作を生成できます (詳細は、次の項の「アラーム構文のリファレンス」を参照してください)。

また、INCLUDE 文を使用すると、使用するその他のアラーム定義ファイルを識別できます。アラーム定義を小さいファイルに分割する場合などに、この操作を行います。

アラーム構文のリファレンス

この項では、アラーム構文で使用できる文について説明します。構文を使用して有用なアラーム定義を作成する方法の例については、alarmdef ファイルを参照してください。

アラーム構文

```
ALARM  condition [[AND,OR]condition]
  FOR duration [SECONDS, MINUTES]
  [TYPE="string"]
  [SERVICE="string"]
  [SEVERITY=integer]
  [START action]
  [REPEAT EVERY duration [SECONDS, MINUTES] action]
  [END action]
[RED, CRITICAL, ORANGE, MAJOR, YELLOW, MINOR, CYAN, WARNING,
  GREEN, NORMAL, RESET] ALERT message
EXEC "UNIX command"
PRINT message
IF condition
   THEN action
   [ELSE action]
{APPLICATION, PROCESS, DISK, LVOLUME, TRANSACTION, NETIF, CPU,
   FILESYSTEM} LOOP action
INCLUDE "filename"
USE "data source name"
[VAR] name =  value
ALIAS name = "replaced-name"
SYMPTOM  variable [ TYPE = {CPU, DISK, MEMORY, NETWORK}]
  RULE condition  PROB probability
  [RULE condition PROB probability]
    .
    .
```

表記について

- 中かっこ ({ }) は、項目の 1 つを選択することを示します。
- 大かっこ ([ ]) は、オプションの項目を示します。
- 大かっこまたは中かっこ内でコンマによって区切られている項目はオプションです。1 つだけ選択してください。
- 斜体は、置換する変数名を示します。
- 構文キーワードは、すべて大文字です。

共通要素

次に説明する要素はアラーム構文のいくつかの文で使用されます。

- コメント
- 複合文

- 条件
- 定数
- 式
- メトリック名
- メッセージ

コメント

コメントの前には、二重スラッシュ (//) またはポンド記号 (#) を付けます。どちらの場合も、コメントはその行の終わりで終了します。次に例を示します。

any text or characters

または

// any text or characters

複合文

複合文により、文のリストを単一文として実行できます。複合文とは、中かっこ ({ }) 内の文のリストのことです。複合文は、IF 文や LOOP 文、ALARM 文の START、REPEAT、END の各節と共に使用します。複合文には、ALARM 文および SYMPTOM 文は含められません。

```
{
 statement
 statement
}
```

次の例では、highest_cpu により、変数が定義されています。highest_cpu 値は保存され、さらに大きな highest_cpu 値がこの highest_cpu 値を超えたときにのみユーザーに通知します。

```
highest_cpu = highest_cpu
IF  gbl_cpu_total_util > highest_cpu  THEN
  // Begin compound statement
  {
  highest_cpu = gbl_cpu_total_util
  ALERT "Our new high CPU value is ", highest_cpu, "%"
  }
  // End compound statement
```

条件

条件は、2 つの項目間の比較として定義されます。

```
item1 {>, <, >=, <=, ==, !=}item2
  [AND, OR[item3 {>, <, >=, <=, ==, !=}item4]]
```

ここで、「==」は「等号」を、「!=」は「非等号」を意味します。

条件は、ALARM 文、IF 文、SYMPTOM 文で使用されます。項目は、メトリック名、数値定数、引用符で囲まれた英数字の文字列、別名、変数のいずれかです。英数字の項目を比較するときは、== または != のみを演算子として使用できます。

定数

定数は、数値または英数字です。英数字の定数は、二重引用符で囲む必要があります。次に例を示します。

```
345
345.2
"Time is"
```

定数は、式や条件で有用です。たとえば、次のようにメトリックが高すぎるときにアラームを発生させるには、そのメトリックを条件内の定数値と比較します。

```
gbl_cpu_total_util > 95
```

式

演算式は、複数のオペランドに関して 1 つ以上の算術演算を実行します。数値を使用する場合、いつでも式を使用できます。適切な算術演算子は、次のとおりです。

```
+, -, *, /
```

かっこを使用すると、最初に評価する式の部分を制御できます。

次に例を示します。

```
Iteration + 1
gbl_cpu_total_util - gbl_cpu_user_mode_util
( 100 - gbl_cpu_total_util ) / 100.0
```

メトリック名

アラーム定義でメトリック名を指定する場合、メトリックの現在の値が置き換えられます。メトリック名は、メトリックの定義に示されるとおりに正確に入力する必要があります。ただし、大文字と小文字を区別する必要はありません。メトリックの定義については、『Performance Collection Component Dictionary of Operating Systems Performance Metrics』を参照してください。

メトリックが、SCOPE データ ソース以外のデータ ソースのものである場合 (DSI メトリックなど) は、メトリックの完全修飾名を使用することをお勧めします。

メトリックの完全修飾名を指定するためのフォーマットは、次のとおりです。

```
data_source:instance(class):metric_name
```

SCOPE データ ソース内のグローバル メトリックは、修飾が不要です。次に例を示します。

```
metric_1
```

SCOPE データ ソースで定義されている各アプリケーションに使用可能なアプリケーション メトリックには、アプリケーション名が必要です。次に例を示します。

```
application_1:metric_1
```

アプリケーション、プロセス、ディスク、netif、トランザクション、lvolume、CPU、ファイル システムなどの種類の複数インスタンス データの場合、LOOP 文を使用している場合を除いて、メトリックをデータ型と関連付ける必要があります。この方法は、データ型名を指定し、その後にコロンを入力し、続いてメトリック名を入力します。たとえば、`other_apps:app_cpu_total_util` は、アプリケーション `other_apps` の合計 CPU 使用率を指定します。

- メトリックの完全修飾インスタンスを複数指定し、別名のインスタンス内で使用するときに、別名の 1 つにクラス識別名がある場合、次の例で示される構文を使用することをお勧めします。

```
alias my_fs="/dev/vg01/lvol1(LVOLUME)"
alarm my_fs:LV_SPACE_UTIL > 50 for 5 minutes
```

空白を含むアプリケーション名を使用する場合、空白をアンダースコア (_) に置換する必要があります。たとえば、application 1 は application_1 に変更する必要があります。特殊文字を含む名前や、大文字小文字に意味がある名前の使い方の詳細は、173 ページの「ALIAS 文」を参照してください。

ディスク「other」とアプリケーション「other」がある場合、インスタンスと共にクラスも指定する必要があります。

```
other (disk):metric_1
```

抽出ログ ファイル内のグローバル メトリックは、次のように指定します (ここで scope_extract はデータ ソース名です)。

```
scope_extract:application_1:metric_1
```

DSI メトリックは、次のように指定します。

```
dsi_data_source:dsi_class:metric_name
```

- 特殊文字 (アスタリスクなど) を含むメトリック名はいずれも、指定する前に別名を指定する必要があります。

メッセージ

メッセージとは、PRINT 文または ALERT 文によって送信される情報です。メッセージは、引用符で囲まれた英数字の文字列、数値定数、式、変数の任意の組み合わせで構成されます。メッセージ内の要素は、コンマによって区別されます。次に例を示します。

```
RED ALERT "cpu utilization=", gbl_cpu_total_util
```

数値定数、メトリック、式では、幅 (全桁数) と小数の桁数を設定できます。*width* はフォーマットするフィールドの幅を指定します。*decimals* は使用する小数の桁数を指定します。数値は右揃えです。- (マイナス記号) は左揃えを指定します。英数文字列は必ず左揃えです。次に例を示します。

```
metric names [|[-]width[|decimals]]
gbl_cpu_total_util|6|2   formats as '100.00'
(100.32 + 20)|6          formats as '   120'
gbl_cpu_total_util|-6|0  formats as '100   '
gbl_cpu_total_util|10|2  formats as '    99.13'
gbl_cpu_total_util|10|4  formats as '  99.1300'
```

ALARM 文

ALARM 文は、1 つの条件または条件一式と、その条件が真になるための継続時間を定義します。ALARM 文では、アラーム状態の開始時、繰り返し時、終了時に実行される動作を定義できます。アラームとして定義できる条件またはイベントには、次のものがあります。

- グローバル スワップ スペースが、5 分間ほとんどいっぱいになっている場合
- メモリ ページング率が、1 インターバルについて高すぎる場合
- CPU が最後の 10 分間に 75% の利用率で動作している場合

構文

```
ALARM condition [[AND,OR]condition]
  FOR duration{SECONDS, MINUTES}
  [TYPE="string"]
  [SERVICE="string"]
  [SEVERITY=integer]
  [START action]
  [REPEAT EVERY duration {SECONDS, MINUTES} action]
  [END action]
```

- **ALARM** 文は、トップレベルの文である必要があります。ALARM 文は別の文にネストできません。ただし、いくつかの ALARM 条件を単一の ALARM 文に含めることはできます。複数の条件が **AND** でリンクされている場合、アラームをトリガするには、すべての条件が真になる必要があります。複数の条件が **OR** でリンクされている場合、いずれか 1 つの条件によってアラームがトリガされます。
- **TYPE** は、引用符で囲まれた 38 文字以内の文字列です。アラームを送信している場合は、**TYPE** を使用するとアラームを分類したり、使用するグラフ テンプレートの名前を指定できます。
- **SERVICE** は、引用符で囲まれた 200 文字以内の文字列です。ITO 5.0 の ServiceNavigator を使用している場合、ServiceNavigator で定義したサービスと Performance Collection Component のアラームをリンクできます (『HP Operations ServiceNavigator Concepts and Configuration Guide』を参照してください)。

  ```
  SERVICE="Service_id"
  ```

- **SEVERITY** は、0 から 32767 までの整数です。
- **START**、**REPEAT**、**END** は、アラーム条件が満たされたとき、再び満たされたとき、終了したときに起きる動作を指定するために使用するキーワードです。ALARM 文では、**START**、**REPEAT**、**END** のうち少なくとも 1 つを必ず使用します。これらの各キーワードの後には、*action* が続きます。
- *action* − ALARM START、REPEAT、END のいずれかと共に最も頻繁に使用する動作は、ALERT 文です。EXEC 文を使用すると、メッセージを送信したりバッチ ファイルを実行できます。また、`utility` プログラムを使用して履歴ログ ファイルを分析している場合は、PRINT 文も使用できます。別の ALARM を除き、いずれの構文も正しい構文です。

  START、REPEAT、END の各動作は、複合文になります。たとえば、複合文を使用すると、ALERT と EXEC の両方を提供できます。
- *Conditions* − 条件は、2 つの項目間の比較として定義されます。

  ```
  item1 {>, <, >=, <=, ==, !=}item2
     [AND, OR[item3 {>, <, >=, <=, ==, !=}item4]]
  ```

  ここで、「==」は「等号」を、「!=」は「非等号」を意味します。

  項目は、メトリック名、数値定数、引用符で囲まれた英数字の文字列、別名、変数のいずれかです。英数字の項目を比較するときは、== または != のみを演算子として使用できます。

二次条件間で演算子の「OR」と「AND」を指定して、複合条件を使用できます。次に例を示します。

```
ALARM gbl_cpu_total_util > 90 AND
gbl_pri_queue > 1 for 5 minutes
```

- 二次条件間で演算子の「OR」と「AND」を指定して、複合条件を使用できます。次に例を示します。

  ```
  ALARM gbl_cpu_total_util > 90
  gbl_cpu_sys_mode_util > 50 for 5 minutes
  ```

  これにより、両方の条件が真になるとアラームが発生します。

  FOR *duration* **SECONDS, MINUTES** は、アラームをトリガするために、条件が真のままでなければならない期間を指定します。

  1 分未満の継続時間を指定する場合、特に複数のデータ ソースがシステム上にあるときは注意してください。各データ ソースのデータを非常に短いインターバルでポーリングする必要がある場合は、パフォーマンスが著しく低下します。継続時間は、アラーム条件で示されているメトリックの最長収集インターバルの倍数にする必要があります。

  scope データの場合、継続時間は 5 分ですが、プロセス データの継続時間は 1 分です。DSI データの場合、継続時間は 5 秒以上です。

- **REPEAT EVERY** *duration* **SECONDS, MINUTES** は、アラームが繰り返される前の期間を指定します。

使い方

アラーム サイクルは、AND でリンクされたすべてのアラーム条件、または OR でリンクされたアラーム条件の 1 つが、少なくとも指定の継続時間、真であった最初のインターバルに開始します。その時間に、アラーム ジェネレータは、START *action* を実行し、その後の各インターバルで REPEAT 条件を確認します。十分な時間が確認されると、REPEAT 節の *action* が実行されます (この処理は 1 つ以上のアラーム条件が偽になるまで続きます)。この処理がアラーム サイクルを完了し、END 文がある場合、それが実行されます。

アラームを通知するには、START 文および END 文で ALERT 文を使用します。END ALERT を指定しないと、アラーム ジェネレータは自動的にアラームを Operations Manager に送信し、SNMP トラップを Network Node Manager に送信します。

例

次の ALARM の例では、スワップ利用率が 5 分間高い場合に、レッド アラートを送信しています。これは、デフォルトの `alarmdef` ファイルのアラーム状態と類似しています。この例を `alarmdef` ファイルに追加する場合は、必ずデフォルトのアラーム状態を削除してください。削除しないと、その後のアラート メッセージが混乱する場合があります。

```
ALARM  gbl_swap_space_util > 90 FOR 5 MINUTES
    START
      RED ALERT "swap utilization is very high "
   REPEAT EVERY 15 MINUTES
      RED ALERT "swap utilization is still very high "
    END
      RESET ALERT "End of swap utilization condition"
```

この ALARM の例では、gbl_swap_space_util メトリックをテストして、それが 90 より大きいかどうかを確認しています。アラーム ジェネレータの構成方法により、ALERT を SNMP トラップを介して Network Node Manager に送信したり、メッセージとして Operations Manager に送信できます。

REPEAT 文は、15 分ごとに gbl_swap_space_util 条件を確認します。メトリックが 90 より大きい間は、REPEAT 文が 15 分ごとにメッセージ「swap utilization is still very high」を送信します。

gbl_swap_space_util 条件が 90 以下になると、RESET ALERT 文がメッセージ「End of swap utilization condition」と共に送信されます。

次の例では、ALARM 文での複合動作を定義しています。この例は、イベント発生時にメッセージを送信する方法を示しています。

```
ALARM gbl_cpu_total_util > 90 FOR 5 MINUTES
  START
  {
    RED ALERT "Your CPU is busy."
    EXEC "echo 'cpu is too high'| mailx root"
  }
  END
     RESET ALERT "CPU no longer busy."
```

ALERT は、Network Node Manager への SNMP トラップの送信または Operations Manager へのメッセージの送信をトリガできます。EXEC は、アラーム ジェネレータの構成方法により、Performance Collection Component システム上のローカル アクションとしてメール メッセージの送信をトリガできます。

Operations Manager エージェントが動作している場合、デフォルトにより、ローカル アクションは実行されません。代わりに、ローカル アクションはメッセージとして Operations Manager に送信されます。

次の 2 つの例は、複数の条件の使い方を示したものです。ALARM 文には、複数のテスト条件を指定できます。この場合、送信される ALERT に対してそれぞれの文が真である必要があります。

次の ALARM の例では、gbl_cpu_total_util メトリックと gbl_cpu_sys_mode_util メトリックをテストしています。両方の条件が真である場合、RED ALERT 文がレッド アラートを送信します。テスト条件のいずれかが偽になると、RESET が送信されます。

```
ALARM gbl_cpu_total_util > 90
  AND gbl_cpu_sys_mode_util > 50 FOR 5 MINUTES
START
  RED ALERT "CPU busy and Sys Mode CPU util is high."
END
  RESET ALERT "The CPU alert is now over."
```

次の ALARM の例では、gbl_cpu_total_util メトリックと gbl_cpu_sys_mode_util メトリックをテストしています。いずれかの条件が真である場合、RED ALERT 文がレッド アラートを送信します。

```
ALARM gbl_cpu_total_util > 90
  OR
      gbl_cpu_sys_mode_util > 50 FOR 10 MINUTES
START
  RED ALERT "Either total CPU util or sys mode CPU high"
```

異なるインターバルで記録されるメトリックを、同じアラームで使用しないように注意してください。たとえば、次のように文内のグローバル メトリック (5 分のインターバルで記録) の値に基づいて、プロセス (1 分のインターバルで記録) についてループしないようにしてください。

```
IF global_metric THEN
   PROCESS LOOP...
```

異なるインターバルは、予想どおりに同期しないため、結果は無効になります。

GlancePlus の場合、すべてのプロセスに対してアラームを送信するには、プロセス ループ内部のプロセス メトリックを使用してください。

ALERT 文

ALERT 文により、Network Node Manager、Operations Manager のいずれかにメッセージを送信できます。ALERT 文は、ALARM 内での処理として最も頻繁に使用されます。また、IF 文内で ALERT 文を使用すると、継続時間の経過後ではなく、条件の検出後直ちにメッセージを送信できます。ALARM 文または IF 文以外で ALERT を使用すると、メッセージは各インターバルで送信されます。

構文

[RED, CRITICAL, ORANGE, MAJOR, YELLOW, MINOR, CYAN,
WARNING, GREEN, NORMAL, RESET] *ALERT message*

- **RED** は **CRITICAL** と、**ORANGE** は **MAJOR** と、**YELLOW** は **MINOR** と、**CYAN** は **WARNING** と、**GREEN** は **NORMAL** と同義です。これらのキーワードは、アラーム シンボルをアラーム条件と関連付けられた色に変えます。
- **RESET** − ALARM 条件が終了すると、メッセージ付きの **RESET** ALERT を送信します。アラーム定義にリセットを定義していない場合は、ALARM 条件が終了すると、メッセージなしの **RESET** ALERT が送信されます。
- *message* − メッセージの作成に使用される文字列と数値の組み合わせです。数値のフォーマットは、[|[-]*width*[|*decimals*]] パラメータで設定できます。*width* はフォーマットするフィールドの幅を指定します。*decimals* は使用する小数の桁数を指定します。数値は右揃えです。- (マイナス記号) は左揃えを指定します。英数文字列は必ず左揃えです。

使い方

アラーム ジェネレータの構成方法により、ALERT は Network Node Manager への SNMP トラップの送信や、Operations Manager へのメッセージの送信をトリガできます。Operations Manager に送信されるアラート メッセージについては、WARNINGS がメッセージ ブラウザに青色で表示されます。

例

次に代表的な ALERT 文を示します。

```
RED ALERT "CPU utilization = ", gbl_cpu_total_util
```

Network Node Manager がある場合は、この文によって、Network Node Manager の Alarm Browser ウィンドウに重要度が危険域のアラームが作成されます。

EXEC 文

EXEC 文により、ローカル システム上で実行するシステム (UNIX または Windows) コマンドを指定できます。たとえば、EXEC 文を使用すると、一定の条件が満たされるたびに IT 管理者にメールを送信できます。

EXEC は、ALARM 文または IF 文内で使用されるため、コマンドは指定の条件が満たされたときにのみ実行されます。EXEC 文を ALARM 文または IF 文以外で使用すると、予測できないインターバルで動作が実行されます。

構文

EXEC　" *システム コマンド*"

*システム コマンド*はローカル システム上で実行するコマンドです。

EXEC 文では二重引用符 (") を使用しないようにしてください。これは、perfalarm がアラームを HPOM に送信できなくなるためです。代わりに引用符 (') を使用してください。次に例を示します。

```
EXEC "echo 'performance problem detected' "
EXEC "mkdir c:\\directory\\filename"
```

EXEC 文の構文には、二重引用符に囲まれたファイルのパス名が必要です。ただし、パス名にスペースが含まれている場合、そのパス名を引用符で囲んでから、二重引用符で囲む必要があります。

例 :

```
EXEC "'C:\\Program Files\\Mail Program\\SendMail.exe'"
```

EXEC 文のシステム コマンドの引数に引用符が含まれている場合、EXEC 文でそのコマンドを実行中に、引用符 (') の最初のペアは二重引用符 (") に変換されるため、そのプログラムを引用符で囲む必要があります。

例 :

```
EXEC "'echo' 'test execution'"
```

前述の例では、echo はプログラムであり、引用符に囲まれた引数が含まれているため (この場合、test execution)、このプログラムは引用符で囲まれています。さらに、EXEC 文の構文は、コマンドの文字列全体を二重引用符で囲む必要があります。

EXEC 文では二重引用符 (") を使用しないようにしてください。alarmgen で HPOM へのアラームの送信が失敗します。代わりに引用符 (') を使用してください。

次に例を示します。

```
EXEC "'echo' 'dialog performance problem'"
```

前述の例では、echo はプログラムであり、引用符で囲まれた引数が含まれているため (この場合、dialog performance problem)、このプログラムは引用符で囲まれています。さらに、EXEC 文の構文は、コマンドの文字列全体を二重引用符で囲む必要があります。

使い方

EXEC は、アラーム ジェネレータの構成方法により、ローカル システム上でローカル アクションをトリガできます。たとえば、ローカル アクションのオンとオフを切り替えられます。Operations Manager に情報を送信するようにアラーム ジェネレータを構成している場合、通常ローカル アクションは実行されません。

例

次の例では、gbl_disk_util_peak メトリックが 20 を超えたときに、EXEC 文が UNIX の mailx コマンドを実行します。

```
IF gbl_disk_util_peak > 20 THEN
  EXEC "echo 'high disk utilization detected'| mailx root"
```

次の例は、ネットワーク パケット率が 15 分間の平均で 1 秒あたり 1000 を超えたときに、メールをシステム管理者に送信する EXEC 文を示しています。

```
ALARM gbl_net_packet_rate > 1000 for 15 minutes
    TYPE = "net busy"
    SEVERITY = 5
    START
      {
      RED ALERT "network is busy"
      EXEC "echo 'network busy condition detected'| mailx root"
      }
    END
    RESET ALERT "NETWORK OK"
```

EXEC 文を頻繁に実行する場合、オーバーヘッドが高いコマンドまたはスクリプトと共に EXEC 文を使用するときは、注意が必要です。

アラーム ジェネレータはコマンドを実行し、そのコマンドが完了するまで続行せずに待機します。完了するのに長時間かかるコマンドは指定しないように注意してください。

PRINT 文

PRINT 文により、utility プログラムからその analyze 関数を使用してメッセージを印刷できます。アラーム ジェネレータは、PRINT 文を無視します。

構文

PRINT *message*

- *message* — メッセージを作成する文字列と数値の組み合わせです。数値のフォーマットは、[|[-]*width*[|*decimals*]] パラメータで設定できます。*width* はフォーマットするフィールドの幅を指定します。*decimals* は使用する小数の桁数を指定します。メッセージの英数字の構成要素は、引用符で囲む必要があります。数値は右揃えです。- (マイナス記号) は左揃えを指定します。英数文字列は必ず左揃えです。

例

```
PRINT "The total time the CPU was not idle is",
              gbl_cpu_total_time |6|2, "seconds"
```

この文を実行すると、次のようなメッセージが印刷されます。

```
The total time the CPU was not idle is 95.00 seconds
```

IF 文

IF-THEN 論理を使用して条件を定義するには、IF 文を使用します。IF 文は、ALARM 文内で使用します。ただし、IF 文は単独で使用するか、IF-THEN 論理が必要な alarmdef ファイル内の任意の場所で使用します。

ALARM 文以外で IF 文を指定する場合、その実行頻度は制御できません。

構文

IF *condition* **THEN** *action* **[ELSE** *action***]**

- **IF** *condition* — 条件は、2 つの項目間の比較として定義されます。

  ```
  item1 {>, <, >=, <=, ==, !=}item2
     [AND, OR[item3 {>, <, >=, <=, ==, !=}item4]]
  ```

  ここで、「==」は「等号」を、「!=」は「非等号」を意味します。

  項目は、メトリック名、数値定数、引用符で囲まれた英数字の文字列、別名、変数のいずれかです。英数字の文字列を比較するときは、== または != のみを演算子として使用できます。

- *action* — 任意の動作、または変数を設定します (この場合、ALARM は無効です)。

使い方

IF 文は *condition* をテストします。*condition* が真である場合、THEN の後の *action* が実行されます。*condition* が偽である場合、*action* はオプションの ELSE 節に依存します。ELSE 節が指定されている場合、その後に続く *action* が実行されます。ELSE 節が指定されていない場合、IF 文は何も行いません。

例

この例では、 CPU ボトルネックの兆候が計算され、 ボトルネックが発生する可能性により、シアンまたは赤色の ALERT が定義されます。アラーム ジェネレータの構成方法に応じて、ALERT は Network Node Manager への SNMP トラップの送信や、CPU の利用率を伴った「End of CPU Bottleneck Alert」メッセージの Operations Manager への送信をトリガできます。

```
SYMPTOM CPU_Bottleneck > type=CPU
     RULE gbl_cpu_total_util  > 75  prob 25
     RULE gbl_cpu_total_util  > 85  prob 25
     RULE gbl_cpu_total_util  > 90  prob 25
     RULE gbl_cpu_total_util  >  4  prob 25
     ALARM CPU_Bottleneck > 50 for 5 minutes
       TYPE="CPU"
       START
```

```
  IF CPU_Bottleneck > 90 then
    RED ALERT "CPU Bottleneck probability= ",
              CPU_Bottleneck, "%"
  ELSE
    CYAN ALERT "CPU Bottleneck probability= ",
              CPU_Bottleneck, "%"
REPEAT every 10 minutes
  IF CPU_Bottleneck > 90 then
    RED ALERT "CPU Bottleneck probability= ",
              CPU_Bottleneck, "%"
  ELSE
    CYAN ALERT "CPU Bottleneck probability= ",
              CPU_Bottleneck, "%"
END
   RESET ALERT "End of CPU Bottleneck Alert"
```

異なるインターバルで記録されるメトリックを、同じ文で使用しないように注意してください。たとえば、次のように文内のグローバル メトリック (5 分間のインターバルで記録) の値に基づいて、プロセス (1 分間のインターバルで記録) についてループしないようにしてください。

```
IF global_metric THEN
  PROCESS LOOP ...
```

異なるインターバルは、予想どおりに同期しないため、結果は無効になります。

LOOP 文

LOOP 文は複数インスタンス データ型を調べ、各インスタンスに定義された *action* を実行します。

構文

```
{APPLICATION, PROCESS, LVOLUME, DISK, CPU, FILESYSTEM, TRANSACTION,
NETIF, LOGICAL}
LOOP
    action
```

- **APPLICATION**、**PROCESS**、**LVOLUME**、**DISK**、**CPU**、**FILESYSTEM**、**TRANSACTION**、**NETIF**、**LOGICAL** ― 複数インスタンス データを含む Performance Collection Component のデータ型です。
- *action* ― PRINT、EXEC、ALERT、変数の設定です。

使い方

LOOP 文がデータ型の各インスタンスを繰り返すと、メトリック値が変わります。たとえば、utility プログラムの analyze コマンドを使用している場合、次の LOOP 文は各アプリケーションの名前を stdout に出力します。

```
APPLICATION LOOP
  PRINT app_name
```

LOOP は、最高 5 レベルまで別の LOOP 文内にネストできます。

LOOP を実行するには、LOOP 文が LOOP 文で定義されている型と同じデータ型のメトリックを 1 つ以上参照する必要があります。

例

アクティブなすべてのアプリケーションを繰り返すときには、LOOP 文を使用します。

次の例では、各インターバルで CPU 利用率が最大のアプリケーションを検索する方法を示しています。

```
highest_cpu = 0
 APPLICATION loop
   IF app_cpu_total_util > highest_cpu THEN
    {
      highest_cpu = app_cpu_total_util
      big_app = app_name
     }
 ALERT "Application ", app_name, " has the highest cpu util at
",highest_cpu_util|5|2, "%"
 ALARM highest_cpu > 50
   START
     RED ALERT  big_app, " is the highest CPU user at ", highest_cpu, "%"
   REPEAT EVERY 15 minutes
     CYAN ALERT big_app, " is the highest CPU user at ", highest_cpu, "%"
   END
 RESET ALERT "No applications using excessive cpu"
```

INCLUDE 文

INCLUDE 文は、デフォルトの alarmdef ファイルと共に別のアラーム定義ファイルを含めるときに使用します。

構文

INCLUDE "*filename***"**

ここで、*filename* は別のアラーム定義ファイルの名前です。ファイル名は、必ず完全修飾名にします。

使い方

INCLUDE 文を使用すると、論理的に異質なアラーム定義一式を別々のファイルに分離できます。

例

たとえば、独立したファイルにトランザクション メトリックに関するいくつかのアラーム定義があり、そのファイルに次のような名前が付いていると仮定します。

```
trans_alarmdef1
```

alarmdef1 ファイルのアラーム定義に次の行を追加すると、alarmdef1 ファイルを使用できます。

```
INCLUDE "/var/opt/perf/trans_alarmdef1"
```

USE 文

デフォルトの SCOPE データ ソース以外のデータ ソースを参照する場合は、alarmdef ファイルのメトリック名の使い方を単純化するために、USE 文を追加できます。これにより、データ ソース名を使用せずにメトリック名を指定できます。

データ ソース名は datasources ファイルで定義する必要があります。無効なデータ ソース名が見つかると、alarmdef ファイルの構文の確認が失敗します。

alarmdef ファイルに USE 文があっても、USE 文に続くすべてのメトリック名が指定のデータ ソースから得られるわけではありません。

構文

USE "*datasourcename***"**

使い方

アラーム ジェネレータは、alarmdef ファイルの構文が有効であることを確認するときに、ファイル内で参照する順序に従ってすべてのデータ ソースの検索リストを作成します。メトリックの完全修飾名または USE 文が見つかると、Perfalarm はエントリを順番にこのデータ ソース検索リストに追加します。その後、このリストは完全に修飾されていないメトリック名を適切なデータ ソース名に適合させるために使用されます。USE 文は、データ ソースを perfalarm の検索リストに追加する便利な方法です。これにより、alarmdef ファイルのメトリック名は短縮できます。メトリック名の構文については、この章で前述している 159 ページの「メトリック名」を参照してください。

メトリック名をデータ ソースに適合させるための Perfalarm のデフォルトの動作では、最初に SCOPE データ ソースでメトリック名を検索します。perfalarm が alarmdef ファイルの最初のメトリック名を見つけると、暗黙的に指定されている USE "SCOPE" 文が実行されます。この機能により、alarmdef ファイルの SCOPE のメトリックが完全に修飾されていなくても参照できるように、SCOPE データ ソースに対してデフォルトの検索パスが設定されます。次のページに、この例を示します。

```
ALARM gbl_cpu_total_util > 80 FOR 10 MINUTES
     START RED ALERT "CPU utilization too high"
USE "ORACLE7"
ALARM ActiveTransactions >= 95 FOR 5 MINUTES
      START RED ALERT "Nearing limit of transactions for ORACLE7"
```

perfalarm が上記の文を含む alarmdef ファイルの構文を確認すると、メトリック「gbl_cpu_total_util」が見つかり、次にそのデータソースを検索します。Perfalarm のデータ ソース検索リストにまだデータ ソースがないため、暗黙的に指定されている USE "SCOPE" 文を実行し、SCOPE データ ソースでメトリック名を検索します。適合するメトリック名を見つけた後、引き続き perfalarm は alarmdef ファイルの残りのデータを確認します。

USE "ORACLE7" 文を見つけると、perfalarm はデータ ソース検索リストに ORACLE7 データ ソースを追加します。メトリック名 "ActiveTransactions" が見つかると、perfalarm は SCOPE データ ソースからデータ ソースのリストを順に検索します。SCOPE にそのメトリック名が含まれない場合、次に ORACLE7 データ ソースが検索され、適合するメトリック名を見つけます。

perfalarm により、適合するメトリック名がどのデータ ソースでも見つからなかった場合は、エラー メッセージが出力され、perfalarm が終了します。

デフォルトの検索動作を変更するには、メトリック名を参照する前に、alarmdef ファイルの先頭に USE 文を追加します。これにより、USE 文で指定されているデータ ソースが、SCOPE データ ソースの前に、データ ソース検索リストに追加されます。USE 文内のデータ ソースは、適合するメトリック名を見つけるために、SCOPE データ ソースの前に検索されます。次に、この例を示します。

USE 文によってデータ ソースが参照されると、データ ソースの順序を変更したり、検索リストからデータ ソースを削除することはできません。

```
USE "ORACLE7"
ALARM gbl_cpu_total_util > 80 FOR 10 MINUTES
      START RED ALERT "CPU utilization too high"
ALARM ActiveTransactions >= 95 FOR 5 MINUTES
      START RED ALERT "Nearing limit of
      transactions for ORACLE7"
```

上記の例では、alarmdef ファイル内の文の順序が変更されています。メトリック名が参照される前に USE "ORACLE7" 文が定義されるため、データ ソース検索リストの最初のデータ ソースとして ORACLE7 データ ソースが追加されます。perfalarm が最初のメトリック名 "gbl_cpu_total_util" を見つけると、暗黙的に指定されている USE "SCOPE" 文が実行されます。"gbl_cpu_total_util" メトリック名は完全修飾名ではないため、perfalarm は ORACLE7 からデータ ソースのリストを順に検索します。ORACLE7 にそのメトリック名が含まれない場合、次に SCOPE データ ソースが検索され、適合するメトリック名を見つけます。

引き続き perfalarm は alarmdef ファイルの残りのデータを確認します。"ActiveTransactions" メトリックが見つかると、perfalarm は ORACLE7 からデータ ソースのリストを検索します。適合するメトリック名を見つけた後、perfalarm は引き続き alarmdef ファイルの残りのデータを検索します。perfalarm により、適合するメトリック名 (完全に修飾されていないメトリック名) がすべてのデータ ソースで見つからなかった場合は、エラー メッセージが出力され、perfalarm が終了します。

複数のデータ ソースに同じメトリック名が含まれる場合、注意して USE 文を使用してください。perfalarm はデータ ソースのリストを順番に検索します。同じメトリック名を使用する異なるデータ ソースのアラーム条件を定義している場合、正しいデータ ソースからメトリックの値が検索されるように、データ ソース名でメトリック名を修飾する必要があります。次の例を参照してください。アラーム文内の各メトリック名には、データ ソースが含まれています。

```
ALARM ORACLE7:ActiveTransactions >= 95 FOR 5 MINUTES
   START RED ALERT "Nearing limit of transactions for ORACLE7"
ALARM FINANCE:ActiveTransactions >= 95 FOR 5 MINUTES
   START RED ALERT "Nearing limit of transactions for FINANCE"
```

VAR 文

VAR 文により、変数を定義し、その変数に値を割り当てることができます。

構文

[VAR] *name* **=** *value*

- *name* ― 変数名は必ず英字から始まり、英数字とアンダースコア文字を使用できます。変数名に大文字と小文字の区別はありません。
- *value* ― 値が英数字の文字列の場合は、引用符で囲む必要があります。

使い方

VAR は、値をユーザー変数に割り当てます。変数が事前に存在しない場合は、作成します。

定義した変数は、alarmdef ファイルの任意の場所で使用できます。

例

変数は、何かを割り当てることによって定義します。次の例では、*highest_CPU_value* に 0 の値を割り当てることにより、数値変数を定義しています。

```
highest_CPU_value = 0
```

次の例では、*my_name* に空の文字列値を割り当てることにより、英数字変数を定義しています。

```
my_name = ""
```

ALIAS 文

メトリック名の部分 (クラス、インスタンス、メトリックのいずれか) に大文字と小文字を区別する名前がある場合や、特殊文字を含む名前がある場合、ALIAS 文により別名に置換できます。ALIAS 文はこのような場合にのみ使用します。

構文

ALIAS *name* **= "***replaced-name***"**

- *name* ― 名前は必ず英字から始まり、英数字とアンダースコア文字を使用できます。
- *replaced-name* ― アラーム ジェネレータが一意に認識できるように、ALIAS 文で置換する必要がある名前です。

使い方

alarmdef ファイルの処理方法によって、メトリック名の一部 (クラス、インスタンス、メトリック名のいずれか) が、大文字と小文字を認識することで一意に識別される場合は、別名を作成する必要があります。また、特殊文字を含む名前についても、別名を作成する必要があります。たとえば、アプリケーション「BIG」と「big」がある場合、「big」に別名を定義して、別々のアプリケーションとして表示されるようにする必要があります。別名は、alarmdef ファイル内で、置換する名前の最初のインスタンスより前に定義する必要があります。

例

構文では特殊文字を使用できず、大文字と小文字が区別されないため、アプリケーション名「AppA」と「appa」を使用するとエラーが発生します。これは、処理がこれらの名前を区別できないためです。「AppA」の別名を定義して、一意に認識可能な名前を付けます。次に例を示します。

```
ALIAS appa_uc = "AppA"
 ALERT "CPU alert for AppA.util is",appa_uc:app_cpu_total_util
```

クラス識別名のある別名を使用する場合、インスタンス名とクラス名の両方が含まれる別名である必要があります。次の例は、インスタンス名が 'other'、クラス名が 'APPLICATION' の別名を示しています。

```
ALIAS my_app="other(APPLICATION)"
   ALERT my_app:app_cpu_total_util > 50 for 5 minutes
```

SYMPTOM 文

SYMPTOM により、条件一式に基づいて単一の変数値を設定する方法が提供されます。すべての条件が真である場合、必ずその確率値が SYMPTOM の変数値に追加されます。

構文

SYMPTOM *variable*
  RULE *condition* **PROB** *probability*
  [RULE *condition* **PROB** *probability***]**
      .
      .
      .

- **SYMPTOM** キーワードおよび **RULE** キーワードは、SYMPTOM 文でのみ使用し、別の構文では使用できません。SYMPTOM 文は、トップレベルの文でなければならず、他の文内にネストできません。SYMPTOM の後の他の文は、SYMPTOM に対応するすべての RULE 文が終了するまで、処理されません。
- *variable* は、この兆候の名前になる変数名です。SYMPTOM 文で定義された変数名は別の構文で使用できますが、それらの構文で変数値を変更しないようにしてください。
- **RULE** は SYMPTOM 文のオプションであり、独立して使用できません。**RULE** オプションは SYMPTOM 文内で必要な数だけ使用できます。SYMPTOM 変数は、各インターバルで規則に従って評価されます。
- *condition* は、2 つの項目間の比較として定義されます。

  ```
  item1 {>, <, >=, <=, ==, !=}item2
  [item3 {>, <, >=, <=, ==, !=}item4]
  ```

  ここで、「==」は「等号」を、「!=」は「非等号」を意味します。

  項目は、メトリック名、数値定数、引用符で囲まれた英数字の文字列、別名、変数のいずれかです。英数字の項目を比較するときは、== または != のみを演算子として使用できます。
- *probability* は数値定数です。真である各 SYMPTOM RULE の確率を加算して、SYMPTOM 値を作成します。

使い方

測定値と値との間の条件が真であるすべての確率の合計が、その兆候が起きている確率になります。

例

```
SYMPTOM CPU_Bottleneck
RULE gbl_cpu_total_util > 75  PROB 25
RULE gbl_cpu_total_util > 85  PROB 25
RULE gbl_cpu_total_util > 90  PROB 25
RULE gbl_run_queue      >  3  PROB 50
IF CPU bottleneck > 50 THEN
CYAN ALERT "The CPU symptom is: ", CPU_bottleneck
```

アラーム定義の例

次の例は、一般的なアラーム定義の使用例です。

CPU の問題の例

この例では、CPU 使用率が 5 分間にわたって 90 % を超え、CPU 実行待ち行列が 5 分間 3 を超えた場合に、アラーム ジェネレータの構成方法に応じて、Network Node Manager に SNMP トラップを送信したり、Operations Manager にメッセージを送信します。

```
ALARM gbl_cpu_total_util > 90 AND
  gbl_run_queue > 3 FOR 5 MINUTES
START
  CYAN ALERT "CPU too high at", gbl_cpu_total_util, "%"
REPEAT EVERY 20 MINUTES
{
  RED ALERT "CPU still to high at ", gbl_cpu_total_util, "%"
  EXEC "/usr/bin/pager -n 555-3456"
}
END
  RESET ALERT "CPU at ", gbl_cpu_total_util, "% - RELAX"
```

両方の条件が 20 分後も真のままであれば、Network Node Manager の Alarm Browser ウィンドウに重要度が危険域のアラームが作成されます。次にプログラムが実行され、システム管理者に注意を促します。

アラーム条件のいずれか 1 つが真でなくなった場合、アラーム シンボルが削除され、メッセージが送信されて、グローバル CPU 使用率、アラートの終了時間、RELAX への注意が表示されます。

スワップ利用率の例

この例では、スワップ スペースの利用率が 5 分間 95% を超えた場合、アラーム ジェネレータの構成方法により、ALERT は Network Node Manager への SNMP トラップの送信や、Operations Manager へのメッセージの送信をトリガできます。

```
ALARM gbl_swap_space_util > 95 FOR 5 MINUTES
  START
    RED ALERT "GLOBAL SWAP space is nearly full "
  END
    RESET ALERT "End of GLOBAL SWAP full condition"
```

時間に起因するアラームの例

アラームの状態がアクティブであるようにインターバルを指定できます。たとえば、正規のインターバルで実行するように指定されたシステムの保守作業を行う場合、通常の操作時間のアラーム状態と、それとは異なるセットでシステム保守時間のアラーム状態を指定できます。

この例では、8:00 AM から 5:00 PM の間にのみアラームはトリガされます。

```
start_shift = "08:00"
end_shift = "17:00"
ALARM gbl_cpu_total_util > 80
    TIME > start_shift
    TIME < end_shift for 10 minutes
  TYPE = "cpu"
```

```
START
  CYAN ALERT "cpu too high at ", gbl_cpu_total_util, "%"
REPEAT EVERY 10 minutes
  RED ALERT"cpu still too high at ", gbl_cpu_total_util, "%"
END
  IF time == end_shift then
  {
    IF gbl_cpu_total_util > 80 then
     RESET ALERT "cpu still too high, but at the end of shift"
  ELSE
     RESET ALERT  "cpu back to normal"
  ELSE
```

ディスク インスタンスのアラームの例

アラームは、特定のディスク インスタンス名と対応するメトリック名を指定すると、アラームを特定のディスクに生成できます。

アラーム構文の次の例では、特定のディスク インスタンスのアラームが生成されます。ディスク インスタンスで特殊文字が使用されている場合、別名を使用する必要があります。

```
ALIAS diskname=""
ALARM diskname:bydsk_phys_read > 1000 for 5 minutes
TYPE="Disk"
  START
     RED ALERT "Disk  "
  REPEAT EVERY 10 MINUTES
     CYAN ALERT "Disk  cyan alert"
  END
     RESET ALERT "Disk  reset alert"
```

アラーム定義のカスタマイズ

アラーム定義ファイル alarmdef でアラームを発生させる条件を指定します。Performance Collection Component の初期インストール時、alarmdef ファイルにはデフォルトのアラーム定義セットが書き込まれます。これらのデフォルトのアラーム定義は、そのまま使用できますが、必要に応じてカスタマイズもできます。

alarmdef ファイルは次の手順でカスタマイズします。

1 必要に応じてアラームの定義を改訂します。この章のアラーム定義構文の例を参照してください。

2 ファイルを保存します。

3 Performance Collection Component の utility プログラムを使用して、アラームの定義の妥当性を検査します。

   a **utility** と入力します。

   b プロンプトで次のように入力します。

   checkdef

   これにより、アラームの構文が確認され、ファイルに問題がある場合は、エラーまたは警告が表示されます。

4 新しいアラームの定義を有効にするために、次のように入力します。

   ovpa restart alarm

   または

   mwa restart alarm

   これにより、アラーム ジェネレータが終了して再起動し、カスタマイズした alarmdef ファイルを読み取ります。

Performance Collection Component の各システムに固有のアラーム定義セットを使用するか、グループ全体で同じアラーム定義セットを使用することによりシステムのグループの監視作業を標準化できます。

alarmdef ファイルが大きくなると、Performance Collection Component アラーム ジェネレータおよび utility プログラムが正しく動作しないことがあります。この問題は、システム リソースの空き容量に起因する場合があります。

パフォーマンス アラームについて習得するには、新しいアラーム定義の追加や、デフォルトのアラーム定義の変更を試すことが最良の方法といえます。

10 RTMA コンポーネントのアドバイザ

アドバイザ機能を使用するには、HP Ops OS Inst to Realtime Inst LTU または Glance Pak Software LTU を有効にする必要があります。

アドバイザ機能を使用すると、RTMA コンポーネントによって収集された一部のメトリックの値が設定したしきい値を超過した (または下回った) 場合に、アラームを生成、表示できます。アドバイザ機能は、**adviser** スクリプトと **padv** ユーティリティで構築されます。adviser スクリプトは、監視対象システムのパフォーマンスの低下の兆候が見られた際に、アラームを生成するといったルールを作るのに便利です。padv ユーティリティは、対象のシステムで adviser スクリプトを実行するのに使用します。

このトピックでは、RTMA コンポーネントで使用可能なアドバイザ機能を詳述します。GlancePlus ソフトウェアには、アドバイザ ユーティリティで使用する追加の機能が備わっています。GlancePlus ソフトウェアでのアドバイザ機能の使用方法については、*GlancePlus* のオンライン ヘルプを参照してください。

アラームと兆候

アラームを使用すると、メトリック条件を強調表示できます。adviser スクリプトでは、RTMA コンポーネントによって監視するメトリックのしきい値を定義できます。メトリックの値が設定したしきい値を超えると、RTMA コンポーネントはアラーム メッセージの形式でアラームを生成します。このメッセージは stdout の形で **padv** ユーティリティに送信されます。

指定した条件を満たした場合に常にアラームを生成するよう設定できます。アラームは指定した期間 (1 つのまたは複数のインターバル) に基づいて生成されます。

兆候は、システムのパフォーマンスに影響を与える条件を組み合わせたものです。

さまざまなメトリックを対応するしきい値で観察し、ボトルネックの原因となっている確率に値を追加することにより、アドバイザはボトルネックが発生する複合的な確率を表す値を計算します。

adviser スクリプトでの作業

padv コマンドを実行すると、HP Operations エージェント はコマンドで指定したスクリプトの走査を行い、必要なアクションを実行します。padv コマンドでスクリプト ファイルを指定しない場合、アドバイザ ユーティリティは以下のデフォルトのスクリプト ファイルから必要な情報を取得します。

- *Windows: %ovdatadir%*\perf\perfd
- *UNIX/Linux:* /var/opt/perf/perfd

オペレーティング システム固有の診断およびアクションを含むスクリプトを実行する場合は、以下のデフォルトのスクリプトを使用します。

- *Windows:* *%ovdatadir%*\perf\perfd\os*<OS の種類>*\adv
- *UNIX/Linux:* /var/opt/perf/perfd/os/*<OS の種類>*/adv

この例では、*<OS の種類>* はスクリプトを実行するノードのオペレーティング システムのタイプになります。

adviser スクリプトを実行した結果、以下の作業を実行できます。

- 生成されたアラームに基づいてシステムのステータスをテキスト ファイルに出力
- padv コマンドを実行したコマンド コンソールにシステムのリアルタイムのステータスを表示

アドバイザの使用

アドバイザ コンポーネントを使用してシステムのリアルタイムのヘルスを監視するには、次の手順を実行します。

1. ご使用の環境に合わせて adviser スクリプトを構成します。以下のディレクトリにはサンプルのスクリプトが用意されています。
   - *Windows オペレーティング システム:* *%ovinstalldir%*\examples\adviser
   - *UNIX および Linux オペレーティング システム:* /opt/perf/examples/adviser
2. スクリプトを実行するノードを特定します。
3. 特定したシステムで perfd プロセスが実行されていることを確認します。
4. 次のコマンドを実行します。

   padv -s *<script_name>* **-n** *<system_name>*

   指定したシステムで adviser スクリプトが開始され、スクリプト ファイルの構成に応じて結果が作成されます。

複数のシステム上での adviser スクリプトの実行

mpadv コマンドを使用すると、複数のシステム上で adviser スクリプトを実行できます。mpadv コマンドを使用するには、次の手順を実行します。

1. スクリプトを実行するノードを特定します。
2. 特定したすべてのシステムの名前をリストしたテキスト ファイルを作成します。
3. ローカル システムにテキスト ファイルを保存します。
4. ご使用の環境に合わせて adviser スクリプトを構成します。以下のディレクトリにはサンプルのスクリプトが用意されています。
   - *Windows オペレーティング システム:* *%ovinstalldir%*\examples\adviser
   - *UNIX および Linux オペレーティング システム:* /opt/perf/examples/adviser
5. 特定したシステムで perfd プロセスが実行されていることを確認します。

6 次のコマンドを実行します。

mpadv -l <*system_list_text_file*> **-s** <*script_name*>

<*system_list_text_file*> ファイルで指定したシステムで adviser スクリプトが開始され、スクリプト ファイルの構成に応じて結果が作成されます。

アドバイザ構文

アドバイザ構文は簡単なスクリプト言語で、アラームの設定および兆候の条件の定義が可能です。

デフォルトの構文ファイル、`adviser.syntax` は以下のディレクトリにあります。

- *Windows オペレーティング システム : %ovdatadir%*`\perf`
- *UNIX/Linux:* `/var/opt/perf`

構文ファイルを編集することで、独自のアラームと兆候を定義できます。

構文表記

- 中かっこ ({ }) は、項目の 1 つを選択することを示します。
- 大かっこ ([ ]) は、オプションの項目を示します。
- 大かっこまたは中かっこ内でコンマによって区切られている項目はオプションです。1 つだけ選択してください。
- 斜体は、置換する変数名を示します。
- アドバイザ構文のキーワードは大文字にする必要があります。

コメント

構文 :

 # [任意のテキストまたは文字列]

または

// [任意のテキストまたは文字列]

コメントの前には、二重スラッシュ (//) または # 記号を付けます。どちらの場合も、コメントはその行の終わりで終了します。

条件

条件は、2 つのメトリック名、ユーザー変数、数値定数間の比較として定義されます。

```
item1 {>, <, >=, <=, ==, !=} item2 [OR item3 \
            {>, <, >=, <=, ==, !=} item4]
```

または：

```
item1 {>, <, >=, <=, ==, !=} item2 [AND item3 \
            {>, <, >=, <=, ==, !=} item4]
```

（「==」は「等号」を、「!=」は「非等号」を意味します）

条件は、ALARM 文、IF 文で使用されます。2 つの数値メトリック、変数または定数を比較できるほか、2 つの文字列メトリック名、ユーザー変数、文字列定数にも使用できます。文字列の条件の場合は、== または != のみを演算子として使用できます。

二次条件間で演算子の OR と AND を指定して、複合条件を使用できます。

例：

```
gbl_swap_space_reserved_util > 95

proc_proc_name == "test" OR proc_user_name == "tester"

proc_proc_name != "test" AND
            proc_cpu_sys_mode_util > highest_proc_so_far
```

定数

定数は、英数字または数値です。英数字の定数は、二重引用符で囲む必要があります。数値定数には、整数または実数の 2 種類があります。整数定数には数字とオプションの符号標識を使用できます。実数定数には小数点も含めることができます

例：

345	整数
345.2	実数
"Time is"	英数字リテラル

式

式を使用すれば数値を評価できます。式は、条件またはアクション内で使用できます。

式には以下のものを含めることができます。

- 数値定数
- 数値メトリック名
- 数値変数
- 上記の算術比較
- 括弧を使用して上記をグループ化したものの組み合わせ

例：

```
Iteration + 1
3.1416
gbl_cpu_total_util - gbl_cpu_user_mode_util
( 100 - gbl_cpu_total_util ) / 100.0
```

アドバイザ構文内のメトリック名

アドバイザ構文内のメトリックはその位置に関わらず直接参照できます。アドバイザ構文には以下のタイプのメトリックを使用できます。

- グローバル メトリック (接頭辞 gbl_ または tbl_)
- アプリケーション メトリック (接頭辞 app_)
- プロセス メトリック (接頭辞 proc_)
- ディスク メトリック (接頭辞 bydsk_)
- CPU ごとのメトリック (接頭辞 bycpu_)
- ファイル システム メトリック (接頭辞 fs_)
- 論理ボリューム メトリック (接頭辞 lv_)
- ネットワーク インターフェイス メトリック (接頭辞 bynetif_)
- スワップ メトリック (接頭辞 byswp_)
- ARM メトリック (接頭辞 tt_ または ttbin_)
- PRM メトリック (接頭辞 prm_)
- ローカリティ ドメイン メトリック (接頭辞 ldom_)

LOOP 文のコンテキストには、プロセス、論理ボリューム、ディスク、ファイル システム、LAN、スワップの各メトリックのみが使用できます。

メトリックには英数字 (gbl_machine または app_name など) または数値データを含めることができますので、異なる種類の測定を反映させることができます。たとえば、以下のようにメトリック名の末尾は測定対象を示します。

- a _util メトリック。利用率をパーセンテージで測定
- a _rate メトリック。1 秒当りの単位を計測
- a _queue メトリック。リソースを待っているプロセスまたはスレッドの数を計測

特定のメトリックの計測単位が分からない場合は、メトリック定義ドキュメントを参照してください。

LOOP 文を使用しているとき以外は、アプリケーション メトリックに特定のアプリケーションを関連付ける必要があります。この方法は、アプリケーション名を指定し、その後にコロンを入力し、続いてメトリック名を入力します。たとえば、other_apps:app_cpu_total_util は、アプリケーション other_apps の合計 CPU 使用率を指定します。構文内でのアプリケーション メトリックの使用方法の詳細は、ALIAS 文の説明を参照してください。

parm ファイルによって定義されることから、アプリケーション名には特殊文字およびスペースを使用できます。構文でこれらの名前を使用するには (アプリケーション名を変数名の形と一致させる必要があります)、大文字と小文字を区別せずに名前を作成し、スペースをアンダースコアに変換します。つまり、「Other Apps」と定義されたアプリケーション名は、構文内では other_apps とも参照されます。特殊文字を使用して定義されたアプリケーション名の場合、ALIAS 文を使用して別名を指定する必要があります。

明示的に指定されている場合は、アプリケーション メトリックは構文内のあらゆる場所で参照可能となります。指定されていないアプリケーション メトリックは、LOOP 文のコンテキスト内でのみ参照されます。LOOP 文は、アプリケーション メトリックまたはプロセス メトリックを暗黙的に指定する反復文です。

LOOP 文のコンテキスト内でのみ、プロセス メトリックを参照できます。プロセスを明示的に参照する方法はありません。

出力リスト

出力リストは、正しくフォーマットされた式、メトリック名、ユーザー変数、定数の組み合わせです。正しいフォーマットについては、次の例を参照してください。

式の例 :

```
expression [|width[|decimals]]
```

メトリック名またはユーザー変数の例 :

```
metric names [|width[|decimals]]
```

または

user variables [|width[|decimals]]

メトリック名またはユーザー変数は英数字で指定します。

定数の例 :

定数にはフォーマット化は必要ありません。

フォーマット化の例 :

```
gbl_cpu_total_util|6|2    formats as  '100.00'
(100.32 + 20)|6    formats as     '   120'
gbl_machine|5      formats as         '7013/'
"User Label"       formats as       "User Label"
```

変数

変数は必ず英字から始まり、英数字とアンダースコア文字を含められます。変数では、大文字と小文字を区別しません。

変数は、何かを割り当てることにより定義します。次の例では、highest_CPU_value に 0 の値を割り当てることにより、数値変数を定義しています。

```
highest_CPU_value = 0
```

次の例では、my_name に null の文字列値を割り当てることにより、英数字変数を定義しています。

```
my_name = ""
```

アドバイザ構文

ALARM 文

ALARM 文を使用すると、定義した一部のイベントがシステムで発生した際に通知を行います。ALARM 文を使用すると、adviser スクリプトで padv コマンドの発信元コンソールにメッセージを送信して通知するよう設定できます。

構文：

```
ALARM  condition  [FOR duration {SECONDS, MINUTES, INTERVALS}]
  [condition [FOR duration {SECONDS, MINUTES, INTERVALS}]  ] ...

[START statement]

[REPEAT [EVERY duration [SECONDS, MINUTES, INTERVAL, INTERVALS]]
  statement]

[END statement]
```

ALARM 文は、トップレベルの文である必要があります。ALARM 文は別の文にネストできません。

ただし、いくつかの ALARM 条件を単一の ALARM 文に含めることはできます。この場合、トリガとなるためには、すべての条件が真となる必要があります。また、アラーム サイクル中の適切な時間に実行される COMPOUND 文も使用できます。

START、REPEAT、END は、ALARM 文のキーワードです。これらの各キーワードが文を指定します。ALARM 文には START、REPEAT、END が必要で、正しい順番にリストします。

アラーム サイクルは、すべてのアラーム条件、少なくとも指定の継続時間真であった最初のインターバルに開始します。その時間に、adviser スクリプトは、START 文を実行し、その後の各インターバルで REPEAT 条件を確認します。十分な時間が確認されると、REPEAT 節の文が実行されます。この処理は 1 つ以上のアラーム条件が偽になるまで続きます。この処理がアラーム サイクルを完了し、END 文が実行されます。

REPEAT 文で EVERY 仕様が省略されている場合、adviser スクリプトは各インターバルで REPEAT 文を実行します。

ALARM の例 : 通常の ALARM 文

次の ALARM の例では、セマフォ テーブルがほぼいっぱいになった場合のレッド アラートを設定しています。これは、デフォルト構文の定義済みアラームと類似しています。デフォルトを削除せずにこの設定を構文に追加すると、その後のアラート メッセージが混乱するため追加しないでください。

```
ALARM tbl_sem_table_util > 90 FOR 1 MINUTE

  START RED ALERT "Semaphore Table is nearly full"
```

```
  REPEAT EVERY 30 SECONDS
    RED ALERT "Semaphore Table still nearly full"

  END RESET ALERT "End of Semaphore Table full condition"
```

この ALARM の例では、メトリック tbl_sem_table_util をテストし 90 を超えているかどうかを確認します。超えている場合、RED ALERT 文が重要度 RED のメッセージを、padv コマンドの発信元コンソールに送信します。

REPEAT 文は、30 秒ごとに tbl_sem_table_util 条件を確認します。条件が 90 を超えている限り、REPEAT はアドバイザに対して RED ALERT 条件を保持することを通知し、「Semaphore Table still nearly full」メッセージを padv コマンドの発信元に送信します。

tbl_sem_table_util 条件が 90 以下になると、RESET ALERT 文が「End of Semaphore Table full condition」というメッセージを padv コマンドの発信元コンソールに表示します。

ALARM の例 : 複数の条件の使用

ALARM 文には、複数のテスト条件を指定できます。この場合、有効にするアラーム ボタンに対してそれぞれの文が真である必要があります。次に例を示します。

```
ALARM gbl_cpu_total_util > 90 FOR 2 MINUTES
  gbl_cpu_sys_mode_util > 50 FOR 1 MINUTES
    START RED ALERT
      "The CPU is busy and System Mode CPU utilization is high."
    END RESET ALERT "The CPU alert is now over."
```

この ALARM の例では、gbl_cpu_total_util メトリックと CPU_Bottleneck メトリックをテストしています。両方の条件が真である場合、RED ALERT 文が危険域アラートを設定します。テスト条件のいずれかが偽になると、RESET 文が実行されます。

ALARM の例 : Swap Space (スワップ スペース)

```
//GLOBAL SWAP ALARM
symp_swap_util = gbl_swap_space_used / gbl_swap_space_avail
ALARM symp_swap_util > 0.9
  START
    RED ALERT "GLOBAL SWAP space is nearly full"
  END RESET ALERT "GLOBAL SWAP space crisis is over"
```

新しい変数 symp_swap_util はスワップ利用率を表します。スワップ利用率が 90% を超えると、adviser スクリプトがアラームを作成します。次のインターバルに、symp_swap_util が 90% を下回ると、アラーム条件が偽となり、ALARM がリセットされます。

ALARM の例 : イエロー アラート

```
ALARM Symp_Global_Cpu_Bottleneck >  50 FOR 2 MINUTES

  START YELLOW ALERT "CPU Bottleneck probability= ",
    Symp_Global_Cpu_Bottleneck, "% for the last 2 minutes"
```

```
  REPEAT every 2 minutes
    YELLOW ALERT "CPU Bottleneck probability= ",
      Symp_Global_Cpu_Bottleneck, "% for the last 2 minutes"

  END
    RESET ALERT " CPU Bottleneck Yellow Alert over, probability=",
      Symp_Global_Cpu_Bottleneck, "%"
```

ALARM は、SYMPTOM 文の Symp_Global_Cpu_Bottleneck に定義されている SYMPTOM 変数をテストします。SYMPTOM 変数が 2 分間にわたって 50 を超えている場合、ALARM は YELLOW ALERT を padv コマンド コンソールに通知します。

ALARM 条件が偽となるまで、ALARM は 2 分おきに繰り返されます。偽となった場合、END 文が ALERT をリセットします。

ALARM の例 : CPU の障害

```
ALARM

  gbl_cpu_total_util > 90 FOR 30 SECONDS

  gbl_run_queue > 3 FOR 30 SECONDS

  START YELLOW ALERT "CPU AT ", gbl_cpu_total_util,
    "% at ", gbl_stattime

  REPEAT EVERY 300 SECONDS {
    RED ALERT "CPU AT ", gbl_cpu_total_util
    exec "/usr/bin/pager -n 555-3456"
  }

  END ALERT "CPU at ", gbl_cpu_total_util, "% at ",
    gbl_stattime, " - RELAX"
```

この例では、CPU 使用率が 30 秒間 90% を超え、CPU 実行待ち行列が 30 秒間 3 を超えた場合に、padv コマンド コンソールにイエロー アラートが生成されます。

両方の条件が真のままであれば、レッド アラートが生成され、システム管理者に注意を促すプログラムが呼び出されます。

ALERT 文

ALERT 文を使用すると、padv コマンド コンソールにメッセージを配置できます。ALARM が障害を検知した場合は、ALERT 文を実行して、指定した重要度のメッセージを padv コマンド コンソールに送信するよう設定できます。

ALERT 文は ALARM 文と組み合わせて使用できます。

構文：

```
[(RED or CRITICAL), (YELLOW or WARNING), RESET] ALERT printlist
```

RED は CRITICAL と、YELLOW は WARNING と同義です。

ALERT の例

次に ALERT 文の例を示します。

```
RED ALERT "CPU utilization = ", gbl_cpu_total_util,
  " at ", gbl_stattime
```

この文を実行すると、以下のようなメッセージが padv コマンド コンソールに書き込まれます。

```
CPU utilization = 85.6 at 14:43:10
```

ALIAS 文

ALIAS 文を使用すると、特殊文字または空白を含むアプリケーション名に変数を割り当てることができます。

構文：

```
ALIAS variable = "alias name"
```

ALIAS の例

構文では特殊文字や空白を使用できないため、下記の PRINT 文でアプリケーション名「other user root」を使用するとエラーが発生します。ALIAS を使用すれば、「other user root」や特殊文字や空白を持つ他の文字列も構文で使用できます。

```
ALIAS otherapp = "other user root"


PRINT "CPU for other root login processes is: ",
  otherapp:app_cpu_total_util
```

ASSIGNMENT 文

ASSIGNMENT 文を使用すると、ユーザー変数に数値、英数字、式を割り当てられます。

構文：

```
[VAR] variable = expression
[VAR] variable = alphaitem
[VAR] variable = alphaitem
```

ASSIGNMENT の例

ユーザー変数は、最初の割り当て時に、数値か英数字かを定義します。ASSIGNMENT 文では異なるタイプの変数を混在させることはできません。

この例は、英数字アプリケーション名を新しいユーザー変数に割り当てます。

```
myapp_name = other:app_name
```

この例は、最初に英数字と定義したユーザー変数 (例 1) に数値を割り当てているため、不正となります。

```
myapp_name = 14
```

この例は、数値を新しいユーザー変数に割り当てます。

```
highest_cpu = gbl_cpu_total_util
```

この例は、最初に数値と定義したユーザー変数 (例 3) に英数字リテラルを割り当てているため、不正となります。

```
highest_cpu = "Time is"
```

COMPOUND 文

COMPOUND 文は、IF 文や LOOP 文、ALARM 文の START、REPEAT、END の各節と共に使用します。COMPOUND 文を使用すると、リスト化した文を実行できます。

構文

```
{
statement
statement
}
```

COMPOUND 文を作成するには、中かっこ ({ }) 内に文をグループ化します。こうすることで、COMPOUND 文は構文内で単一の文として処理されます。

複合文には、ALARM 文および SYMPTOM 文は含められません。(COMPOUND は文のタイプであり、キーワードではありません。)

COMPOUND の例

```
highest_cpu = highest_cpu
IF gbl_cpu_total_util > highest_cpu  THEN
  // Begin compound statement
  {
    highest_cpu = gbl_cpu_total_util
    PRINT "Our new high CPU value is ", highest_cpu, "%"
  }
  // End compound statement
```

この例では、highest_cpu = highest_cpu により、変数 highest_cpu が定義されています。adviser スクリプトによって highest_cpu 値が保存され、さらに大きな値がこの highest_cpu 値を超えたときにのみユーザーに通知します。

この例では、highest_cpu = highest_cpu を highest_cpu = 0 で置き換えると、highest_cpu の値が各インターバルで 0 にリセットされます。

各インターバルでは、highest_cpu の値が通知されます。

EXEC 文

EXEC 文を使用すると、アドバイザ構文から UNIX コマンドを実行できます。たとえば、ある条件を満たすたびに MIS スタッフに対してメール メッセージを送信する場合などに、EXEC コマンドを使用します。

構文

```
EXEC printlist
```

結果の出力リストは、実行用としてオペレーティング システムに送信されます。

指定した EXEC コマンドは更新間隔ごとに 1 回実行されますので、オーバーヘッドの高くなるオペレーティング システムのコマンドまたはスクリプトと EXEC 文を併用する際には注意が必要です。

EXEC の例

次の例では、EXEC は各インターバルに UNIX の mailx コマンドを実行します。

```
EXEC "echo 'gpm mailed you a message' | mailx root"
```

次の例では、gbl_disk_util_peak メトリックが 20 を超えたときにのみ、EXEC が UNIX の mailx コマンドを実行します。

```
IF gbl_disk_util_peak > 20 THEN
  EXEC "echo 'gpm detects high disk utilization' | mailx   root"
```

IF 文

IF 文を使用すると、adviser スクリプト構文に定義した条件をテストできます。

構文 :

```
IF condition  THEN statement [ELSE statement]
```

IF 文 は condition をテストします。condition が真である場合、THEN の後の文が実行されます。condition が偽である場合、動作はオプションの ELSE 節に依存します。

ELSE 節が指定されている場合、その後に続く文が実行されます。ELSE 節が指定されていない場合、IF 文は何も行いません。文を COMPOUND 文にすれば、adviser スクリプトで複数の文を実行するように設定できます。

IF の例

```
IF gbl_cpu_total_util > 90 THEN
  PRINT "The CPU is running hot at: ", gbl_cpu_total_util
ELSE IF gbl_cpu_total_util < 20 THEN
```

```
PRINT "The CPU is idling at: ", gbl_cpu_total_util
```

この例では、IF 文が条件 (gbl_cpu_total_util > 90) をチェックします。条件が真の場合、「The CPU is running hot at:」が CPU の使用率と共に padv コマンド コンソールに表示されます。

条件 (gbl_cpu_total_util > 90) が偽の場合、ELSE IF は次の行に進み、条件 (gbl_cpu_total_util < 20) をチェックします。その条件が真の場合、「The CPU is idling at: 」が CPU の使用率と共に padv コマンド コンソールに表示されます。

LOOP 文

LOOP 文を使用すると、システムに関する情報を確認できます。たとえば、CPU 使用率の最も高いプロセス、最も頻繁に使用されているスワップ領域などを確認できます。この情報を確認するには、LOOP 文のほか、情報収集対象のシステム条件にメトリック名を使用するそれぞれの文を使用します。

構文 :

```
{APPLICATION, APP, CPU, DISK, DISK_DETAIL, FILESYSTEM, FS, FS_DETAIL, LAN,
LOGICALVOLUME, LV, LV_DETAIL, NETIF, NFS, NFS_BYSYS_OPS, NFS_OP, PRM,
PRM_BYVG, PROCESS, PROC, PROC_FILE, PROC_REGION, PROC_SYSCALL, SWAP,
SYSTEMCALL, SC, THREAD, TRANSACTION, TT, TTBIN, TT_CLIENT, TT_INSTANCE,
TT_UDM, TT_RESOURCE, TT_INSTANCE_CLIENT, TT_INSTANCE_UDM, TT_CLIENT_UDM,
LDOM, PROC_LDOM}

LOOP statement
```

LOOP は、最高 5 レベルまで他の構文内にネストできます。ループの各反復で実行される文のブロックを含む COMPOUND 文として設定することもできます。BREAK 文を指定することで、LOOP 文からエスケープできます。

特定のデータを収集する構文内に LOOP 文を設定しても、システムに対応するメトリック データがない場合は、adviser スクリプトは LOOP をスキップし、別の構文または指示文に進みます。たとえば、LOGICAL VOLUME LOOP を定義したが、システムに論理ボリュームがない場合、adviser スクリプトは LOGICAL VOLUME LOOP をスキップし、次の構文に進みます。

プラットフォームに存在しないループの場合はエラーが生成されます。

各インターバルで LOOP 文が繰り返されると、文で使用されるメトリックの値が変わります。たとえば、以下の LOOP 文はシステム上のアクティブなアプリケーションごとに 1 回 PRINT 文を実行するため、各アプリケーションの名前が出力されます。

APP LOOP

```
PRINT app_name
```

HP_UX 11.23 のようにスレッド化されたオペレーティング システムでは、adviser スクリプトが THREAD LOOP に対応しています。特定のプロセスの各スレッドを調査するために、スレッド ループをプロセス ループ内にネストできます。スレッド ループ内の PROC_ メトリックを参照している場合、予期しない結果 (スレッド情報) を返すことがあります。

スレッド ループはプロセス ループの外部に設定することもできます。この場合、システム上のアクティブなスレッドをすべて調査します。スレッド ループ内ではプロセス ループをネストしないでください。

LOOP 文は各インターバルで開始されることから、パフォーマンスに影響をおよぼす原因となりますので、慎重に使用してください。特にネストされた LOOP 文の使用の際には注意してください。

APPLICATION LOOP の例

アクティブなすべてのアプリケーションを繰り返すときには、APPLICATION LOOP 文を使用します。

APPLICATION LOOP では、グローバル (gbl_)、テーブル (tbl_)、アプリケーション (app_) メトリックを使用できます。

次の例では、APPLICATION LOOP を使用して、あるインターバルにおいて最も CPU 使用率の高いアプリケーションを検索します。

```
big_app = ""
highest_cpu = 0
APPLICATION LOOP
  IF (app_cpu_total_util > highest_cpu) THEN
  {
    highest_cpu = app_cpu_total_util
    big_app = app_name
  }
  IF (highest_cpu > 20) THEN
    YELLOW ALERT "The application ", big_app,
      " is the highest CPU user at", highest_cpu, "%"
```

アプリケーションが見つかると、adviser スクリプトは CPU の値が 20 を超える app_name と CPU の値と共にメッセージを padv コマンド コンソールに書き込みます。

CPU LOOP の例

システム上の CPU に関するデータを繰り返すときには、CPU LOOP 文を使用します。CPU LOOP では、グローバル (gbl_)、テーブル (tbl_)、CPU ごと (bycpu_) のメトリックを使用できます。

この例では、システムの各 CPU の CPU 使用率を出力します。

```
Print "----------", glb_stattime, "----------"
CPU LOOP
  PRINT "CPU # ", bycpu_id, " used ", bycpu_cpu_total_util, " % CPU"
```

以下は 2 つの CPU を持つシステムにおける、2 つのインターバルの出力結果です。

```
----------10:52:01----------
CPU #             0 used   0.6 % CPU
CPU #             1 used   3.4 % CPU
----------10:52:11----------
CPU #             0 used   0.4 % CPU
```

```
CPU #               1 used    2.3 % CPU
```

DISK LOOP の例

構成済みのディスク デバイスに対して処理をループするときには、DISK LOOP 文を使用します。この LOOP を使用すると、adviser スクリプトは [IO by Disk (ディスク別の IO)] ウィンドウに表示されている特定のディスク情報をチェックします。DISK LOOP では、グローバル (gbl_)、テーブル (tbl_)、ディスクごとのメトリックを使用できます。

この例では、システム上の各ディスクの物理的な書き込み率を出力します。

```
PRINT "---------------------", gbl_stattime, "--------------------"

DISK LOOP

  PRINT bydsk_devname, " write rate: ", bydsk_phys_write_rate
```

以下は 3 つのディスクを持つシステムにおける、2 つのインターバルの出力結果です。

```
---------------------11:00:23-------------------------

/dev/hdisk0          write rate:    2.4

/dev/hdisk1          write rate:    0.0

/dev/cd0             write rate:    0.0


---------------------11:00:33-------------------------

/dev/hdisk0          write rate:    0.0

/dev/hdisk1          write rate:    0.0

/dev/cd0               write rate:     0.0
```

FILE SYSTEM LOOP の例

FILE SYSTEM LOOP は、構成済みのファイル システムに対して処理をループし、adviser スクリプトから [IO By File System (ファイルシステム別の IO)] ウィンドウに表示されている情報に関するレポートを行うよう設計されています。FILE SYSTEM LOOP では、グローバル (gbl_)、テーブル (tbl_)、ファイル システムごとの IO (fs_) メトリックを使用できます。

以下の例では、3 つのデバイスを持つシステム上の各ファイル システム デバイスで使用されている容量をレポートします。

```
PRINT "-------------------", gbl_stattime, "----------------------"

FS LOOP

  PRINT fs_devname, " is ", fs_space_util, "% full at ",

    fs_max_size," megabytes"
```

3 つのファイル システムを持つシステムにおける、2 つのインターバルの出力結果は以下のようになります。

```
---------------------11:11:28-------------------------

/dev/hd4             is   77.9% full at       32 megabytes

/dev/hd2             is   94.9% full at      928 megabytes

/dev/hd9var          is   93.9% full at       56 megabytes
```

```
---------------------11:11:38-------------------------
/dev/hd4              is   77.9% full at       32 megabytes
/dev/hd2              is   94.9% full at      928 megabytes
/dev/hd9var           is   93.6% full at       56 megabytes
```

NFS BY OPERATION LOOP の例

実行済みの NFS 操作に対して処理をループするには、NFS BY OPERATION LOOP を使用します。この LOOP を使用すると、adviser スクリプトは特定の NFS 操作をチェックします。NFS_OP LOOP では、グローバル (gbl_)、テーブル (tbl_)、操作ごとのメトリックを使用できます。

以下の例では、実行したサーバーおよびクライアント操作を出力します。

```
PRINT "--------------------", gbl_stattime, "---------------------"
NFS_OP LOOP
  PRINT byop_server_count," server and ",byop_client_count,
    " client ",byop_name," operations performed"
```

システム上で NFS サーバーとしてのアクティビティを実行していないが、別の NFS サーバーでディレクトリの一覧表示を実行したユーザーがいる場合、出力結果は以下のようになります。

```
---------------------14:55:41-------------------------
    0 server and      0 client null           operations performed
    0 server and      2 client getattr        operations performed
    0 server and      0 client setattr        operations performed
    0 server and      0 client root           operations performed
    0 server and    886 client lookup         operations performed
    0 server and    884 client readlink       operations performed
    0 server and      0 client read           operations performed
    0 server and      0 client writecache     operations performed
    0 server and      0 client write          operations performed
    0 server and      0 client create         operations performed
    0 server and      0 client remove         operations performed
    0 server and      0 client rename         operations performed
    0 server and      0 client link           operations performed
    0 server and      0 client symlink        operations performed
  0 server and      0 client mkdir          operations performed
    0 server and      0 client rmdir          operations performed
```

```
    0 server and     28 client readdir        operations performed
    0 server and      1 client statfs         operations performed
```

NETWORK INTERFACE LOOP の例

構成済みの LAN デバイスに対して処理をループするときには、NETWORK INTERFACE LOOP 文を使用します。

```
# This version will only work with hp-ux 11.x.If you want it to(このバージョンは hp-ux 11.x でのみ動作します)。

# work for 10.20 you need to remove the "BYNETIF_QUEUE," string (10.20 で使用する場合は「BYNETIF_QUEUE」と後続の文字列を削除してください。)

# below as that metric is only available from 11.x glance. (このメトリックは、11.x glance でのみ使用可能です。)

# The following string variable should be changed to the interface(次の文字列変数を対象のインターフェイスに変更する必要があります。)

# of interest.  For example:(たとえば、)

#   netif_to_examine = "lan0"

# If you want to see all interfaces, leave it an empty string (""):(すべてのインスタンスを確認する場合は、以下のように空白文字("")のままにしてください。)

#   netif_to_examine = ""

# initialize variables: (変数の初期化を行います。)


headers_printed = headers_printed

netif loop {

# print information for the selected interface or if null then all:(選択したインターフェイス、null の場合はすべてのインターフェイスの情報を出力します。)

IF (BYNETIF_NAME == netif_to_examine) or

    (netif_to_examine == "") THEN

  {


# print headers the first time through the loop: (ループの初回にヘッダーを出力します。)

     IF headers_printed == 0 THEN

    {

      print "Time    Interface  InPkts OutPkts  OutQ  Colls  Errs"

      print "    "

      headers_printed = 1

    }

# print one line per interface reported:(報告されたインターフェイスごとに 1 行出力します。)
```

```
    print GBL_STATTIME, " ", BYNETIF_NAME|8,
      BYNETIF_IN_PACKET, BYNETIF_OUT_PACKET,
      BYNETIF_QUEUE, BYNETIF_COLLISION, BYNETIF_ERROR
    # (note that some interface types do not report collisions or (一部のインターフェイス型では衝突またはエラーを報告しませんので、)
    # errors)(注意してください。)
}
}
print "  "
```

出力結果は以下のようになります。

```
Time     Interface   InPkts OutPkts   OutQ   Colls   Errs

22:43:42 lan3          49       3      0       0       0
22:43:42 lan0           0       0      0       0       0
22:43:42 lan1           0       0      0       0       0
22:43:42 lan2           0       0      0       0       0
22:43:42 lo0            0       0      0       0       0

22:43:47 lan3         329       2      0       0       0
22:43:47 lan0           0       0      0       0       0
22:43:47 lan1           0       0      0       0       0
22:43:47 lan2           0       0      0       0       0
22:43:47 lo0            0       0      0       0       0
```

LOGICAL VOLUME の例

構成済みの論理ボリュームに対して処理をループするには、LOGICAL VOLUME ループを使用します。LOGICAL VOLUME LOOP では、グローバル (gbl_)、テーブル (tbl_)、論理ボリュームのメトリックを使用できます。

```
 PRINT "-----------------", gbl_stattime,    "-------------------"
  LV LOOP
    PRINT "Volume ", lv_dirname, " was read at a rate of ",
      lv_read_rate, " per second"
```

論理ボリュームを持つシステムにおける、2 つのインターバルの出力結果は以下のようになります。

```
----------------------------11:46:50-------------------------------
Volume /dev/vg00            was read at a rate of    0.0 per second
Volume /dev/vg00/group      was read at a rate of    0.0 per second
Volume /dev/vg00/lvol3      was read at a rate of  314.3 per second

----------------------------11:47:00-------------------------------
Volume /dev/vg00            was read at a rate of    0.0 per second
Volume /dev/vg00/group      was read at a rate of    0.0 per second
Volume /dev/vg00/lvol3      was read at a rate of   70.6 per second
```

PRM LOOP の例

*HP-UX でのみ使用できます。*Process Resource Manager (PRM) グループで検索された情報を繰り返すときには、PRM LOOP を使用します。PRM LOOP では、グローバル (gbl_)、テーブル (tbl_)、アプリケーション メトリックを使用できます。

以下の PRM LOOP の例では、利用率の高い待ち行列と CPU 割り当て使用率を超えた PRM グループのチェックを行います。

```
IF gbl_run_queue > 3 THEN {
  print " "


  print "--- High run queue = ", gbl_run_queue, " at ", gbl_stattime,
    " ---"
  prm loop {
      IF app_prm_state > 2 THEN
       IF app_cpu_total_util > app_prm_cpu_entitlement THEN
         print "    Note PRM group ", app_name_prm_groupname,
           " exceeds entitlement."
  }
}
```

各インターバルの出力は以下のようになります。

```
--- High run queue =   3.4 at 15:53:29 ---
    Note PRM group Testing exceeds entitlement.
```

PRM_BYVG LOOP の例

*HP-UX でのみ使用できます。*ボリューム グループの PRM グループに対して処理をループするには、PRM_BYVG ループを使用します。(PRM 情報は、PRM 構成ファイルで指定したボリューム グループでのみ使用可能です。)PRM_BYVG ループは LV ループにネストする必要があります。以下の例では、PRM グループによるディスク リソース利用率の統計値を表示します。

```
PRM loop {
```

```
  disk_state = app_prm_disk_state
}

IF disk_state == 0 THEN{
  print "  Disk manager state: Not Installed"
}

else IF disk_state == 1 THEN {
  print "  Disk manager state: Reset"
}

else IF disk_state == 2 THEN {
  print "  Disk manager state: Disabled"
}

else IF disk_state == 3 THEN {

  print "  Disk manager state: Enabled"

  lv loop {

    IF lv_type == "G" THEN {

      print "  Volume Group: ", lv_dirname
      print "                        %        %          KB"
      print "PRM Group     PRMID entitled achieved   transferred"
      print "-------------------------------------------------"

      prm_byvg loop {
        print prm_byvg_prm_groupname|13, prm_byvg_prm_groupid|5,
          prm_byvg_group_entitlement|8, prm_byvg_group_util|8,
          prm_byvg_transfer
      }
      print "  "
    }
  }
}
```

それぞれのインターバルの出力は以下のようになります。

```
Disk manager state: Enabled
Volume Group: /dev/vg00

                  %        %        KB

PRM Group     PRMID entitled achieved transferred
-------------------------------------------------
PRM_SYS           0       0     100          8
OTHERS            1      50       0          0
tools             2      50       0          0
```

PROCESS LOOP の例

アクティブなすべてのプロセスを繰り返すときには、PROCESS LOOP 文を使用します。PROCESS LOOP では、グローバル (gbl_)、テーブル (tbl_)、プロセスごと (proc_) のメトリックを使用できます。次の例では、PROCESS LOOP を使用して、あるインターバルにおいて最も CPU 使用率の高いプロセスを検索します。

```
big_proc_id = 0
big_proc_name = ""
big_proc_cpu = 0
PROCESS LOOP
IF proc_cpu_total_util > big_proc_cpu THEN {
  big_proc_cpu = proc_cpu_total_util
  big_proc_name = proc_proc_name
  big_proc_id = proc_proc_id
}


IF big_proc_cpu > 10 THEN
  YELLOW ALERT "Possible loop, process ", big_proc_name,
    " pid ", big_proc_id|6|0, " using ", big_proc_cpu, " % CPU"
```

SWAP LOOP の例

SWAP LOOP を使用すると、構成済みのスワップ領域に対して処理をループし、adviser スクリプトから [Swap Space] ウィンドウからの情報に関する報告を行えます。SWAP LOOP では、テーブル (tbl_)、グローバル (gbl_)、スワップごと (byswp_) のメトリックを使用できます。

以下の例では、2 つのスワップ デバイスを持つシステム上の利用可能なスワップ スペースをレポートします。

```
PRINT "--------------------", gbl_stattime, "----------------------"
SWAP LOOP
```

```
  PRINT BYSWP_SWAP_SPACE_NAME, " has ", BYSWP_SWAP_SPACE_USED,
    " used out of", BYSWP_SWAP_SPACE_AVAIL, " megabytes "
```

以下は 1 つのスワップ領域を持つシステムにおける、2 つのインターバルの出力結果です。

```
--------------------15:31:59-----------------------
/dev/hd6             has     37 used out of   128 megabytes

--------------------15:32:09-----------------------
/dev/hd6             has     37  used out of  128 megabytes
```

SYSTEM CALL LOOP の例

システム上のコールを繰り返すときには、SYSTEM CALL LOOP 文を使用します。SYSTEM CALL LOOP を使用すると、adviser スクリプトは [System Call] ウィンドウに表示されている情報をチェックします。SYSTEM CALL LOOP では、グローバル (gbl_)、テーブル (tbl_)、システム コール (syscall_) メトリックを使用できます。

以下の例では、高いシステム コール率をチェックし、最も頻繁に実行されるコールを出力します。

```
IF gbl_syscall_rate > 6000 THEN {
  print " "
  print "--- High syscall rate = ", gbl_syscall_rate, " at ",
    gbl_stattime, " ---"
  highestrate = 0

  systemcall loop {
    IF syscall_call_rate > highestrate THEN {
      highestrate = syscall_call_rate
      highestcall = syscall_call_name
    }
  }
  print "    Most frequent syscall was ", highestcall, " at",
    highestrate, " per second"
}
```

出力は以下のようになります。

```
--- High syscall rate =   6750.6 at 15:50:27 ---

    Most frequent syscall was gettimeofday at 6632.90 per second
```

TT LOOP の例

最後のインターバル中に記録されたトランザクション情報に対して処理をループするには、TT LOOP を使用します。この LOOP を使用すると、adviser スクリプトは [Transaction Tracking] ウィンドウに表示されている特定のトランザクション情報をチェックします。TT LOOP では、グローバル (gbl_)、テーブル (tbl_)、トランザクション追跡 (tt_) メトリックを使用できます。

以下の例では、システムに登録された各トランザクション名に対して完了したトランザクション数と平均応答時間を出力します。

```
PRINT "-------------------", gbl_stattime, "----------------------"

TT LOOP

  PRINT tt_name, " had ", tt_count, " transactions; ",

    "response time ", tt_wall_time_per_tran, " secs"
```

以下は 4 つのトランザクションを持つシステムにおける、2 つのインターバルの出力結果です。

```
-------------------------13:24:44--------------------------


First_Transaction  had   1 transactions; response time   1.000355 secs

Second_Transaction had   1 transactions; response time   2.000221 secs

Third_Transaction  had   1 transactions; response time   3.000231 secs

Fourth_Transaction had   0 transactions; response time   0.000000 secs


-------------------------13:24:54--------------------------


First_Transaction  had   3 transactions; response time   1.000383 secs

Second_Transaction had   1 transactions; response time   2.000216 secs

Third_Transaction  had   0 transactions; response time   0.000000 secs

Fourth_Transaction had   0 transactions; response time   0.000000 secs
```

TTBIN LOOP の例

システム上のアクティブな各トランザクションの応答時間 bin に対して処理をループするには、TTBIN LOOP を使用します。TTBIN LOOP は TT ループにネストする必要があります。この LOOP を使用すると、adviser スクリプトは [Transaction Graph] ウィンドウに表示されている特定のトランザクション情報をチェックします。TTBIN LOOP では、グローバル (gbl_)、テーブル (tbl_)、トランザクション追跡、トランザクション追跡 bin メトリックを使用できます。

以下の例では、インターバル中に完了したトランザクションを持つ各トランザクション名に対する応答時間 bin を出力します。

```
PRINT "--------------------", gbl_stattime, "----------------------"

TT LOOP

 IF (tt_count > 0) THEN

 {
```

```
  print "Transaction ", tt_name, " had ", tt_count, " transactions"
    lower_bin_limit = 0
    TTBIN LOOP
    {
      IF (ttbin_trans_count > 0) THEN {
        print "   ", ttbin_trans_count, " were between ",
          lower_bin_limit, " and ", ttbin_upper_range, " seconds"
        lower_bin_limit = ttbin_upper_range

            }
        }
  }
```

以下は 4 つのトランザクションを持つシステムにおける、2 つのインターバルの出力結果です。

```
-------------------------13:46:31--------------------------
Transaction First_Transaction    had      4 transactions
       2 were between     1.00 and   2.000000 seconds
Transaction Second_Transaction   had      1 transactions
       1 were between     2.00 and   3.000000 seconds
Transaction Third_Transaction    had      1 transactions
       1 were between     3.00 and   5.000000 seconds

-------------------------13:46:41--------------------------
Transaction First_Transaction    had      3 transactions
       1 were between     1.00 and   2.000000 seconds
Transaction Second_Transaction   had      1 transactions
       1 were between     2.00 and   3.000000 seconds
Transaction Fourth_Transaction   had      1 transactions
       1 were between     3.00 and   5.000000 seconds
```

TT LOOP ARM の例

ARM 2.0 では、TT_CLIENT ループ、TT_INSTANCE ループ、TT_UDM ループを TT LOOP 内にネストできます。The TT_CLIENT ループは相関関係のあるトランザクションを、TT_INSTANCE ループは最大 2048 のトランザクション インスタンスを、TT_UDM ループは特定のトランザクションに対するユーザー測定をリスト表示します。TT LOOP では、グローバル (gbl_)、テーブル (tbl_)、トランザクション追跡メトリックを使用できます。

TT_CLIENT ループ内に、TT_CLIENT_UDM ループをネストし、個々のコリレータのユーザー測定を表示できます。TT_INSTANCE_UDM ループまたは TT_INSTANCE_CLIENT ループを TT_INSTANCE ループにネストすれば、特定のインスタンスに固有のコリレータまたはユーザー測定を表示できます。

以下の例では、複数のループを使用して、特定のトランザクション インスタンスに対するユーザー測定を調べる方法を示しています。

例 1：SLO 違反の検索

```
# The following example loops through all transactions looking for (以下の例では、SLO 違反に対して)

# SLO violations, then prints the UDM information for all (処理をループし、すべてのインスタンスの UDM 情報を)

# instances:(出力します。)


print "----------------------", GBL_STATTIME, "--------------------"


tt loop {


  IF tt_slo_count > 0 THEN {
    print "  "
   print "SLO violation count:", tt_slo_count,
       " for transaction:", tt_name, " user:", tt_uname,
       " app:", tt_app_name, " threshold: ", tt_slo_threshold


    tt_instance loop {
      starttime = gbl_stattime - gbl_interval
      IF tt_instance_stop_time > starttime THEN {


        # found a completed instance in the transaction, print info: (トランザクションで完了したインスタンスを検索し、情報を出力)
        print "instance pid:", tt_instance_proc_id,
          " wall time:", tt_instance_wall_time


        tt_instance_udm loop {
          print "   ", tt_instance_user_measurement_name|44,
            " value= ", tt_instance_user_measurement_value
        }
      }
    }
```

```
  }
}
```

以下は 1 つのインターバルにおける出力です。

```
----------------------------17:19:03----------------------------
SLO violation count:    1 for transaction:Client_tra00       user:gracel
app:Client_Appl0         threshold:   5.000000
instance pid: 12137 wall time: 13.0407


SLO violation count:    1 for transaction:Server_transaction   user:joe
app:Server_Application   threshold:   5.000000
instance pid: 12137 wall time: 13.0358
   Metric #1 - Type 1 is a COUNTER32            value=           32
   Metric #2 - Type 4 is a GAUGE32              value=           37
   Metric #3 - Type 2 is a COUNTER64            value=     19088743
   Metric #4 - Type 9 is a STRING8              value=     String 8
   Metric #5 - Type 3 is a COUNTER32/DIVISOR32  value=        2.000
   Metric #6 - Type 8 is a NUMERICID64          value=     19088434
   The last field is always a STRING32          value=            0
instance pid: 12137 wall time:  3.0291
   Metric #1 - Type 1 is a COUNTER32            value=           32
  Metric #2 - Type 4 is a GAUGE32              value=           37
   Metric #3 - Type 2 is a COUNTER64            value=     19088743
   Metric #4 - Type 9 is a STRING8              value=     String 8
   Metric #5 - Type 3 is a COUNTER32/DIVISOR32  value=       21.333
   Metric #6 - Type 8 is a NUMERICID64          value=     19088434
   The last field is always a STRING32          value=            0
instance pid: 12137 wall time:  3.0256
   Metric #1 - Type 1 is a COUNTER32            value=           32
   Metric #2 - Type 4 is a GAUGE32              value=           37
   Metric #3 - Type 2 is a COUNTER64            value=     19088743
   Metric #4 - Type 9 is a STRING8              value=     String 8
   Metric #5 - Type 3 is a COUNTER32/DIVISOR32  value=       21.333
   Metric #6 - Type 8 is a NUMERICID64          value=     19088434
   The last field is always a STRING32          value=            0
instance pid: 12137 wall time:  2.0201
   Metric #1 - Type 1 is a COUNTER32            value=           32
   Metric #2 - Type 4 is a GAUGE32              value=           37
   Metric #3 - Type 2 is a COUNTER64            value=     19088743
```

```
   Metric #4 - Type 9 is a STRING8              value=     String 8
   Metric #5 - Type 3 is a COUNTER32/DIVISOR32  value=       21.333
   Metric #6 - Type 8 is a NUMERICID64          value=     19088434
  The last field is always a STRING32          value=            0
instance pid: 12137 wall time:  1.0101
   Metric #1 - Type 1 is a COUNTER32            value=           32
   Metric #2 - Type 4 is a GAUGE32              value=           37
   Metric #3 - Type 2 is a COUNTER64            value=     19088743
   Metric #4 - Type 9 is a STRING8              value=     String 8
   Metric #5 - Type 3 is a COUNTER32/DIVISOR32  value=       21.333
   Metric #6 - Type 8 is a NUMERICID64          value=     19088434
   The last field is always a STRING32          value=            0
```

例 2: ARM 2.0 構文

```
# The following example prints info for all completed transactions(以下の例では、インターバル中のすべての完了したトランザクションの)
# during the interval.(情報を出力します。)

print "-------------------", GBL_STATTIME,  "--------------------"

header_printed = 0
tt loop {

  tt_instance loop {

    starttime = GBL_STATTIME - GBL_INTERVAL

    IF TT_INSTANCE_STOP_TIME > starttime THEN {

      IF header_printed == 0 THEN {
        print "   "
        print "TranID  StartTime                 StopTime",
          "                   "
        header_printed = 1
      }

      print TT_TRAN_ID|6, "   ", TT_INSTANCE_START_TIME, "   ",
```

```
        TT_INSTANCE_STOP_TIME
      print "          TranName: ",TT_NAME|40
    }
  }
}
```

以下は 1 つのインターバルにおける出力です。

```
-----------------------------17:21:24-----------------------------
TranID  StartTime                  StopTime
     3  Wed Jun  3 17:21:07 1998  Wed Jun  3 17:21:20 1998
          TranName: Client_tra00

     7  Wed Jun  3 17:21:07 1998  Wed Jun  3 17:21:20 1998
          TranName: Server_transaction

     7  Wed Jun  3 17:21:17 1998  Wed Jun  3 17:21:20 1998
          TranName: Server_transaction

     7  Wed Jun  3 17:21:17 1998  Wed Jun  3 17:21:20 1998
          TranName: Server_transaction

     7  Wed Jun  3 17:21:18 1998  Wed Jun  3 17:21:20 1998
          TranName: Server_transaction

     7  Wed Jun  3 17:21:19 1998  Wed Jun  3 17:21:20 1998
           TranName:  Server_transaction
```

LDOM LOOP の例

HP-UX 11iv2 以降のみ。システム上のローカリティ ドメインのデータを繰り返すときには、LDOM LOOP 文を使用します。LDOM LOOP では、グローバル (gbl_)、テーブル (tbl_)、アイドル (ldom_) メトリックを使用できます。

この例では、システム上の各ローカリティ ドメインの LDOM メモリ使用量を出力します。

```
print " "
PRINT "----------------------", gbl_stattime, "----------------------"
print "LDOM Phys          Num              Mem      Mem    Mem"
print "  ID    ID Active CPUs Type      Avail     Free Used %"
PRINT "-----------------------------------------------------"
LDOM LOOP
{
```

```
 print LDOM_ID, " ", LDOM_PHYS_ID, " ", LDOM_ACTIVE, " ", LDOM_NUM_CPU, " ",
LDOM_MEM_TYPE, " ", LDOM_MEM_AVAIL, " ", LDOM_MEM_FREE, " ", LDOM_MEM_UTIL

}

PRINT "-----------------------------------------------------"
```

以下は 3 つの LDOM を持つシステムにおける、2 つのインターバルの出力結果です。

```
----------------------03:30:27-----------------------
LDOM Phys          Num              Mem      Mem    Mem
  ID   ID Active CPUs Type       Avail     Free Used %
-----------------------------------------------------
   0    0      1    4 LOCAL        0mb      0mb    0.0
   1    2      1    4 LOCAL        0mb      0mb    0.0
  na   na      1    0 GLOBAL     8.0gb    5.5gb   30.5
-----------------------------------------------------


----------------------03:30:32-----------------------
LDOM Phys          Num              Mem      Mem    Mem
  ID   ID Active CPUs Type       Avail     Free Used %
-----------------------------------------------------
   0    0      1    4 LOCAL        0mb      0mb    0.0
   1    2      1    4 LOCAL        0mb      0mb    0.0
  na   na      1    0 GLOBAL     8.0gb    5.5gb   30.5
-----------------------------------------------------
```

PROC_LDOM LOOP の例

*HP-UX 11iv2 以降のみ。*プロセスがメモリを取得できるローカリティ ドメインに対してループするには、PROC_LDOM を使用します。

PROC_LDOM ループは PROCESS ループにネストする必要があります。

以下の例では、異なる LDOM からメモリを取得できる scopeux プロセスを表示します。

```
print " "

PRINT "-----------------------------------", gbl_stattime,
"-----------------------------------"

print "Process           LDOM LDOM         RSS       RSS       RSS       RSS
LDOM"

print "Name                ID Type       Total     Shared    Private   Weighted
Mem %"

PRINT
"-------------------------------------------------------------------------------
-----"
```

```
PROCESS LOOP
{
if PROC_PROC_NAME == "scopeux" then
{
PROC_LDOM LOOP
{
print PROC_PROC_NAME,"  ",PROC_LDOM_ID," ", PROC_LDOM_TYPE," ",
PROC_LDOM_TOTAL," ", PROC_LDOM_SHARED," ", PROC_LDOM_PRIVATE," ",
PROC_LDOM_WEIGHTED," ", PROC_LDOM_PCT
}
}
}
PRINT
"--------------------------------------------------------------------------
-----"
```

scopeux プロセスに対してフィルタリングされた 3 つの LDOM を持つシステムにおける、2 つのインターバルの出力結果は以下のようになります。

```
-----------------------------------04:53:50---------------------------------
----
Process            LDOM LDOM          RSS        RSS        RSS        RSS   LDOM
Name                 ID Type        Total     Shared    Private   Weighted  Mem %
-------------------------------------------------------------------------------
----
scopeux               0 LOCAL         0kb        0kb        0kb        0kb    0.0
scopeux               1 LOCAL         0kb        0kb        0kb        0kb    0.0
scopeux              na GLOBAL     31.9mb     23.7mb      8.2mb     13.4mb    0.2
-------------------------------------------------------------------------------
----

-----------------------------------04:53:55---------------------------------
----
Process            LDOM LDOM          RSS        RSS        RSS        RSS   LDOM
Name                 ID Type        Total     Shared    Private   Weighted  Mem %
-------------------------------------------------------------------------------
----
scopeux               0 LOCAL         0kb        0kb        0kb        0kb    0.0
scopeux               1 LOCAL         0kb        0kb        0kb        0kb    0.0
scopeux              na GLOBAL     31.9mb     23.7mb      8.2mb     13.4mb    0.2
-------------------------------------------------------------------------------
----
```

PRINT 文

PRINT 文を使用すると、収集しているデータを標準出力 (padv コマンド コンソール) に出力できます。メトリックまたは計算済みの変数を記録するのに、PRINT 文を使用できます。

構文 :

```
PRINT printlist

PRINT Example

PRINT "The Application OTHER has a total CPU of ",


  other:app_cpu_total_util, "%"
```

この文が呼び出されると、以下のようなメッセージが padv コマンド コンソールに書き込まれます。

```
The Application OTHER has a total CPU of 89%
```

SYMPTOM 文

構文 :

```
SYMPTOM variable [TYPE = {CPU, DISK, MEMORY, NETWORK}]

RULE measurement {>, <, >=, <=, ==, !=} value PROB probability

[RULE measurement {>, <, >=, <=, ==, !=} value PROB probability]

   .

   .

   .
```

SYMPTOM キーワードおよび RULE キーワードは、SYMPTOM 文でのみ使用し、別の構文では使用できません。SYMPTOM 文は、トップレベルの文でなければならず、他の文内にネストできません。

variable は、この兆候の名前になる変数名です。SYMPTOM 文で定義された変数名は別の構文で使用できますが、それらの構文で変数値を変更しないようにしてください。

RULE は SYMPTOM 文のオプションであり、独立して使用できません。RULE オプションは SYMPTOM 文内で必要な数だけ使用できます。

SYMPTOM 変数は、各インターバルで RULE に従って評価されます。

- measurement は、RULE の一部として評価される変数またはメトリックの名前です。
- value は、measurement と比較される定数、変数、メトリックです。
- probability は数値定数、変数、メトリックです。

真であるすべての SYMPTOM RULE の確率を加算して、SYMPTOM 値を作成します。SYMPTOM 値は padv コマンド コンソールのメッセージ内に表示されます。

測定値と値との間の条件が真であるすべての確率の合計が、その兆候が起きている確率になります。

SYMPTOM の例

構文 :

```
SYMPTOM CPU_Bottleneck TYPE=CPU
```

```
RULE gbl_cpu_total_util > 50  PROB 25
RULE gbl_cpu_total_util > 85  PROB 25
RULE gbl_cpu_total_util > 90  PROB 25
RULE gbl_run_queue      >  3  PROB 50
SYMPTOM CPU_Level TYPE=CPU
RULE gbl_cpu_sys_mode_util > 40  PROB 25
RULE gbl_cpu_sys_mode_util > 50  PROB 25
RULE gbl_cpu_sys_mode_util > 60  PROB 25
RULE gbl_cpu_sys_mode_util > 70  PROB 50
```

上記で定義された CPU の兆候のうちどの兆候が最も高い総確率 (PROB) を持つかに関わらず、その兆候が padv コマンド コンソールのメッセージの重要度のレベルを決定します。

SYMPTOM の例 : グローバル CPU ボトルネック

```
SYMPTOM Symp_Global_Cpu_Bottleneck TYPE=CPU
RULE gbl_cpu_total_util > 50  PROB 25
RULE gbl_cpu_total_util > 85  PROB 25
RULE gbl_cpu_total_util > 90  PROB 25
RULE gbl_run_queue      > 3   PROB 75
```

たとえば、インターバルの CPU 使用率 (gbl_cpu_total_util) が 93% で、実行待ち行列が 2 であった場合、最初の 3 つの規則がすべて真となり、確率に 25 が 3 回追加されます。4 つめの規則は真でないため、75 は追加されません。したがって、このインターバルの Symp_Global_Cpu_Bottleneck 変数の値は 75 (%) となります。

11 Windows での Performance Collection Component の使用

Performance Collection Component のグラフィカル ユーザー インターフェイスにアクセスするには、以下のフォルダの [HP Operations Agent Software] アイコンをクリックします。

[スタート] → [すべてのプログラム] → [HP] → [Operations Agent] → [Performance Collection Component]

図 2　　Performance Collection Component メイン ウィンドウ

この章では、Performance Collection Component グラフィカル インターフェイスを使用して実行する次の作業について説明します。

- データ型とクラス
- 要約レベル
- 抽出またはエクスポートするデータの範囲
- ログ ファイル データの抽出 および ログ ファイル データのエクスポート
- ログ ファイル データのアーカイブ
- ログ ファイルのサイズ変更
- ログ ファイルの走査
- ログ ファイルの分析
- エクスポート テンプレートの構成
- ユーザー オプションの構成
- 収集パラメータの構成
- アラーム定義の構成
- Performance Collection Component のステータス確認
- パフォーマンス カウンタ収集の構築

データの抽出、エクスポート、アーカイブが伴う作業で Performance Collection Component を使用し始める前に、以下の各項をお読みください。これらの項では、データ型と要約レベルの選択方法や、抽出、エクスポート、アーカイブの対象データの範囲の選択方法について説明します。

データ型とクラス

抽出またはエクスポートの対象として、次のデータ型の scopent ログ ファイル データを選択できます。

データ型	測定タイプ
global	システム全体の情報、つまりグローバル情報
application	ユーザー定義の各アプリケーションに含まれるプロセス
configuration	システム構成の使用率
process	選択した対象プロセス
disk	ディスク デバイスの使用量
filesystem	論理デバイスの使用量
logicalsystem	論理システムの使用量
cpu	CPU の使用量
netif	ネットワーク インターフェイス デバイスの使用量
transaction	トランザクション追跡データ

エクスポートする DSI ログ ファイル データは、クラスに従って選択できます。各クラスは、1 つのソースの受信データを表し、一緒に記録される関連データ項目 (メトリック) のグループから構成されます。

要約レベル

データを抽出またはエクスポートするには (ログ ファイル データを抽出またはエクスポートするときに)、希望の要約レベル (詳細、要約、またはその両方) を指定する必要があります。

- 詳細を指定すると、5 分間隔の詳細データが、プロセス データを除くすべてのデータ型からエクスポートされます。プロセス データからは、1 分間隔の詳細データがエクスポートされます。
- 要約を指定すると、1 時間分の要約データがエクスポートされます。
- 詳細と要約ではエクスポート データの量が最大になります。

要約は、エクスポート データのサイズに影響します。たとえば、1 時間ごとの要約データは、5 分間の詳細データのサイズの約 10 分の 1 になります。

抽出またはエクスポートするデータの範囲

抽出またはエクスポートするデータは、データの記録日時に基づいて選択できます。たとえば、特定の日付から 30 日間にわたって毎日 (月曜日から日曜日まで) 8:00 am から 8:00 pm の間に記録されたデータを選択することが可能です。

エクスポートするデータの範囲を指定しないと、データは、デフォルトの開始日 (ログ ファイル内の最終日から 30 日前の日付) を使用して抽出またはエクスポートされます。存在するデータの量が 30 日分に満たない場合、ログ ファイル内で最も古いレコードの日付が開始日になります。

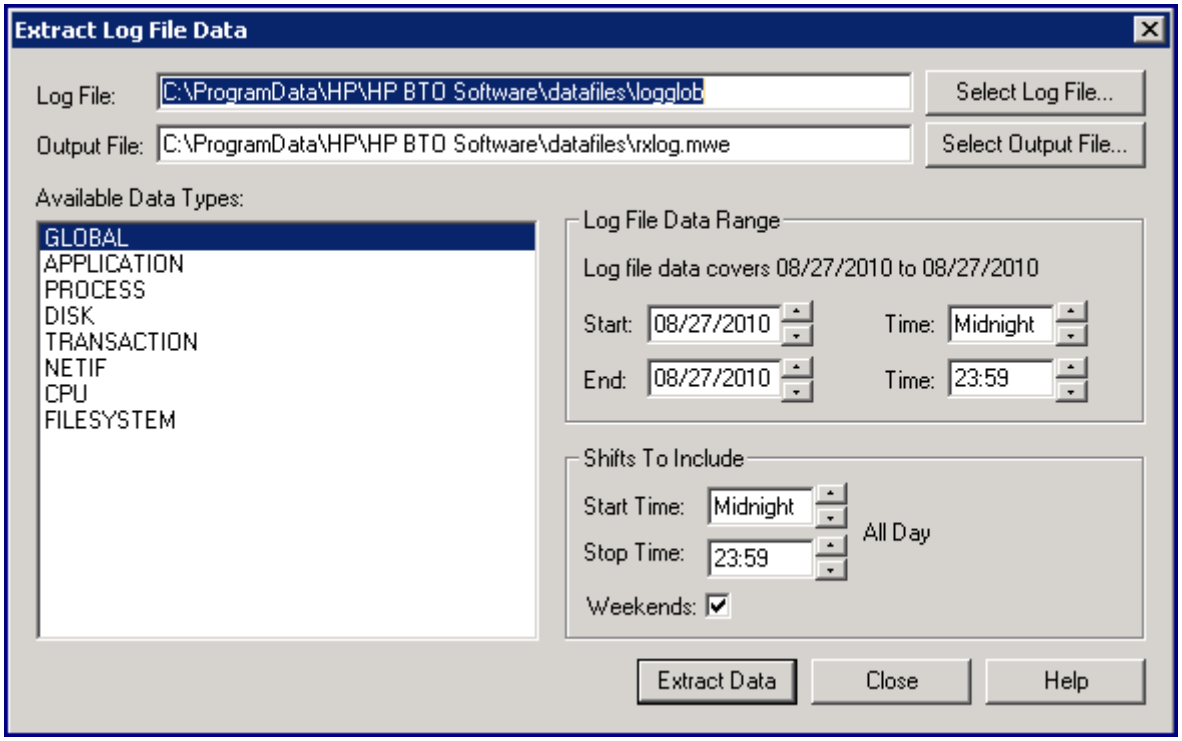
月曜日から金曜日までの作業シフトに対応する時間帯 (8:00 am から 5:00 pm までなど) に記録されたデータのみを、抽出またはエクスポートすることもできます。シフトを指定しないと、デフォルトとして、週末を含めた毎日の 24 時間分のデータが抽出またはエクスポートされます。

ログ ファイル データの抽出

データ コレクタである scopent は、データを継続的に収集し、それを生ログ ファイルに記録します。ユーザーは、デフォルトのグローバル ログ ファイル セットである logglob から抽出ログ ファイルに特定のデータを抽出できます。抽出ログ ファイルは、後で HP Performance Manager などの分析プログラムによる分析やアーカイブに使用できます。また、既存の抽出ログ ファイルからもデータを抽出できます。DSI ログ ファイル データは抽出できません。

logglob を指定すると、ログ ファイル セット内に含まれる他のすべての生ログ ファイルが自動的にオープンします。たとえば、プロセス データを抽出するために logproc ログ ファイルをオープンする必要はありません。logglob をオープンすると、生ログ ファイル セット内のすべてのデータ型にアクセスできます。

図 3　　[Extract Log File Data] ダイアログ ボックス



Performance Collection Component を開始した後に [Extract Log File Data] または [Export Log File Data] ダイアログ ボックスを表示すると、現在アクティブなログ ファイルを示すためにデフォルトのグローバル ログ ファイル名 logglob が [Log File] ボックスに表示されます。Logglob は、Performance Collection Component を終了するか、別のログ ファイルを選択するまでアクティブです。別のログ ファイルを選択すると、そのファイルの名前が現在アクティブなログ ファイルとして [Log File] ボックスに表示されます。

ログ ファイル データの抽出手順

ログ ファイル データを抽出するには、次の手順を実行します。

1 メイン ウィンドウの [**Logfile**] メニューから [**Extract**] をクリックします。[Extract Log File Data] ダイアログ ボックスが表示され、現在アクティブなログ ファイルと現在選択している出力ファイルの名前が示されます。別のログ ファイルと出力ファイルも簡単に指定できます。

2 抽出するデータ型を選択してから、抽出するログ ファイル データの範囲と対象シフトを選択します。[**Extract Data**] ボタンをクリックすると、抽出プロセスが開始します。

ログ ファイル データの抽出手順を参照するには、[Help] メニューから [**Help Topics**] を選択し、[**How Do I...?**] を選んでから、[**Extract log file data**] を選んでください。

ログ ファイル データのエクスポート

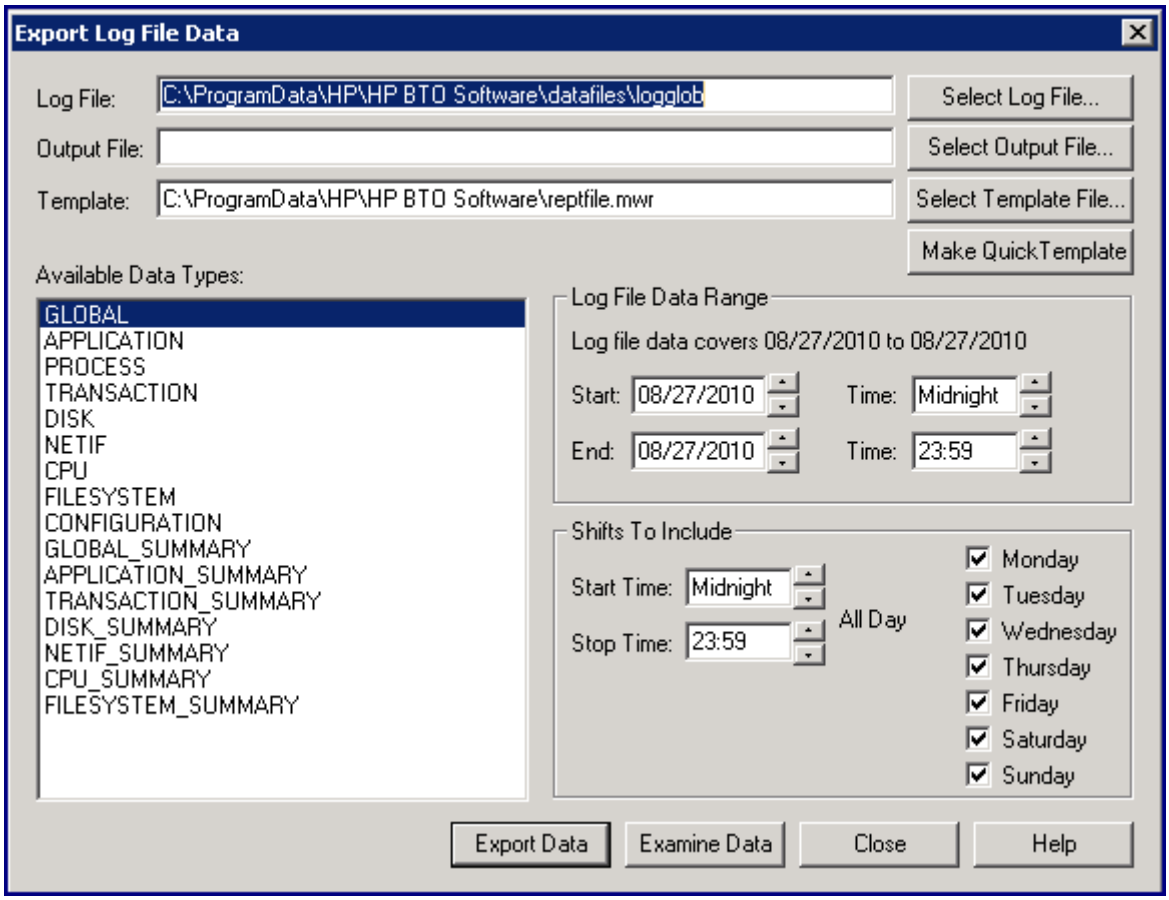
特定のデータを scopent の生ログ ファイルまたは抽出ログ ファイル、または DSI ログ ファイルから書式付きエクスポート ファイルに抽出できます。書式付きエクスポート ファイルは、表計算ソフトやその他のレポート作成ツールで使用できます。

エクスポート関数はデータをログ ファイルから削除しません。

以下の項では、次の事柄について説明します。

- エクスポート データに指定できるファイル属性
- エクスポート タスクを簡略化するためのエクスポート テンプレート
- デフォルトのエクスポート ファイル
- エクスポート タスクの概要

図 4 [Export Log File Data] ダイアログ ボックス



ファイル属性

エクスポート データには、ファイルのフォーマット、欠落データを表す値、フィールドの区切り記号、カラムのヘッダー、分単位の要約インターバル、レイアウト、データ型、エクスポート ファイルに含めるメトリックなど、各種のファイル属性を割り当てられます。これらの属性は、

エクスポート テンプレート ファイルに保存できます。また、[Make Quick Template] ダイアログ ボックスを使用して直接指定できます (詳細については、221 ページの「クイック エクスポート テンプレートの作成」を参照してください)。

ファイル フォーマット

エクスポート ファイルの出力フォーマットには、ASCII、datafile、binary、WK1 (表計算シート) のいずれかを指定できます。

- ASCII フォーマットは印刷可能な文字データで、印刷されるレポートや、ユーザーが記述したプログラムまたはバッチ ファイルによるポスト処理に適したものです。
- datafile フォーマットは、数値以外の項目が二重引用符 (" ") で囲まれる点を除いて ASCII と同じです。datafile フォーマットは、ほとんどの表計算ソフトおよびグラフ作成ソフトへのデータ転送に使用できます。
- binary フォーマットは印刷できません。数値が 2 進整数値として表されるためデータがコンパクトになります。変換が最小限で済み、メトリック精度が最高であるため、このフォーマットは、ユーザーが記述した分析プログラムへの入力に最も適しています。
- WK1 (表計算シート) フォーマットは、Microsoft Excel などの表計算ソフト、データベース、グラフ作成ソフトに対応します。

欠落値

エクスポート ファイルには、ソース ログ ファイルから欠落したデータを補うデータ値を入れることができます。データの欠落は、特定のバージョンの scopent コレクタで使用できないメトリックがある場合に発生します。さらに、アプリケーション、ディスク、トランザクションに関する複数レイアウト エクスポート フォーマットは、すべてのアプリケーション、ディスク、トランザクションに関する出力レコードに領域を確保します。特定のエントリに関してデータが一定期間記録されない場合、そのデータは「欠落データ」になります。

フィールド区切り記号

ASCII フォーマットおよび datafile フォーマットのエクスポート ファイルでは、メトリック間にフィールド区切り記号を挿入できます。区切り記号として使用する文字には、印刷可能なものと印刷不可能なもの (タブ文字など) があります。

デフォルトの区切り記号はスペースですが、多くのプログラムでは、フィールド区切り記号としてコンマを使用する必要があります。

分単位の要約インターバル

各要約インターバルを分単位で指定します。各要約インターバルは、分単位で 5 ～ 1440 分 (1 日) の範囲で指定できます。

ヘッダー

エクスポート ファイルにはカラムのヘッダーを含められます。ファイル内の最初のレコードはエクスポート データです (WK1 フォーマットのファイルを除く)。ただし、ファイルにヘッダーを含める場合、ASCII フォーマットおよび datafile フォーマットのファイルでは、エクスポート

ファイルのタイトル (指定されている場合) とメトリックの各カラムのヘッダーがファイルの最初のレコードの前に書き込まれます。バイナリ フォーマット ファイルでは、ヘッダーがファイルの最初のレコードの前に書き込まれ、メトリックの説明が含まれます。

WK1 ファイルでは、必ずヘッダーが書き込まれます。

複数レイアウト

アプリケーションやディスクなどのマルチインスタンスのデータ型には、(1 つのレコード出力に) 複数のレイアウトを指定できます。

単一のレイアウトは、特定の時間インターバルでアクティブであった各アプリケーションまたは各ディスクについて 1 レコードを書き込みます。複数のレイアウトは、各時間インターバルで 1 レコードを書き込みます。このレコードの一部は、構成されている各アプリケーションまたは各ディスクについて保存されます。

エクスポート ファイルのタイトル

エクスポート ファイルにはタイトルを指定できます。タイトルには、リテラル文字列と置換キーワードを使用できます。`export title` 文字列には、次の置換キーワードが使用できます。

`!date`	エクスポート ファイルの作成日
`!time`	エクスポート ファイルの作成時刻
`!logfile`	データ取得元のログ ファイルの名前
`!collector`	コレクタ プログラム (`scopent` または `dsilog`) の名前とバージョン
`!class`	必要なデータの型
`!system_id`	`scopent` の生ログ ファイル データまたは抽出ログ ファイル データを収集するシステムの識別子 (DSI ログ ファイル データに対しては無効)

たとえば、次のような文字列を入力します。

```
export "!system_id data from !logfile on !date !time"
```

この文字列により、次のようなタイトルが生成されます。

```
gemini data from logglob on 10/25/99 08:30 AM
```

エクスポート ファイルのテンプレート

エクスポート タスクは、エクスポート ファイルのファイル属性を定義するエクスポート テンプレートを使用します。デフォルトのファイル属性は、次の指定が含まれるファイル `<rpmtools>\data\reptfile.mwr` から得られます。

- ASCII ファイル フォーマット
- 欠落値に対する 0 (ゼロ)
- フィールド区切り記号の空白
- 60 分ごとの要約
- ヘッダーを含める

- 特定のデータ型に推奨されるメトリック セットをエクスポートに含める

別のエクスポート テンプレート ファイルを指定しても、特別なファイル属性を指定してもかまいません (221 ページの「クイック エクスポート テンプレートの作成」を参照してください)。

カスタマイズしたエクスポート テンプレートも [Configure Export Templates] ダイアログ ボックスを使用して作成できます (222 ページの「エクスポート テンプレートの構成」を参照してください)。

デフォルトのエクスポート ファイル

エクスポート データを収める出力ファイルを指定しないと、エクスポート タスクでは、指定したデータ型および要約レベルに基づいて *<ディスク ドライブ>*:\Program Files\HP\HP BTO Software\data\datafiles ディレクトリにデフォルトの出力ファイルを作成します。

scopent データ

scopent ログ ファイル データをエクスポートする場合、次のデフォルト ファイル名がエクスポート ファイルに割り当てられます。

xfrdGLOBAL.ext	グローバル詳細データ
xfrsGLOBAL.ext	グローバル要約データ
xfrdAPPLICATION.ext	アプリケーション詳細データ
xfrsAPPLICATION.ext	アプリケーション要約データ
xfrdPROCESS.ext	プロセス詳細データ
xfrdDISK.ext	ディスク デバイス詳細データ
xfrsDISK.ext	ディスク デバイス要約データ
xfrdCPU.ext	CPU 詳細データ
xfrsCPU.ext	CPU 要約データ
xfrdFILESYSTEM.ext	ファイルシステム詳細データ
xfrsFILESYSTEM.ext	ファイルシステム要約データ
xfrdNETIF.ext	netif 詳細データ
xfrsNETIF.ext	netif 要約データ
xfrdTRANSACTION.ext	トランザクション追跡詳細データ
xfrsTRANSACTION.ext	トランザクション追跡要約データ
xfrdCONFIGURATION.ext	構成詳細データ
xfrdLOGICAL.ext	論理システム詳細データ ファイル
xfrsLOGICAL.ext	論理システム要約データ ファイル

エクスポート タスクが完了すると、[Export Log File Data] ダイアログ ボックスの [**Examine Data**] ボタンをクリックして、出力エクスポート ファイルの内容を表示できます。

デフォルトのファイル名は、データ型名から作成されます。データが詳細データであるか要約データであるかによって、接頭辞が xfrd または xfrs になります。拡張子 (.ext) は、エクスポート テンプレート ファイルで指定されているファイル フォーマットに応じて、asc (ASCII)、bin (バイナリ)、dat (データファイル)、wk1 (表計算シート) のいずれかになります。

次に例を示します。

xfrdNETIF.wk1 には、NETIF データ型に関する詳細データが表計算シート フォーマットで書き込まれています。

xfrsAPPLICATION.asc には、アプリケーション データ型に関する要約データが ASCII フォーマットで書き込まれています。

DSI データ

DSI ログ ファイル データを抽出する場合、デフォルトのファイル名はクラス名から作成されます。データが詳細データであるか要約データであるかによって、接頭辞が xfrd または xfrs になります。拡張子は、エクスポート テンプレート ファイルで指定されているファイル フォーマットに応じて、asc (ASCII)、dat (データファイル)、wk1 (表計算シート) のいずれかです。

次に例を示します。

xfrdACCTG.wk1 には、ACCTG クラスに関する詳細データが表計算シート フォーマットで書き込まれています。

xfrsPERSONL.asc には、PERSONL クラスに関する要約データが ASCII フォーマットで書き込まれています。

ログ ファイル データのエクスポート手順

ログ ファイル データをエクスポートするには、次の手順を実行します。

1 メイン ウィンドウの [Logfile] メニューから [**Export**] をクリックします。[Export Log File Data] ダイアログ ボックスが表示され、現在アクティブなログ ファイルと現在選択しているエクスポート テンプレート ファイルの名前が示されます。別のログ ファイルおよびエクスポート テンプレート ファイルも指定できます。

出力ファイルを指定すると、選択したすべてのデータ型またはクラスのデータがそのファイルに収められます。出力ファイルを指定しない場合、指定したデータ型またはクラス、および要約レベルに基づいてデフォルトの出力ファイルが作成されます (219 ページの「デフォルトのエクスポート ファイル」を参照してください)。

2 エクスポートするデータ型またはクラスを 1 つまたは複数選択してから、エクスポートするログ ファイル データの範囲と、対象シフトを選択します。

3 デフォルトのエクスポート テンプレートで指定されているファイル属性を無効にする場合や、エクスポート ファイルに含めるメトリックを選択する場合、[**Make Quick Template**] ボタンをクリックしてください。

4 クイック テンプレートの作成が終了し、エクスポート ファイルに含めるメトリックを選択した後、[Export Log File Data] ダイアログ ボックスに戻ってください。[**Export Data**] ボタンをクリックして、エクスポート プロセスを開始します。

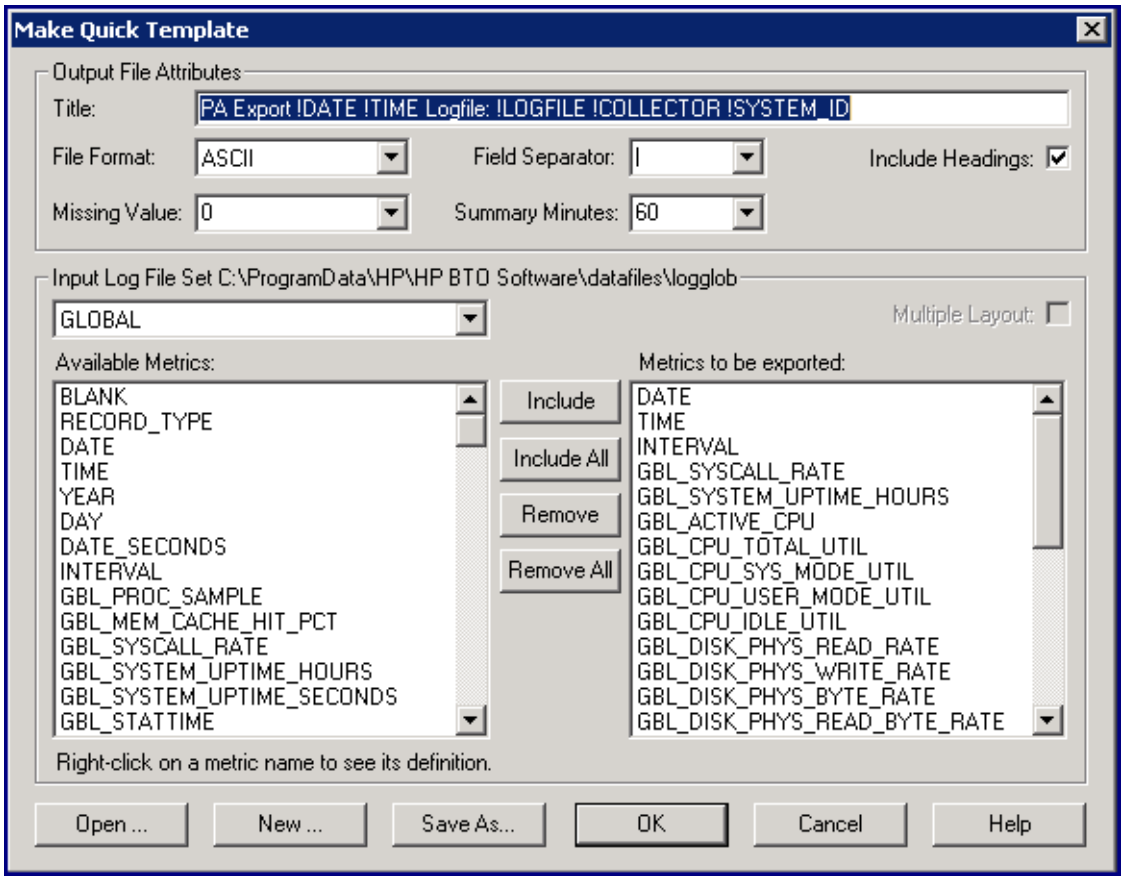
5 エクスポートの終了後、[**Examine Data**] ボタンをクリックしてエクスポート ファイルの内容を参照できます。

ログ ファイル データのエクスポート手順を参照するには、[Help] メニューから [**Help Topics**] を選択し、[**How Do I...?**] を選んでから、[**Export log file data**] を選んでください。

クイック エクスポート テンプレートの作成

エクスポート ファイルに含めるメトリックを変更するときや、エクスポート用に選択したエクスポート テンプレートに指定されているファイル属性やメトリックを変更するときは、[Export Log File Data] ダイアログ ボックスの [Make Quick Template] 機能を使用します。

図 5　　[Make Quick Template] ダイアログ ボックス



クイック テンプレートを作成するには、次の手順を実行します。

1. [Export Log File Data] ダイアログ ボックスの [**Make Quick Template**] ボタンをクリックします。[Make Quick Template] ダイアログ ボックスが表示され、エクスポート用に選択しているエクスポート ファイルのタイトルが示されます。
2. [Output File Attributes] の下にある各ボックスには、エクスポート用に選択しているエクスポート テンプレートのファイル属性の設定に基づき、エクスポート ファイルに関する現在の設定値が示されます。これらの設定値は変更できます。
3. 別のエクスポート テンプレートファイルを使用するには、[**Open**] ボタンをクリックします。
4. [Make Quick Template] ダイアログ ボックスから既存の設定値をすべてクリアし、[**New**] ボタンをクリックしてまったく新しいエクスポート テンプレート ファイルを作成することもできます。

エクスポートするメトリックの選択

- エクスポートするメトリックのデータ型またはクラスを選択した後、エクスポートに含めるメトリックを選択できます。それぞれのデータ型またはクラスには独自のメトリック セットがあり、[Available Metrics] にリストされます。[Metrics to be exported] にリストされるメトリックは、エクスポートに含める現在のエクスポート テンプレートで指定されているメトリックです。このリストはそのまま使用できますが、リストからメトリックを削除したり、他のメトリックを選択してエクスポートに含めることもできます。
- アプリケーション、ディスク、cpu、ファイルシステム、netif またはトランザクションのいずれかのデータ型を選択した場合、[**Multiple Layout**] チェック ボックスを選択すると複数レイアウト (1 レコード出力につき) を生成できます。また、[**Multiple Layout**] チェック ボックスを選択しないと単一レイアウトを生成できます。

選択内容の保存

選択後に、次のいずれかの処理が行えます。

- テンプレート ファイルに対する変更内容を保存せずに、エクスポートするファイル属性およびメトリックのリストに対して行った選択を使用して、エクスポートを続行できます。
- 将来のエクスポートで使用できるように、新しいエクスポート テンプレート ファイルに選択内容を保存できます。元のエクスポート テンプレート ファイルは変更せずにそのままにします。

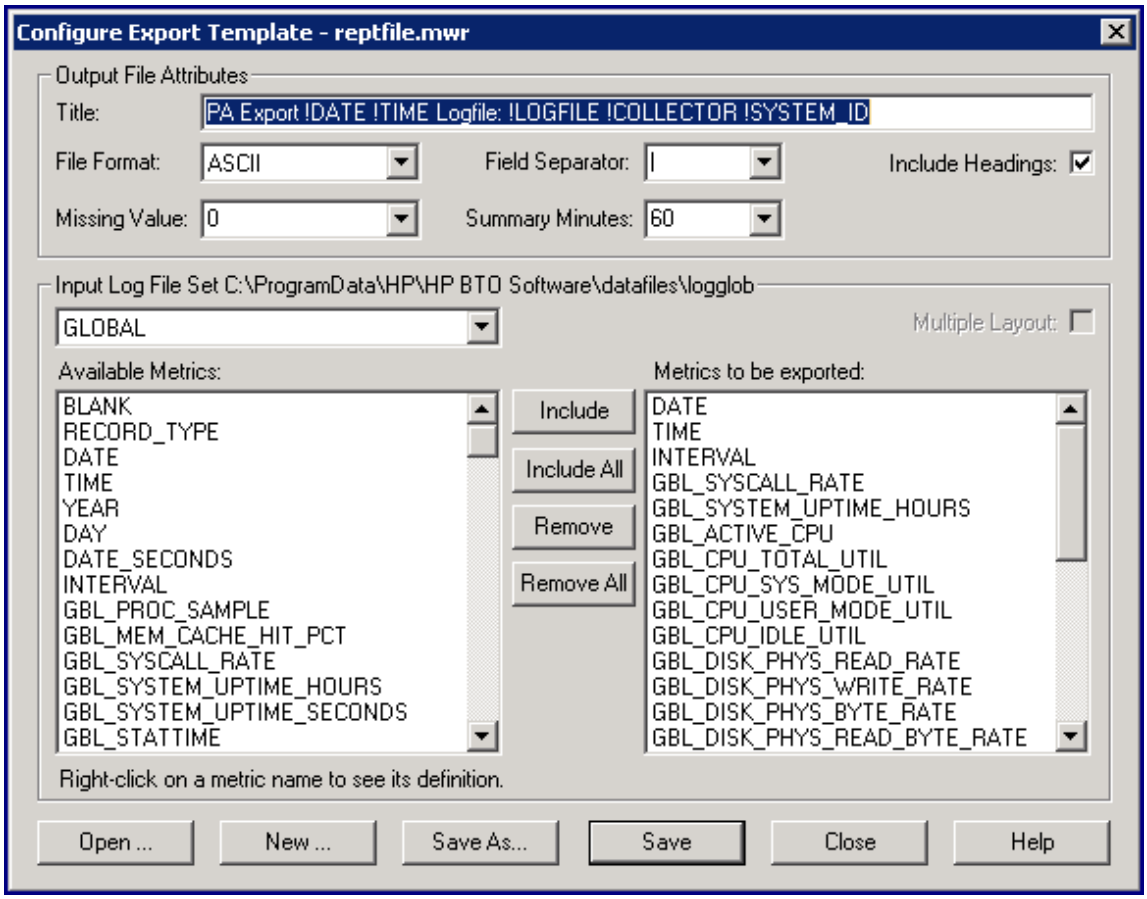
選択内容を新しいエクスポート テンプレート ファイルに保存する場合、新しいテンプレート ファイル名を指定するときにファイル名拡張子 .mwr を含める必要があります。たとえば、mytmplte.mwr のようなファイル名です。

クイック エクスポートのテンプレートの作成手順を参照するには、[Help] メニューから [**Help Topics**] を選択し、[**How Do I...?**] を選んでから、[**Make a quick export template**] を選んでください。

エクスポート テンプレートの構成

既存のエクスポート テンプレート ファイルをカスタマイズするときや、エクスポート テンプレート ファイルを新規作成するときには、[Configure] メニューの [**Export Templates**] コマンドを使用します。テンプレートに含める新しいファイル属性やメトリックを選択するときには、[Configure Export Template] ダイアログ ボックスを使用します。

図 6　　[Configure Export Template] ダイアログ ボックス



エクスポート テンプレートを設定するには、次の手順を実行します。

1. メイン ウィンドウの [Configure] メニューから [**Export Templates**] をクリックします。[Configure Export Template] ダイアログ ボックスが表示され、現在オープンしているエクスポート テンプレート ファイルの名前がダイアログ ボックス タイトルに示されます。別のエクスポート テンプレートファイルを編集するには、[**Open**] ボタンをクリックします。
2. [Output File Attributes] の下にある各ボックスには、構成しているエクスポート テンプレートのファイル属性の設定に基づき、エクスポート ファイルに関する現在の設定値が示されます。これらの設定値は変更できます。
3. [Configure Export Template] ダイアログ ボックスから既存の設定値をすべてクリアし、[**New**] ボタンをクリックしてまったく新しいエクスポート テンプレート ファイルを作成することもできます。

エクスポートするメトリックの選択

- エクスポートするメトリックのデータ型またはクラスを選択した後、エクスポートに含めるメトリックを選択できます。それぞれのデータ型またはクラスには独自のメトリック セットがあり、[Available Metrics] にリストされます。[Metrics to be exported] にリストされるメトリックは、エクスポートに含める現在のエクスポート テンプレートで指定されているメトリックです。このリストはそのまま使用できますが、リストからメトリックを削除したり、他のメトリックを選択してエクスポートに含めることもできます。

- アプリケーション、ディスク、cpu、ファイルシステム、netif またはトランザクションのいずれかのデータ型を選択した場合、[**Multiple Layout**] チェック ボックスを選択すると複数レイアウト (1 レコード出力につき) を生成できます。また、[**Multiple Layout**] チェック ボックスを選択しないと単一レイアウトを生成できます。

選択内容の保存

編集したテンプレートを保存するときには、次の 3 つの選択肢があります。

- 現在のテンプレート ファイルに変更内容を保存します。
- 新しいテンプレート ファイルに変更内容を保存します。この場合、元のテンプレート ファイルは変更されません。
- 変更内容をキャンセルして、テンプレート ファイルの変更を防ぎます。

選択内容を新しいエクスポート テンプレート ファイルに保存する場合、新しいファイル名を指定するときにファイル名拡張子 .mwr を含める必要があります。

テンプレート ファイルに変更を加え、変更内容を保存する前に [**Close**] ボタンをクリックすると、変更内容のキャンセルまたは保存を促すプロンプトが表示されます。[**Cancel**] ボタンをクリックすると、[Configure Export Template] ダイアログ ボックスに戻ります。

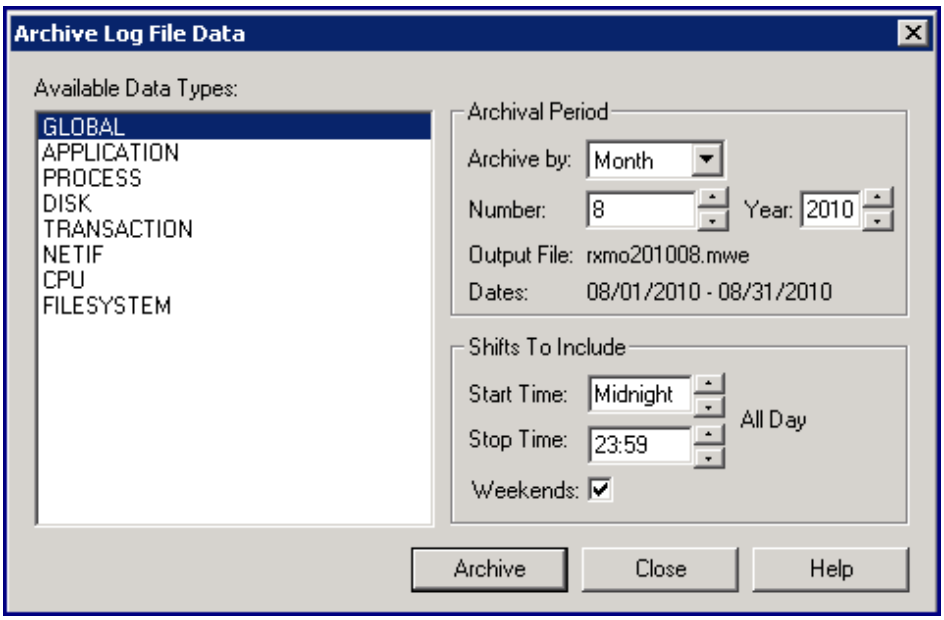
エクスポート テンプレート ファイルの構成手順を参照するには、[Help] メニューから [**Help Topics**] を選択し、[**How Do I...?**] を選んでから、[**Configure an export template file**] を選んでください。

ログ ファイル データのアーカイブ

アーカイブや将来のデータ分析に使用する scopent ログ ファイル データの選択箇所を抽出するときには、[Logfile] メニューの **[Archive]** コマンドを使用します。

アーカイブが目的の場合、データは生ログ ファイルから抽出できます。抽出データは、*< ディスク ドライブ >*:Program Files\HP\HP BTO Software\data\datafiles ディレクトリにあるアーカイブ出力ファイルに自動的に配置されます。このディレクトリ名は、選択しているアーカイブ期間を反映します。このようなファイルは、テープにコピーしてオフラインで保存した後、ディスク領域を解放するために削除できます。

図 7　　[Archive Log File Data] ダイアログ ボックス



アーカイブ期間

特定の週、月、年に記録されたデータに基づき選択できます。

また、作業シフトに対応する特定の時間帯に記録されたデータのみを抽出したり、週末 (土曜日と日曜日) を含めたり除外したりできます。シフトを指定しないと、デフォルトとして毎日の 24 時間分のデータが抽出されます。デフォルトでは、週末が含まれます。

アーカイブ データの追加

アーカイブ関数は特殊な機能を備えています。選択したアーカイブ期間 (週間、月間、年間) に応じて、前回のアーカイブ期間の出力ファイルに最終日までの抽出データが含まれているかどうかが自動的に確認されます。最終日までの抽出データが含まれていない場合、その出力ファイルにデータが追加され、前回のアーカイブ期間の抽出が完了します。

たとえば、1999 年 5 月 7 日に、1999 年 5 月の月間データのアーカイブを開始すると仮定します。出力ファイル rxmo199905.mwe が作成され、5 月 1 日から現在の日付 (5 月 7 日) までのデータが収められます。

1999 年 6 月 4 日に、次の月間アーカイブ期間が呼び出されます。6 月の rxmo199906.mwe ファイルが作成される前に、前月の rxmo199905.mwe ファイルが確認されます。このファイルが完成していない場合は、データが追加され、1999 年 5 月 31 日までの抽出を完了します。次に、rxmo199906.mwe ファイルが作成され、1999 年 6 月 1 日から現在の日付 (6 月 4 日) までのデータを保持します。

別の月間 (週間、年間) アーカイブが月 (週、年) に少なくとも 1 度呼び出されると、この機能は各アーカイブ期間のファイルを完成させてから次のアーカイブ期間のファイルを作成します。

アーカイブに関するヒント

ログ ファイル データのアーカイブに関するヒントを次に示します。

- 月に 1 度、月間アーカイブ期間を指定し、各生ログ ファイルからすべての詳細データを 1 つの抽出ログ ファイルに抽出してください。
- 毎月 64 メガバイトを超えるデータがシステムで生成される場合、データを毎週抽出することをお勧めします。また、プロセス詳細データを抽出から除外することもできます。
- グローバル要約データとアプリケーション要約データを年単位で抽出すると、必要なディスク領域を最小限に抑えられます。また、このアーカイブ ファイルは、長期的な傾向変動分析に使用できます。

ログ ファイル データをアーカイブするには、次の手順を実行します。

1 メイン ウィンドウの [Logfile] メニューから [**Archive**] をクリックします。[Archive Log File Data] ダイアログ ボックスが表示されます。アーカイブするデータは、生ログ ファイル セットから抽出されます。

2 アーカイブするデータ型を [Available Data Types] 一覧から選択してから、アーカイブ期間 (Week、Month、Year のいずれか) と、対象シフトを指定します。

3 [**Archive**] ボタンをクリックして、アーカイブ処理を開始します。

ログ ファイル データのアーカイブ手順を参照するには、[Help] メニューから [**Help Topics**] を選択し、[**How Do I...?**] を選んでから、[**Archive log file data**] を選んでください。

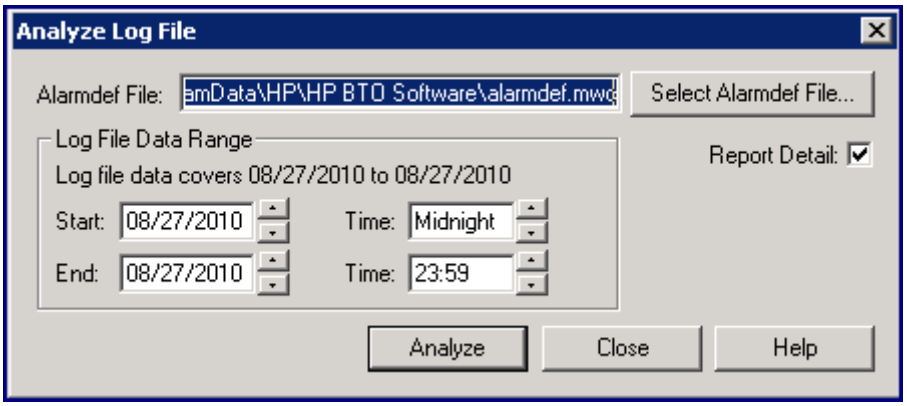
ログ ファイルの分析

アラーム定義ファイルのアラーム定義や実際に発生したアラームの活動状況に関するレポートと比較して生ログ ファイル セットのデータを分析するときには、[Logfile] メニューの [**Analyze Log File**] コマンドを使用します。

このタスクを実行すると、アラーム定義がシステムで収集された履歴データと適合しているかどうか評価できます。また、アラーム定義が分析用システム上で非常に多数のアラームを発生させるか、少数のアラームを発生させるかを決定できます。

分析する生ログ ファイルは、Performance Collection Component のデフォルト データ ソースである SCOPE で参照します。別のログ ファイルを分析するには、対象のログ ファイルを参照するデータ ソースの名前を指定する USE 文をアラーム定義に配置する必要があります。

図 8　　[Analyze Log File] ダイアログ ボックス



分析するデータの範囲

特定期間に収集されたログ ファイル データを分析できます。分析するデータの範囲を指定しないと、データは、デフォルトの開始日 (ログ ファイル内の最終日から 30 日前の日付。30 日以前のデータがない場合は、ログ ファイル内の最新レコードの日付) を使用して分析されます。

分析レポート

この作業を行うと、複数のアラーム イベントと 1 つのアラーム要約の一覧の印刷可能なレポートが生成されます (アラーム イベントの一覧は、[Analyze Log File] ダイアログ ボックスの [Report Detail] ボックスがチェックされているときにのみ表示されます)。

- アラーム イベントには、START、END、REPEAT の各アラームのステータスと、関連する PRINT 文のテキストが含まれます。また、PRINT 文のテキストが条件としてリストされている場合 (IF 文で)、その条件が真になるとテキストが含まれます。EXEC 文は実行されませんが、リストされるため、実行内容を参照できます。
- アラーム要約には、アラームの発生回数と、各アラームがアクティブな状態 (オン) であった時間が示されます。回数には、アラームの開始回数と繰り返し回数が含まれますが、アラームの終了回数は含まれません。

ログ ファイルを分析するには、次の手順を実行します。

1. メイン ウィンドウの [Logfile] メニューから [**Analyze**] をクリックします。[**Analyze Log File**] ダイアログ ボックスが表示され、現在選択しているアラーム定義ファイルの名前が示されます。
2. 別のアラーム定義ファイルを使用するには、[**Select Alarmdef File**] ボタンをクリックします。
3. 分析するログ ファイル データの範囲を選択します。
4. 分析レポートにアラーム イベントを含めるには、[**Report Detail**] ボックスをチェックします。チェックしない場合、アラーム要約のみが生成されます。
5. [**Analyze**] ボタンをクリックして分析を開始します。分析結果は [MeasureWare Agent Report Viewer] ウィンドウに表示されます。

ログ ファイルの分析手順を参照するには、[Help] メニューから [**Help Topics**] を選択し、[**How Do I...?**] を選んでから、[**Analyze a log file**] を選んでください。

ログ ファイルの走査

scopent ログ ファイルを走査してその内容に関するレポートを作成するときは、[Logfile] メニューの **[Scan Log File]** コマンドを使用します。特定期間に収集されたデータを検索するためにログ ファイル全体、およびログ ファイルの一部を走査できます。

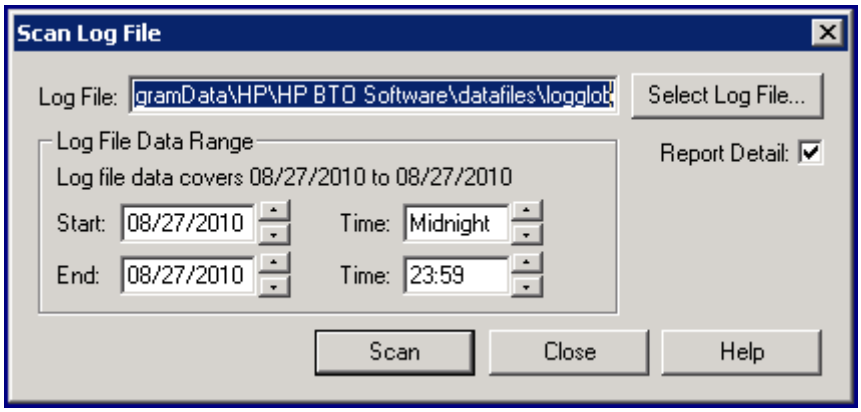
走査の結果生成されるレポートは、12 のセクションから構成されます。次の 4 つのセクションが必ず出力されます。

- プロセス要約レポート
- コレクタの適用範囲の要約
- ログ ファイルの内容の要約
- ログ ファイルの空きスペースの要約

次の 8 つのセクションは、[Scan Log File] ダイアログ ボックスで **[Report Detail]** を選択したときにのみ出力されます。

- 初期 parm ファイルのグローバル情報およびシステム構成情報
- 初期 parm ファイルのアプリケーション定義
- parm ファイルのグローバル変更内容
- parm ファイルのアプリケーションおよび変更通知
- コレクタのオフタイム通知
- アプリケーション固有の要約レポート

図 9　　[Scan Log File] ダイアログ ボックス



ログ ファイルを走査するには、次の手順を実行します。

1. メイン ウィンドウの [Logfile] メニューから **[Scan Log File]** をクリックします。[Scan Log File] ダイアログ ボックスが表示され、現在オープンしているログ ファイルの名前が反転表示されます。
2. 別のログ ファイルを走査するには、**[Select Log File]** ボタンをクリックします。
3. 特定期間に記録されたデータを走査するには、[Log File Data Range] でその期間の開始日時および終了日時を選択するか入力してください。
4. 完全な走査レポートが必要な場合、**[Report Detail]** ボックスをチェックします。チェックしない場合、レポートのサブセットのみが生成されます。

5 走査プロセスを開始するには、[**Scan**] ボタンをクリックします。走査結果は [Performance Collection Component Report Viewer] ウィンドウに表示されます。

ログ ファイル データの走査手順を参照するには、[Help] メニューから [**Help Topics**] を選択し、[**How Do I...?**] を選んでから、[**Scan a log file**] を選んでください。

ログ ファイルのサイズ変更

scopent の生ログ ファイルのサイズを変更するときには、[Logfile] メニューの [**Resize Log File**] コマンドを使用します。サイズ変更は、特定のファイルのメガバイト単位のサイズを使用するか、ファイルにデータを保持する日数を使用して行います。

生ログ ファイルの最大サイズは parm ファイルの size パラメータで指定します。ログ ファイルのサイズを変更すると、ログ ファイル データのロール バック頻度を制御しやすくなります。

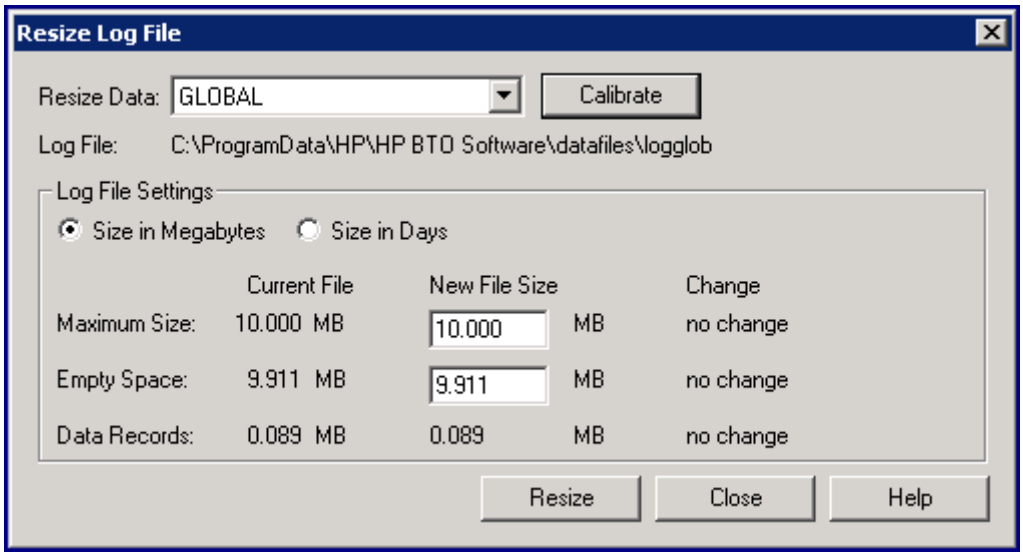
サイズを変更するデータ型 (グローバル、アプリケーション、プロセス、デバイス、トランザクションのいずれか) は選択できます。各データ型は、logglob、logappl、logproc、logdev、logtran の各生ログ ファイルに対応しています。次に、サイズ変更の実行方法 (メガバイト数の変更または日数による変更) を選択します。選択したサイズ変更方法に応じて、[Resize Log File] ダイアログ ボックスの [Log File Settings] ボックスには、次のようなフィールドが示されます。

- [Maximum Size] 各フィールドには、現在のファイル サイズ、新しいファイル サイズ、サイズ変更内容が示されます。
- [Empty Space] の各フィールドには、現在のファイルの空き容量、サイズ変更プロセスの完了後にファイルに必要な容量、変更内容が示されます。これらの値は、サイズ変更プロセスで削除する必要があるデータがログ ファイル内にあるかどうかを判断するときに使用します。
- [Data Records] の各フィールドには、現在のログ ファイルに含まれているデータ レコードの量と、サイズ変更後にログ ファイルに含まれるデータ レコードの量が示されます。

ログ ファイルのサイズはメガバイト単位で指定します。メガバイトではなく日数でサイズを指定する方が便利な場合もあります。[Size in Days] を選択すると、ダイアログ ボックスのすべての単位が「days」に変わります。メガバイトから日数への換算は、各データ型の「1 日あたりのメガバイト数」に基づいて行われます。この換算には、最初は推定値が使用されます。

より正確な値が、[**Calibrate**] ボタンをクリックすることで得られます。この補正関数は、既存のログ ファイルを実際に測定して、より正確な 1 日あたりのメガバイト数を算出します。メガバイト単位でサイズを指定している場合、換算は不要であるため、補正関数を使用する必要はありません。

図 10　　[Resize Log File] ダイアログ ボックス



ログ ファイルのサイズを変更する前に、scopent コレクタを終了する必要があります。scopent を停止するには、39 ページの「データ収集の終了と再開始」の手順を実行します。

scopent を終了せずにログ ファイルのサイズ変更を行っても、既存のログ ファイルのサイズは変更されません。ログ ファイルのサイズを変更するには、次の手順を実行します。

1. scopent を終了してから、メイン ウィンドウの [Logfile] メニューから [**Resize Log File**] を選択して、[Resize Log File] ダイアログ ボックスを表示します。
2. [Resize Data] ボックスで、サイズを変更するデータ型 (グローバル、アプリケーション、プロセス、デバイス、トランザクションのいずれか) を選択します。
3. [**Size in Megabytes**] か [**Size in Days**] を選択します。選択した指定に応じて、[Current File] および [New File Size] が表示されます。
4. [New File Size] の表示に基づいてサイズ変更を行うには、[**Resize**] ボタンをクリックしてサイズ変更プロセスを開始します。

日数でサイズを決定するときにログ ファイルの追加容量をより正確に見積もるには、次の手順を実行します。

1. [**Calibrate**] ボタンをクリックします。過去 30 日間にファイルに記録されたデータ レコードの実数およびサイズが直ちに表示されます。
2. [**Close**] ボタンをクリックして、[Resize Log File] ダイアログ ボックスに戻ります。日数でサイズを決定する場合、補正に基づいて算出された [New Current File] および [New File Size] の値が表示されます。
3. [**Resize**] ボタンをクリックしてログ ファイルのサイズを変更します。
4. 別のタスクを実行する前に、39 ページの「データ収集の終了と再開始」の手順を使用して scopent を開始します。

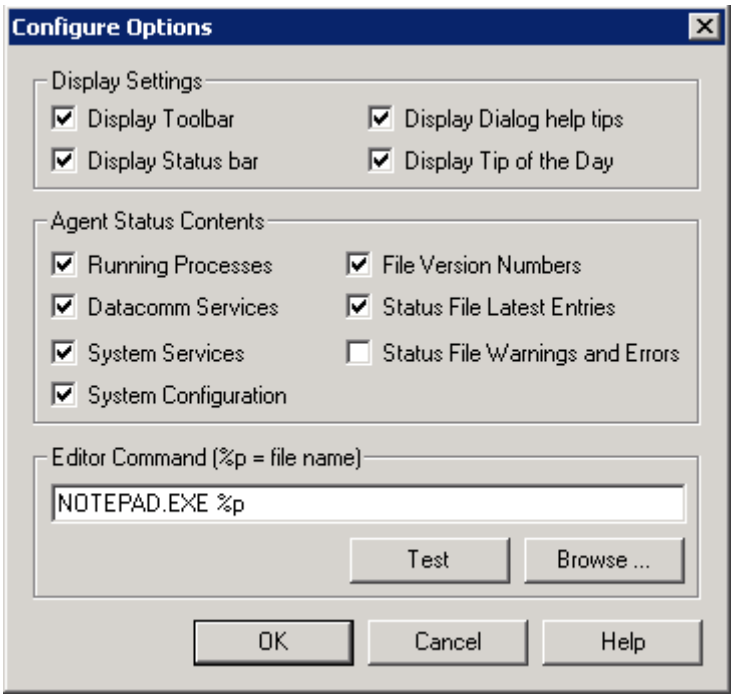
ログ ファイルのサイズ変更手順を参照するには、[Help] メニューから [**Help Topics**] を選択し、[**How Do I...?**] を選んでから、[**Resize a log file**] を選んでください。

ユーザー オプションの構成

Performance Collection Component の使用中またはメイン ウィンドウ上で、ツールバー、ステータス バー、ダイアログ ヘルプのヒント、[Tip of the Day] の表示を制御するときには、[Configure] メニューの [**Options**] コマンドを使用します。

また、[**Options**] コマンドを使用して、収集パラメータやアラーム定義ファイルを変更するエディタやワード プロセッサを構成したり、[Agent] メニューの [**Status**] コマンドを選択するときに表示するステータス情報の種類を選択できます。

図 11　　[Configure Options] ダイアログ ボックス



ユーザー オプションを設定するには、次の手順を実行します。

1. メイン ウィンドウの [Configure] メニューから [**Options**] コマンドをクリックして、[Configure Options] ダイアログ ボックスを表示します。
2. [**Display Toolbar**] チェック ボックスを選択して、メイン ウィンドウにツールバーを表示します。
3. [**Display Status Bar**] チェック ボックスを選択して、各ダイアログ ボックスの下部とメイン ウィンドウに現在のステータスを表示します。
4. [**Display Dialog Help Tips**] チェック ボックスを選択して、ダイアログ ボックスにヘルプのヒントを表示します。
5. [Display Tip of the Day] チェック ボックスを選択して、[Performance Collection Component] メイン ウィンドウをオープンしたときにその日のヒントを表示します。

設定するには、次の手順を実行します。

1. ファイル名拡張子 `.exe` を使用してエディタのディレクトリ パスおよびファイル名を [Editor Command] ボックスに入力します (`C:\MSOffice\winword\winword.exe` など)。

2 [**Browse**] ボタンをクリックして、[Select a Text Editor] ダイアログ ボックスを表示します。このダイアログ ボックスからエディタを選択できます。

3 [**Test**] ボタンをクリックして、選択したエディタが構成されていることを確認してから、[**OK**] をクリックします。

表示するエージェントステータス情報を設定するには、[Agent Status Contents] に示されているオプション ボックスを 1 つまたは複数選択した後、[**OK**] をクリックします。

ユーザー オプションの構成手順を参照するには、[Help] メニューから [**Help Topics**] を選択し、[**How Do I...?**] を選んでから、[**Configure user options**] を選んでください。

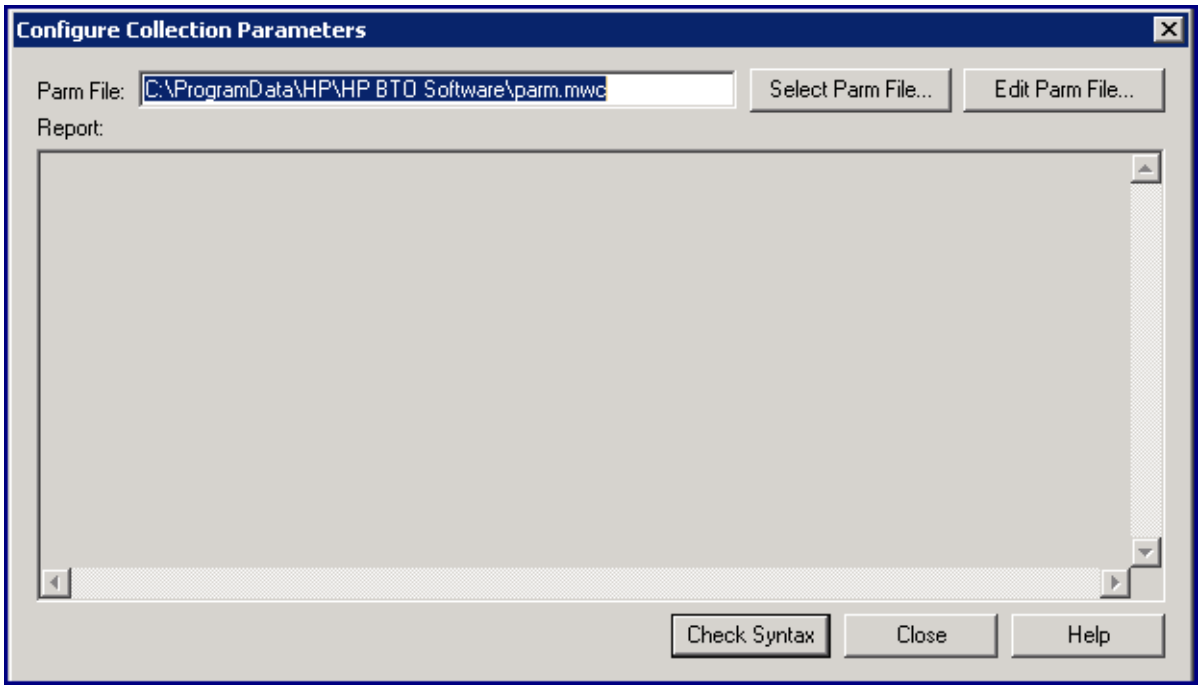
収集パラメータの構成

scopent でデータ収集に使用される parm ファイルの構文を確認するときに、[Configure] メニューの [**Collection Parameters**] コマンドを使用します。parm ファイルの設定値を調べると、構文エラーや警告を見つけ、アプリケーションの定義に使用できる容量を確認できます。

警告やエラーが見つかり、それらを修正する場合や、parm ファイルのパラメータを変更または追加する場合、[Edit Parm File] 機能を使用すると parm ファイルを簡単に変更できます。

parm ファイルとそのパラメータの詳細は、15 ページの「データ収集の管理」を参照してください。

図 12　[Configure Collection Parameters] ダイアログ ボックス



構文を確認するには、次の手順を実行します。

1 Performance Collection Component メイン ウィンドウの [Configuration] メニューから [**Collection Parameters**] をクリックします。[Configure Collection Parameters] ダイアログ ボックスが表示され、現在オープンしている parm.mwc ファイルの名前が [Parm File] ボックスに示されます。

2 別の parm ファイルを確認するには、[**Select Parm File**] ボタンをクリックします。

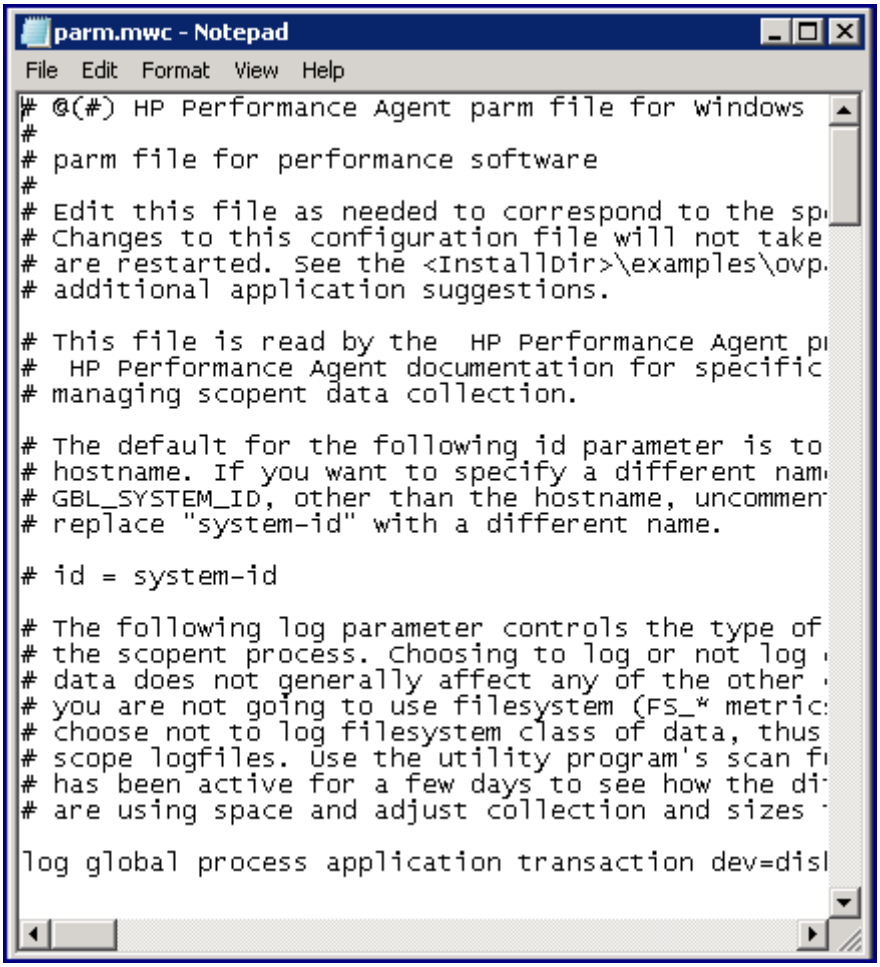
3 parm ファイルの構文を確認するには、[**Check Syntax**] ボタンをクリックします。警告またはエラーがある場合、[Performance Collection Component] ウィンドウに表示されます。

4 parm ファイルのいずれかの箇所を変更する場合、[**Edit Parm File**] ボタンをクリックします。[Edit Parm File] ダイアログ ボックスと [Configure Collection Parameters] ダイアログ ボックスは、同時に使用できるように画面上に配置できます。

parm ファイルの構文確認手順を参照するには、[Help] メニューから [**Help Topics**] を選択し、[**How Do I...?**] を選んでから、[**Check the syntax of a collection parameters file**] を選んでください。

収集パラメータ ファイルの変更

図 13　[Modify Collection Parameters File] ウィンドウ



parm ファイルを変更するには、次の手順を実行します。

1 Performance Collection Component メイン ウィンドウの [Configuration] メニューから [**Collection Parameters**] をクリックして、[Configure Collection Parameters] ダイアログ ボックスの [**Edit Parm File**] ボタンをクリックします。現在オープンしている parm ファイルの内容が、事前に指定したエディタまたはワード プロセッサに表示されます（エディタやワード プロセッサを指定するには、231 ページの「ユーザー オプションの構成」を参照してください）。

2 ファイルを変更する前に、18 ページの「parm ファイル」を参照して、規則や記述方法を参考にしてください。

3 必要に応じてファイルを変更し、テキスト フォーマットで保存します。

> Performance Collection Component の実行中、Windows の **WordPad** エディタで parm ファイルを変更することはできません。Performance Collection Component を終了してから、**WordPad** を使用した後、Performance Collection Component を再開始する必要があります。ただし、**Notepad** エディタは、Performance Collection Component の実行中でも、ファイルの変更に使用できます。

別の作業を始める前に、parm ファイルに対して行った変更を有効にする必要があります。次の手順を実行します。

1 [MeasureWare Services] ウィンドウを開くには、Performance Collection Component のメイン ウィンドウにある [Agent] メニューから [**Start/Stop**] をクリックします。

2 [**Data Collection**] チェック ボックスを選択します。

3 [**Refresh**] ボタンをクリックします。

4 [**Close**] ボタンをクリックして、メイン ウィンドウに戻ります。

parm ファイルの変更手順を参照するには、[Help] メニューから [**Help Topics**] を選択し、[**How Do I...?**] を選んでから、[**Modify a collection parameters file**] を選んでください。

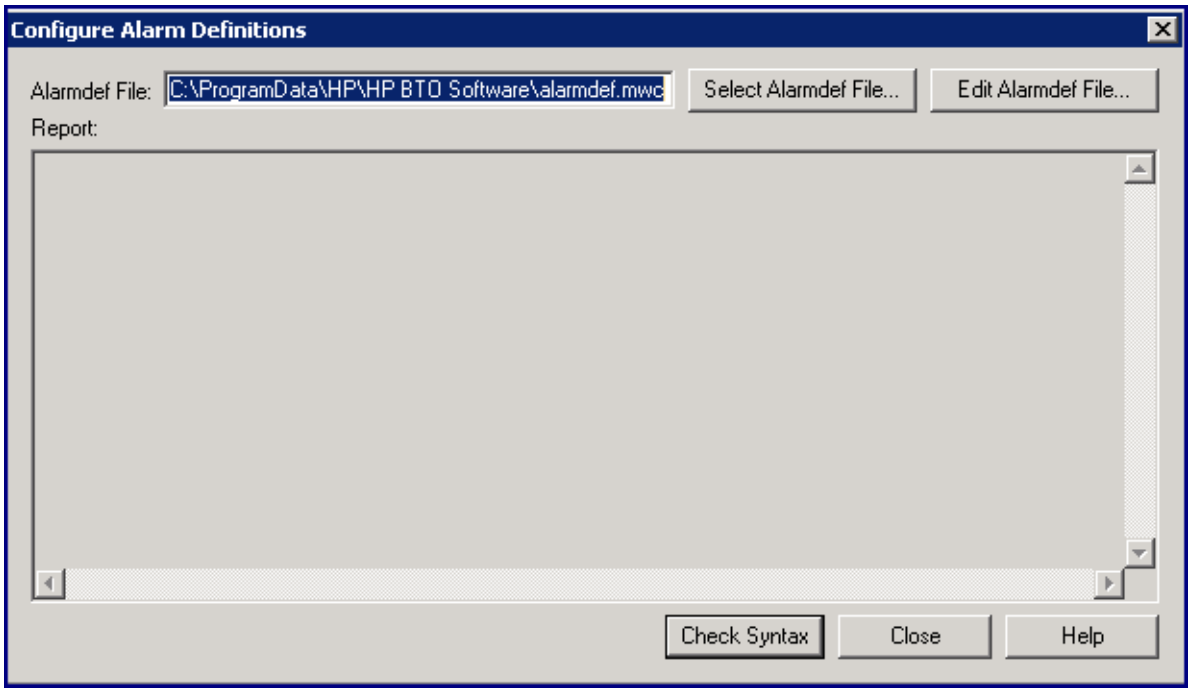
> WordPad、Notepad、Microsoft Word のいずれかを使用して parm.mwc ファイルを変更した後、[名前を付けて保存] コマンドを使用してファイルを保存すると、デフォルトの .txt 拡張子が自動的に付けられ、ファイル名は parm.mwc.txt になります。ファイル名 parm.mwc で保存するには、[名前を付けて保存] コマンドを使用して、ファイルをテキスト ファイルとして保存することを指定し、ファイル名を二重引用符 (") で囲みます。たとえば、"parm.mwc" です。

アラーム定義の構成

アラーム定義ファイル (`alarmdef.mwc`) 内のアラーム定義の構文を確認するときには、[Configure] メニューから [**Alarm Definitions**] コマンドを使用します。アラーム定義の構文が正しいことを確認した後、アラーム定義と比較してログ ファイルを分析し、履歴ログ ファイル データ内のアラームを確認できます (226 ページの 「ログ ファイルの分析」を参照してください)。

警告やエラーが見つかり、それらを修正する場合や、アラーム定義を追加または削除する場合、[Configure Alarm Definitions] ダイアログ ボックス内の [**Edit Alarmdef File**] ボタンを使用すると、アラーム定義ファイルを簡単に変更できます。

図 14　　[Configure Alarms Definitions] ダイアログ ボックス



構文を確認するには、次の手順を実行します。

1. Performance Collection Component メイン ウィンドウの [Configure] メニューから [**Alarm Definitions**] をクリックします。[Configure Alarm Definitions] ダイアログ ボックスが表示され、現在オープンしているアラーム定義ファイルの名前が示されます。
2. 別のアラーム定義ファイルを確認するには、[**Select Alarmdef File**] ボタンをクリックします。
3. [**Check Syntax**] ボタンをクリックして、構文確認プロセスを開始します。数秒後、[Performance Collection Component Report Viewer] ウィンドウに警告やエラーを含めた確認結果が表示されます。
4. アラーム定義ファイルのいずれかの箇所を変更する場合、[**Edit Alarmdef File**] ボタンをクリックします。

アラーム定義ファイルの構文確認手順を参照するには、[Help] メニューから [**Help Topics**] を選択し、[**How Do I...?**] を選んでから、[**Checking the syntax of an alarm definitions file**] を選んでください。

アラーム定義ファイルの変更

alarmdef ファイルを変更するには、次の手順を実行します。

1 Performance Collection Component メイン ウィンドウの [Configure] メニューから [**Alarm Definitions**] を選択して、[Configure Alarm Definitions] ダイアログ ボックスの [**Edit Alarmdef File**] ボタンをクリックします。現在オープンしているアラーム定義ファイルの内容が、事前に指定したエディタまたはワード プロセッサに表示されます (エディタやワード プロセッサを設定するには、231 ページの「ユーザー オプションの構成」を参照してください)。

2 ファイルを変更する前に、157 ページの「アラーム構文のリファレンス」を参照して、アラーム定義の詳細を確認してください。

3 必要に応じてファイルを変更し、テキスト フォーマットで保存します。

WordPad、Notepad、Microsoft Word のいずれかを使用してアラーム定義ファイルを変更した後、[**名前を付けて保存**] コマンドを使用してファイルを保存すると、デフォルトの `.txt` 拡張子が自動的に付けられ、ファイル名は `alarmdef.mwc.txt` になります。ファイル名 `alarmdef.mwc` で保存するには、[名前を付けて保存] コマンドを使用して、ファイルをテキスト ファイルとして保存することを指定し、ファイル名を二重引用符 (") で囲みます。たとえば、"`alarmdef.mwc`" です。

変更の有効化

別の作業を始める前に、アラーム定義ファイルに対して行った変更を有効にする必要があります。次の手順を実行します。

1 [MeasureWare Services] ウィンドウを開くには、Performance Collection Component のメイン ウィンドウにある [Agent] メニューから [**Start/Stop**] をクリックします。

2 [**Alarm Definitions**] チェック ボックスをオンにします。

3 [**Refresh**] ボタンをクリックします。

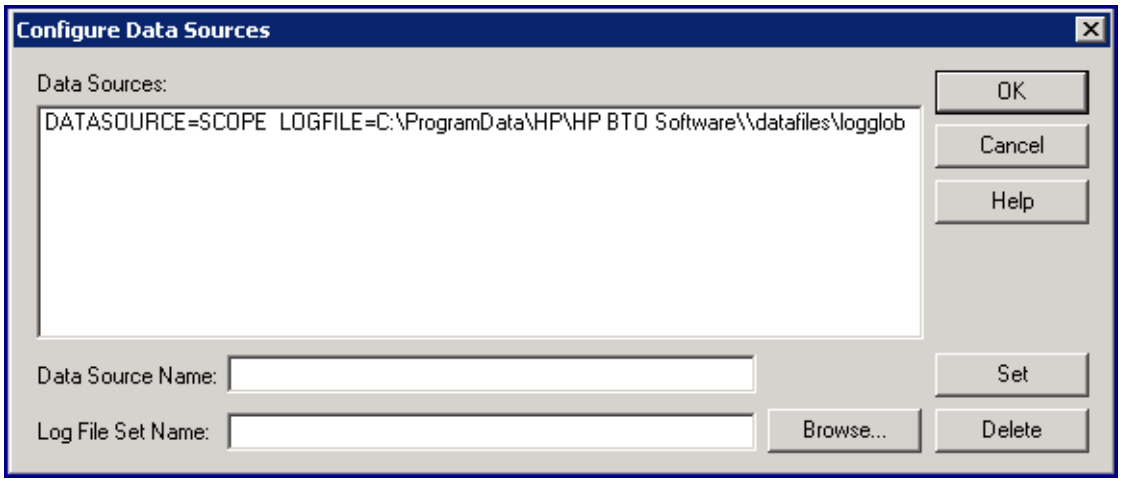
4 [**Close**] ボタンをクリックして、メイン ウィンドウに戻ります。

アラーム定義ファイルの変更手順を参照するには、[Help] メニューから [**Help Topics**] を選択し、[**How Do I...?**] を選んでから、[**Modify an alarm definitions file**] を選んでください。

データ ソースの構成

Performance Collection Component では、`scopent` ログ ファイルや DSI ログ ファイルなど個別のデータ ソースに対して、それぞれデータ ソースを使用します。各データ ソースは単一のログ ファイル セットで構成されます。データ ソースは、`<データ ディレクトリ >\conf\perf` ディレクトリにある `datasources` ファイルに構成します。インストール後に Performance Collection Component を最初に起動した際には、既に SCOPE という名前のデフォルトのデータ ソースが構成されており、`scopent` ログ ファイル セットが使用できます。

図 15　[Configure Data Sources] ダイアログ ボックス



データ ソース ファイルのフォーマット

datasources ファイルにある各エントリは、1 つのログ ファイル セットで構成されるデータ ソースを表します。エントリにより示されるデータ ソース名は、リポジトリ サーバーを識別し、データの格納場所を特定するために使用されるものです。エントリは、大文字と小文字が区別されません。構文は次のようになります。

datasource=*datasource_name* **logfile=***logfile_set*

- **datasource** はキーワードです。*datasource_name* は、アラーム定義または分析ソフトで使用するデータ ソースを識別するための名前です。データ ソース名は固有のものである必要があります。*datasource_name* の長さは 64 文字以内です。
- **logfile** はキーワードです。*logfile_set* は、ログ ファイル セットを識別する完全修飾名です。logfile_set には、scopent により作成された生ログ ファイル セット、extract タスクにより作成された抽出ログ ファイル、DSI ログ ファイル セットのいずれかを指定できます。スペースを含んだログ ファイルのパス名を指定する場合、パス名を二重引用符 (") で囲む必要があります。

scopent ログ ファイル セットを指定するときには、ファイル名 logglob のみを使用してください。生ログ ファイルは 1 つのログ ファイル セットとしてアクセスされるため、他の生ログ ファイル名を指定する必要はありません。

DSI ログ ファイル セットを指定する場合も同じです。DSI ルート ファイルの名前のみを指定してください。DSI ログ ファイル セットの他のいずれかのファイルを指定する必要はありません。

リモート ロケーションからのデータ ソースの構成

ネットワーク化された共用環境に存在するログ ファイル セットを指定するときには、ユニバーサル命名規則 (UNC: universal naming convention) が必要です。Performance Collection Component サービスはシステムの起動時に自動的に開始されますが、リモート接続されたファイル システムに関するドライブ マッピングは、ユーザーがログオンするまで確立されません。したがって、ログ ファイルの参照にドライブ マッピング名を使用するデータ ソースがリモート システムに存在する場合、Coda は無効なデータ ソース エラーを生成します。ログオンした後に Performance Collection Component サービスを開始すると、ドライブ マッピングが確立されているため、データ ソースは処理されます。

データ ソース エントリの 3 つの例を次に示します。

例 1 :

以下の例では、デフォルトのディレクトリ
< ディスク ドライブ >:\Program Files\HP\HP BTO Software\data\datafiles\ にあるデフォルトの SCOPE データ ソースを示します。

```
datasource=SCOPE logfile="C:\Program Files\HP\HP BTO
Software\data\datafiles\logglob"
```

例 2 :

以下の例では、ネットワーク共用環境に存在するログ ファイルを指定するためにユニバーサル命名規則 (UNC) を使用します。

```
datasource=RXLOG logfile=\\lab_sys\my_share\rxlog
```

例 3 :

以下の例では、パス名にスペースを含んだディレクトリに存在する SCOPE データ ソースを示します。

```
datasource=SCOPE logfile="C:\Program Files\HP\HP BTO Software\/
data\donna test\logglob"
```

データ ソースを設定するには、Performance Collection Component メイン ウィンドウの [Configure] メニューから [**Data Sources**] をクリックします。

[Configure Data Sources] ダイアログ ボックスが表示され、現在のデータ ソース エントリが一覧で表示されます。それぞれのエントリが 1 つのデータ ソースを表しています。

変更するには、次の手順を実行します。

1 [Data Sources] 一覧でデータ ソースを選択します。

2 [**Log File Set Name**] ボックスをクリックし、ログ ファイル セット名を変更して [**Set**] ボタンをクリックします。

新しいデータ ソースを追加するには、次の手順を実行します。

1 [**Data Source Name**] ボックスをクリックし、新しい名前を入力します。

2 [**Log File Set Name**] ボックスをクリックし、新しいログ ファイル セットの完全修飾名を入力して、[**Set**] ボタンをクリックします。

または

3 [**Browse**] ボタンをクリックして、既存のデータ ソースを選択します。

データ ソースを削除するには、次の手順を実行します。

1 [Data Sources] 一覧でデータ ソースを選択します。

2 [**Delete**] ボタンをクリックします。

3 データ ソース ファイルの構成が終了した後、[**OK**] をクリックします。

変更内容を有効にする手順

別の作業を始める前に、データ ソースに対して行った変更を有効にする必要があります。次の手順を実行します。

1. [MeasureWare Services] ウィンドウを開くには、Performance Collection Component のメイン ウィンドウにある [Agent] メニューから [**Start/Stop**] を選択します。
2. [**Stop Services**] ボタンをクリックして、MeasureWare サービスを終了します。
3. [**Stop Services**] ボタンが淡色表示になっているときは、[**Start Services**] ボタンをクリックします。
4. [**Close**] ボタンをクリックして、メイン ウィンドウに戻ります。

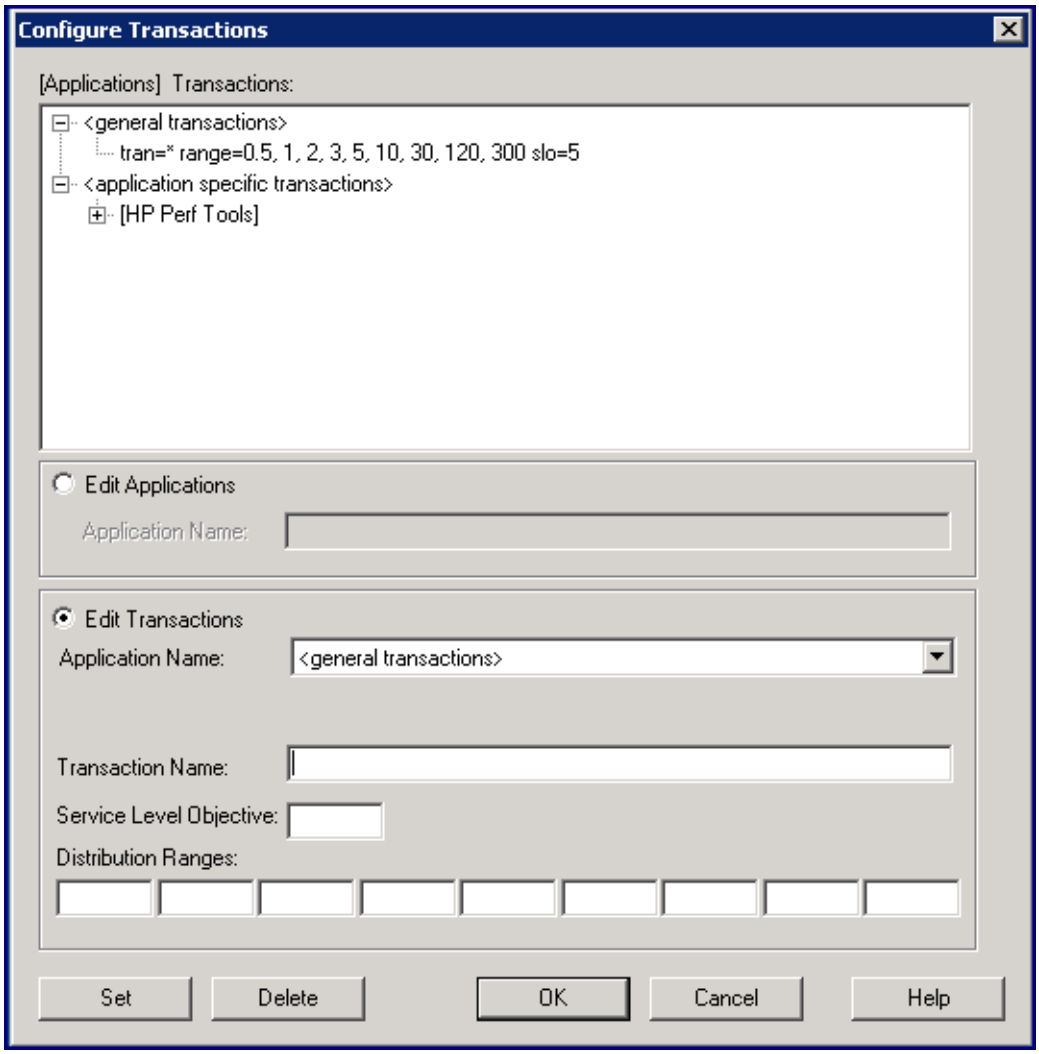
データ ソース ファイルの変更手順を参照するには、[Help] メニューから [**Help Topics**] を選択し、[**How Do I...?**] を選んでから、[**Modify a data source file**] を選んでください。

トランザクションの構成

アプリケーションに合わせてトランザクション データの収集をカスタマイズするときには、トランザクション構成ファイル `ttdconf.mwc` を使用します。このファイルは、トランザクション名、パフォーマンス区分範囲、各トランザクションのサービス レベル目標を定義します。オプションで、アプリケーションに固有のトランザクションも定義できます。

デフォルトの `ttdconf.mwc` ファイルには 3 つのエントリがあります。2 つのエントリは、Performance Collection Component `scopent` コレクタが使用するトランザクションを定義し、もう 1 つのエントリ `tran=*` は、アプリケーション応答測定 (ARM: Application Response Measurement) API 関数コールを備えたアプリケーション内のすべてのトランザクションを登録します。

図 16　　[Configure Transactions] ダイアログ ボックス



デフォルトの ttdconf.mwc ファイルにある tran=* エントリのサービス レベル目標と範囲値を使用するようなシステムに新しいアプリケーションを追加する場合、新しいトランザクションを組み込むために何も行う必要はありません。すべてのデフォルト値は自動的に適用されます。

ただし、固有のサービス レベル目標と区分範囲値が設定されているトランザクションを持つシステムにアプリケーションを追加する場合、それらのトランザクションを ttdconf.mwc ファイルに追加する必要があります。

ttdconf.mwc ファイル内のエントリの順序は任意です。完全に一致するエントリが最初に検索されます。エントリが見つからない場合、末尾にアスタリスク (*) があり、先頭からの一致文字数が最も多いものが使用されます。

ファイルを変更する前に、335 ページの「トランザクション追跡の紹介」を参照して、構成ファイルのフォーマット、トランザクションとアプリケーションの名前、パフォーマンス区分範囲、サービス レベル目標を参考にしてください。設定するには、Performance Collection

Component メイン ウィンドウの [Configure] メニューから [**Transactions**] をクリックすると、[Configure Transactions] ダイアログ ボックスが表示されます。このダイアログ ボックスを使用して、次の作業を実行できます。

1 一般トランザクションを追加する。

2 アプリケーション固有のトランザクションを追加する。

3 トランザクションのパフォーマンス区分範囲またはサービス レベル目標を変更する。

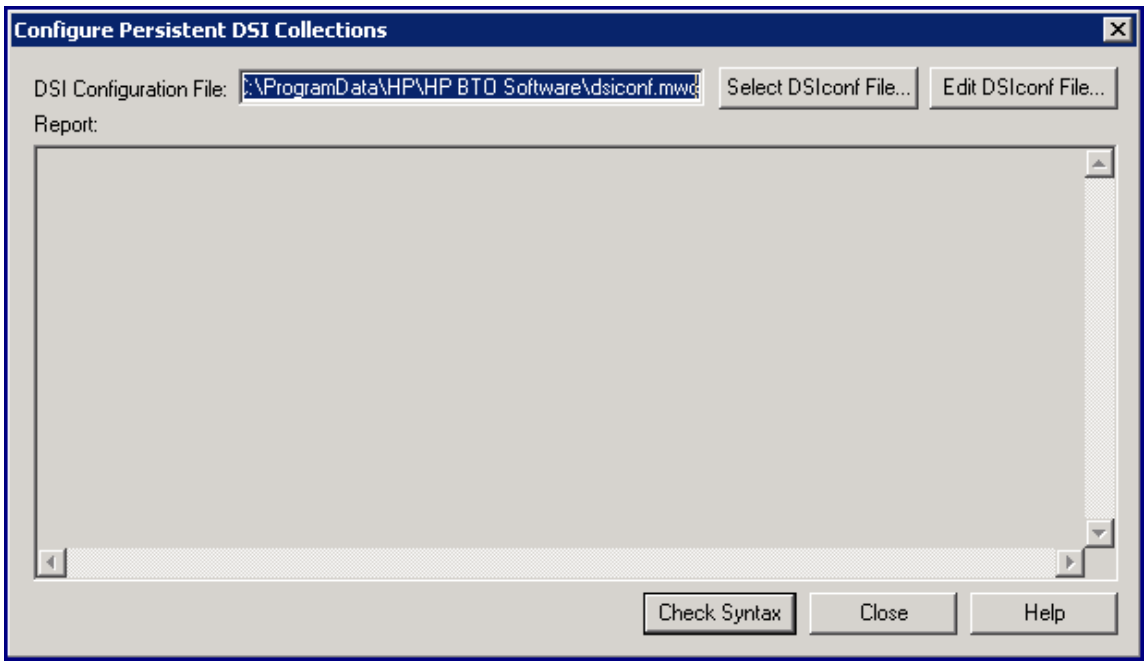
4 トランザクションを削除する。

これらのタスクの実行手順を参照するには、[Help] メニューから [**Help Topics**] を選択し、[**How Do I...?**] を選んでから、[**Configure transactions**] を選んでください。

Persistent DSI Collections の構成

DSI 構成ファイル `dsiconf.mwc` の構文を確認または修正するには、[Configure] メニューの [**Persistent DSI Collections**] コマンドを使用します。`dsiconf.mwc` ファイルは、外部ソースから Performance Collection Component 内に取り込まれたデータ収集の継続的な記録を構成するのに使用します。詳細情報は、247 ページの「データ ソース統合の概要」を参照してください。

図 17 [Configure Persistent DSI Collections] ダイアログ ボックス



DSI 構成ファイルの構文を確認するには、次の手順を実行します。

1 Performance Collection Component メイン ウィンドウの [Configure] メニューから [**Persistent DSI Collections**] をクリックします。[Configure Persistent DSI Collections] ダイアログ ボックスが表示され、現在オープンしている `dsiconf.mwc` ファイルの名前が示されます。

2 別の `dsiconf.mwc` ファイルを確認するには、[**Select DSIconf File**] ボタンをクリックします。

3 ファイルの構文を確認するには、[**Check Syntax**] ボタンをクリックします。警告またはエラーがある場合、[Performance Collection Component] ウィンドウに表示されます。

4 ファイルのいずれかの箇所を変更する場合、[**Edit DSIconf File**] ボタンをクリックします。[Edit DSIconf File] ダイアログ ボックスと [Configure Persistent DSI Collections] ダイアログ ボックスは、同時に使用できるように画面上に配置できます。

DSI 定義ファイルの構文確認手順を参照するには、[Help] メニューから [**Help Topics**] を選択し、[**How Do I...?**] を選んでから、[**Check the syntax of a DSI configuration file**] を選んでください。

DSI 設定ファイルを変更するには、次の手順を実行します。

1 Performance Collection Component メイン ウィンドウの [Configure] メニューから [**Persistent DSI Collections**] をクリックし、[Configure Persistent DSI Collections] ダイアログボックスの [**Edit DSIconf File**] ボタンをクリックします。現在オープンしている `dsiconf.mwc` ファイルの内容が、事前に指定したエディタまたはワード プロセッサに表示されます (エディタやワード プロセッサを指定するには、231 ページの「ユーザー オプションの構成」を参照してください)。

2 ファイルを変更する前に、211 ページの「Windows での Performance Collection Component の使用」を参照して、規則や記述方法を参考にしてください。

3 必要に応じてファイルを変更し、テキスト フォーマットで保存します。

別の作業を始める前に、`dsiconf.mwc` ファイルに対して行った変更を有効にする*必要があります*。次の手順を実行します。

1 [MeasureWare Services] ウィンドウを開くには、Performance Collection Component のメイン ウィンドウにある [Agent] メニューから [**Start/Stop**] をクリックします。

2 [**Persistent DSI Collections**] チェック ボックスを選択します。

3 [**Refresh**] ボタンをクリックします。

4 [**Close**] ボタンをクリックして、メイン ウィンドウに戻ります。

DSI 構成ファイルの変更手順を参照するには、[Help] メニューから [**Help Topics**] を選択し、[**How Do I...?**] を選んでから、[**Modify a DSI configuration file**] を選んでください。

WordPad、Notepad、Microsoft Word のいずれかを使用して `dsiconf.mwc` ファイルを変更した後、[名前を付けて保存] コマンドを使用してファイルを保存すると、デフォルトの `.txt` 拡張子が自動的に付けられ、ファイル名は `dsiconf.mwc.txt` になります。ファイル名 `dsiconf.mwc` で保存するには、
[名前を付けて保存] コマンドを使用して、ファイルをテキスト ファイルとして保存することを指定し、ファイル名を二重引用符 (") で囲みます。たとえば、"`dsiconf.mwc`" です。

Performance Collection Component のステータス確認

Performance Collection Component プロセスの現在のステータスを調べるときには、[Agent] メニューから [**Status**] コマンドを使用します。この情報は `perfstat` プログラムにより生成されます。

ステータス レポートにどの情報を含めるかは、[Configure] メニュー画面から [**Options**] コマンドを選択し、[Configure Options] ダイアログ ボックスで次のいずれかのオプションを選択することで指定できます。

実行中のプロセス

Performance Collection Component の現在実行されているバックグラウンド プロセスおよびフォアグラウンド プロセスの一覧が表示されます。実行中でなければならないバックグラウンド プロセスは、実行中でなくても表示されます。

データコム サービス

データコム サービスは、Performance Collection Component データコム サービスを特定して、そのサービスと通信します。アラーム ジェネレータ データベース サーバー (agdbserver) プロセスが実行中で応答可能な状態であるかどうかを示します。データ通信が無効になっている場合、データコム サービスの応答の待機中、この情報が生成されるまでに 30 秒以上の時間を要することがあります。

システム サービス

Scope Collector、Transaction Manager、Measurement Interface などの Performance Collection Component システム サービスの現在のステータスを示します。

システム構成

システム名、オペレーティング システムのバージョン、プロセッサの種類です。

ファイル バージョン番号

Performance Collection Component ファイルのバージョン番号です。欠落している重要なファイルが通知されます。

ステータス ファイルの最新エントリ

各パフォーマンス ツール ステータス ファイルから最新のエントリをいくつか表示します。

ステータス ファイルの警告とエラー

パフォーマンス ツール ステータス ファイルから文字列「Error」または「Warning」を含む行をリストします。警告が長期間無視されている場合、非常に大きなリストが作成される可能性があります。

現在のステータスの一覧を表示するには、Performance Collection Component メイン ウィンドウの [Agent] メニューから [**Status**] をクリックします。[Configure Options] ダイアログ ボックスから選択した情報が、[Performance Collection Component Report Viewer] に表示されます。

すべてのステータス情報の完全なレポートを作成するには、[Agent] メニューから [**Report**] をクリックします。すべてのステータス情報の完全なリストが、[Performance Collection Component Report Viewer] に表示されます。

Performance Collection Component ステータスの確認手順を参照するには、[Help] メニューから [**Help Topics**] を選択し、[**How Do I...?**] を選んでから、[**Check status of Performance Collection Component processes**] を選びます。

Windows のコマンド プロンプトからも `perfstat` プログラムを実行できます。

パフォーマンス カウンタ収集の構築

Performance Collection Component は Windows のパフォーマンス カウンタにアクセスできます。パフォーマンス カウンタは、システム、システム上のアプリケーション、デバイス パフォーマンスを測定するのに使用します。データ収集を構築するために特定のパフォーマンス カウンタを選択するには、拡張収集ビルダ / マネージャ (ECBM) を使用します。

パフォーマンス カウンタ収集の構築

収集を構築するには、Performance Collection Component メイン ウィンドウの [Agent] メニューから [**Extended Collections**] を選択します。[Extended Collection Builder and Manager] ウィンドウが表示されます。左側のペインには Windows オブジェクトがリストされます。収集の構築手順を参照するには、[Extended Collection Builder and Manager] ウィンドウの [Help] メニューから [**Help Topics**] を選択してください。

Windows のパフォーマンス カウンタ収集を構築した後、最下部の [Extended Collection Manager] ペインを使用して、新規または既存の収集を登録、開始、または終了します。

パフォーマンス カウンタ収集の管理

データ収集を管理するには、[Extended Collection Builder and Manager] の最下部にある [Extended Collection Manager] ペインを使用します。最初、収集は表示されません。データの収集を開始する前に収集を登録する必要があります。

作成した収集の登録または格納後、[Extended Collection Manager] ペインに現在の収集のリストが表示されます。[Extended Collection Manager] ペインには各収集のステータスも表示され、収集に関する情報 (プロパティ) を確認できます。収集の管理手順を参照するには、[Extended Collection Builder and Manager] ウィンドウの [**Help**] メニューから [**Help Topics**] を選択してください。

拡張収集ビルダ / マネージャの使用のヒント

- `<インストール ディレクトリ >\paperdocs\mwa\C\monxref.txt` ファイルは、Windows のパフォーマンス カウンタおよびコマンドへの Performance Collection Component メトリックのクロスリファレンスで構成されます。Performance Collection Component によって収集済みのメトリックに対する拡張収集ビルダ / マネージャを介してデータを記録すると、システム オーバーヘッドが増加します。
- Extended Collection Builder を使用して収集を作成する場合、Performance Collection Component の内部で使用するために、Windows のパフォーマンス カウンタにはデフォルトのメトリック名が割り当てられます。通常、これらのデフォルト名は意味を持たず、簡単に判読できません。メトリック名に意味を持たせたり、ソース アプリケーションによって提供

されたメトリック名に一致させるには、[Extended Collection Builder and Manager] ウィンドウの左ペインから右ペインにメトリック名をドラッグしてから、メトリック名を右クリックまたはダブル クリックします。(詳しい手順は、拡張収集ビルダ / マネージャのオンラインヘルプを参照してください。)

- 61 以上の収集を起動する場合、60 を超える収集がエラー状態になります。この場合、他の収集に不具合が生じるおそれがあります。
- **Wolfpack** で構成されたシステムから論理ディスク メトリックを収集する場合、収集を登録した際には存在しなかった新しいディスク インスタンスのデータを収集するために、収集を再開する必要があります。
- 収集を完全に削除するには、収集を削除した後に Performance Collection Component を再起動する必要があります。Performance Collection Component を再起動しない場合、削除操作中にエラーが表示される場合があります。通常このエラーは、一部のファイルが完全に削除されていないことを表します。Performance Collection Component を再起動後に、残ったファイルおよびディレクトリの手動による削除が必要となる場合があります。
- キャッシュ カウンタを持つ一部のメトリックについて、拡張収集ビルダ / マネージャで欠落値が報告される場合があります。メトリック値がオーバーフローした際に、一部の環境でこの問題が発生するおそれがあります。また、メッセージが ECBM ステータス ファイルに送信されます。収集を再開するとこの問題を解決できます。

拡張収集ビルダ / マネージャの概念、データ収集の作成および表示の手順の説明は、拡張収集ビルダ / マネージャのオンライン ヘルプから参照できます。オンライン ヘルプを表示するには、デスクトップから、[**スタート**] → [**すべてのプログラム**] → [**HP**] → [**Operations Agent**] → [**Performance Collection Component**] → [**ECB-ECM Online Help**] を選択します。[Performance Collection Component] メイン ウィンドウの [Agent] メニューから [**Extended Collections**] を選択し、[Extended Collection Builder and Manager] ウィンドウの [Help] メニューから [**Help Topics**] を選択することもできます。Extended Collection Builder and Manager に表示されるダイアログ ボックスの [**Help**] ボタンを選択すれば、オンライン ヘルプを利用できます。

コマンド ラインからの ECBM の管理

Windows のコマンド プロンプトを使用すれば、<rpmtools>\bin ディレクトリから ECBM プログラムを実行できます。

コマンド ラインから収集を管理するには、以下のコマンドを使用します。

```
\rpmtools\bin\mwcmcmd.exe
```

各種オプションを表示するには、次のコマンドを入力します。

```
\rpmtools\bin\mwcmcmd /?
```

停止した収集を開始するには、次のコマンドを入力します。

```
mwcmcmd start <収集名>
```

可変インスタンス ポリシーから収集を開始するには、次のコマンドを入力します。

```
mwcmcmd start <ポリシー名> <収集名> <インスタンス> [ オプション ]
```

以下のオプションを使用できます。

```
-i <sampling_interval>  - サンプリングの間隔を変更します ( 秒 )
-l <logfile_path_name>  - デフォルトの記録場所を変更します
-a <alarm_file>         - アラーム定義ファイルを変更します
```

有効な収集を停止するには、次のコマンドを入力します。

```
mwcmcmd stop <収集名>
```

以下はポリシー ファイルを登録するためのコマンドです。

```
mwcmcmd register <ポリシー ファイル> <collection/ポリシー名> [オプション]
```

固定インスタンス ポリシーの登録では、以下のオプションのみが使用できます。

```
-i <sampling_interval>    - サンプリングの間隔を変更します(秒)
-l <logfile_path_name>    - デフォルトの記録場所を変更します
-a <alarm_file>           - アラーム定義ファイルを変更します
```

単一の収集を削除する手順

```
mwcmcmd delete <収集 / ポリシー名> [オプション]
```

収集の削除には、以下のオプションのみが使用できます。

```
-p <archive_path>    - 特定のパスにログファイルをアーカイブします
-r                   - Performance エージェントを再起動します
```

複数の収集またはポリシーを削除する手順

```
mwcmcmd delete { <収集 / ポリシー名> | -c | -all }
    -c       - すべての収集を削除します
    -a       - すべての収集およびポリシーを削除します
```

複数のポリシー / 収集を同時に削除する場合は、Performance Collection Component が自動に再起動し、割り当てられているすべてのログ ファイルが削除されます。

登録済みのすべての収集をリスト表示するには、次のコマンドを入力します。

```
mwcmcmd list
```

収集またはポリシーのプロパティをリスト表示するには、次のコマンドを入力します。

```
mwcmcmd properties <収集 / ポリシー名>
```

可変インスタンス オブジェクトをリスト表示するには、次のように入力します。

```
mwcmcmd objects <ポリシー名>
```

12 データ ソース統合の概要

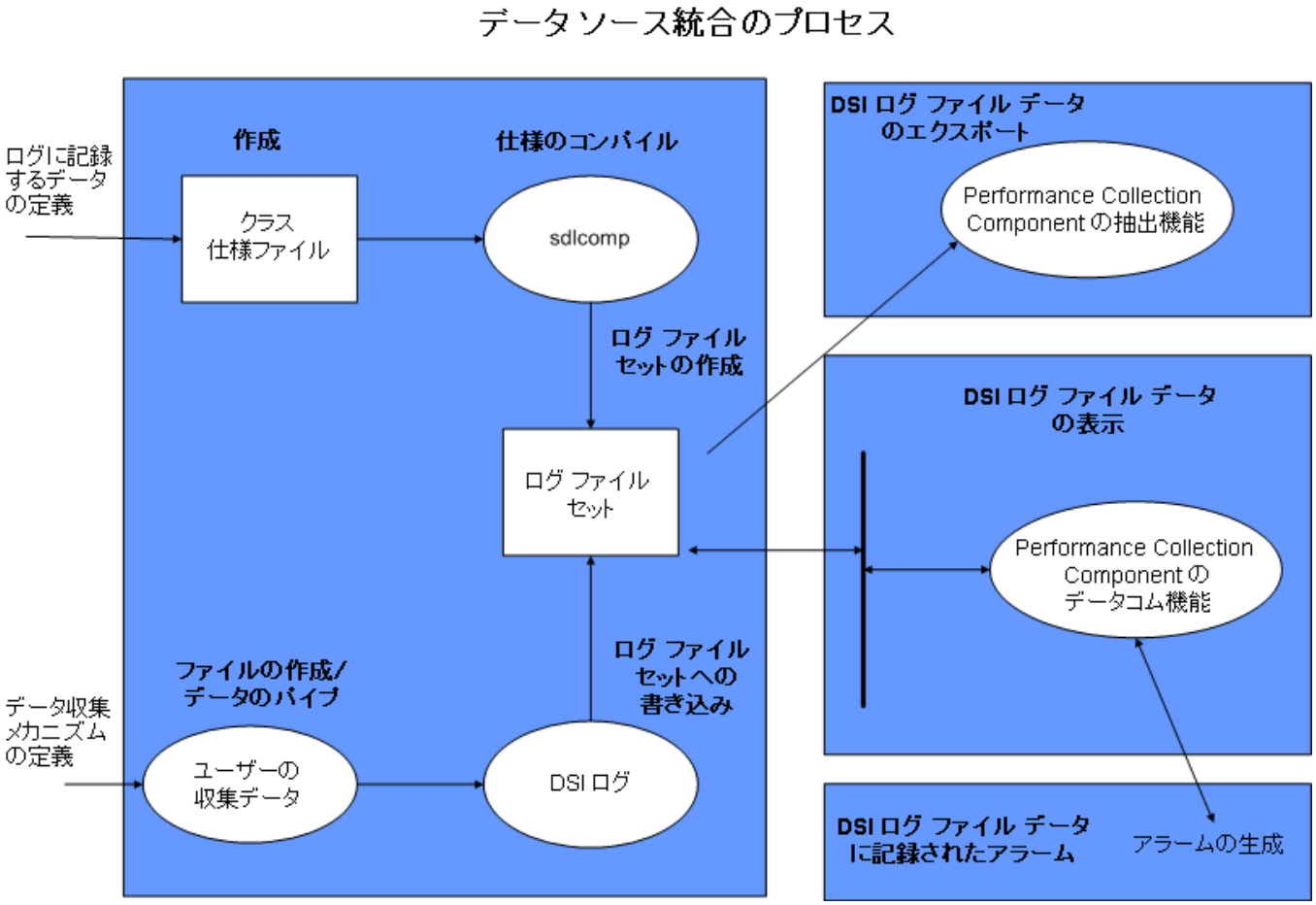
データ ソース統合 (DSI) 技術により、HP Operations エージェント はデータの記録、アラームの定義、Performance Collection Component の scope コレクタが記録するメトリックのみでなく、新しいデータ ソースのメトリックへのアクセスが可能になります。各メトリックは、データベース、LAN モニタ、およびエンドユーザーのアプリケーションなどのデータ ソースから獲得します。

DSI を使用して記録するデータは、scope コレクタが記録する標準のパフォーマンス メトリックと共に HP Performance Manager で表示されます。また、DSI が記録したデータは Performance Collection Component の extract プログラムを使用して、表計算ソフトや類似の分析ソフトウェアで表示するためにエクスポートされます。

DSI の動作

次の図は、DSI ログ ファイルを作成する方法と DSI ログ ファイルを使用してデータを記録し管理する方法を示します。DSI ログ ファイルには、Performance Collection Component の scope コレクタ以外で収集される自己記述データが含まれます。DSI のプロセスの詳細は、次のページで説明します。

図 18　　データ ソース統合のプロセス



DSI を使用してデータを記録するには、次の作業を行います

クラス仕様の作成

はじめに、記録するデータの各クラスの仕様を作成しコンパイルします。仕様では、データ クラスとクラス内で記録する各メトリックについて記述します。DSI コンパイラ、sdlcomp を使用して仕様をコンパイルすると空のログ ファイル セットが作成され、dsilog プログラムからデータを受信します。このプロセスにより、ルート ファイル、記述ファイル、および 1 つ以上のデータ ファイルを含むログ ファイル セットが作成されます。

データの収集と記録

対象となるプロセスを起動して、記録するデータを収集します。収集プロセスの出力は、直接 dsilog プログラムに渡される場合と、データを保存するファイルを介して渡される場合があります。dsilog はデータを仕様に従って処理し、適切なログ ファイルに書き込みます。dsilog プログラムでは、受信データの形式とフォームを指定できます。

DSI プロセスに送信するデータは、複数のデータ レコードを含む必要があります。各レコードは 1 行に含まれるメトリック値で構成されます。DSI に一度に 1 レコードずつデータを送信しプロセスを終了した後、別のレコードを送信する場合、dsilog は付加できますがデータの要約はできません。

データの使用

Performance Manager を使用して、DSI ログ ファイル データを表示できます。または、Performance Collection Component の `extract` プログラムを使用して、他の分析ツールで使用するためにデータをエクスポートできます。DSI メトリックが確定条件を超えたときにアラームを発生するようにも構成できます。

13 データ ソース統合の使用方法

この章では、DSI の使用方法の概要について、次の事項を説明します。

- データ収集の計画
- クラスの仕様ファイルにおけるログ ファイル フォーマットの定義
- 空のログ ファイル セットの作成
- ログ ファイル セットへのデータの記録
- ログ データの使用

DSI クラスの仕様と DSI プログラムについての詳細は、第 14 章の「DSI クラスの仕様」と第 15 章の「DSI プログラムに関する参考資料」を参照してください。

データ収集の計画

DSI クラスの仕様ファイルを作成しログ処理を開始する前に、次の事項について確認する必要があります。

- 動作環境。コンピューティング リソースを管理する場合に有益となるデータの種類を認識するために必要です。
- 使用できるデータ
- データの保存場所
- データの収集方法
- データ項目間の区切り記号。dsilog が適切に処理を実行するには、入力ストリーム内のメトリック値を空白 (デフォルト)、またはユーザー定義の区切り記号で区切る必要があります。
- 収集の頻度
- ログの管理に必要な領域
- データにアクセスする場合に使用するプログラムの出力、またはプロセスの出力
- 生成するアラームと必要な条件
- クラスの仕様と dsilog プロセスで記録する場合に使用できるオプション

ログ ファイル フォーマットの定義

収集するデータの種類が明確になると、クラスの仕様を作成して記録するデータを定義し、記録されたデータを含むログ ファイル セットを定義します。クラスの仕様ファイルには次の情報を入力します。

- データ クラス名と ID 番号
- クラス名の代わりとなるラベル名 (オプション)。ラベル名がある場合は Performance Manager などで使用されます。
- 古いデータをロールアウトして新しいデータの領域を確保する場合の必要事項。詳細は、「ログ ファイルの編成方法」を参照してください。
- メトリック名と他の記述情報。メトリック値を考慮した 10 進数の値などが含まれます。
- 1 時間に記録されるレコード数を制限するデータ要約方法

クラスの仕様の例を次に示します。

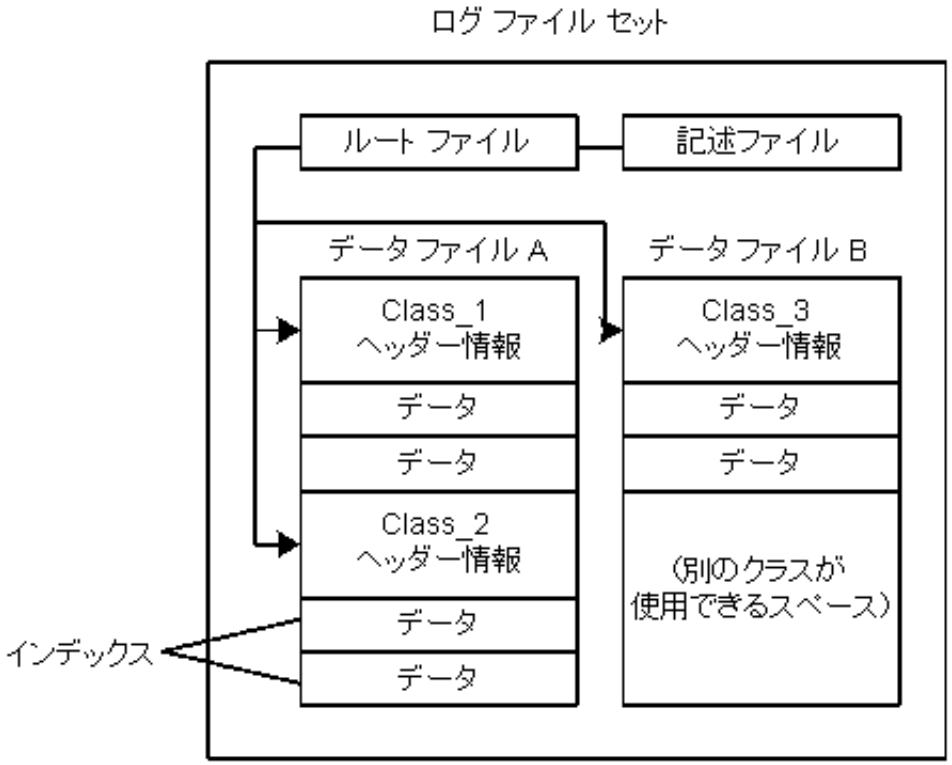
```
CLASS VMSTAT_STATS = 10001
LABEL "VMSTAT data"
INDEX BY HOUR
MAX INDEXES 12
ROLL BY HOUR
RECORDS PER HOUR 120;

METRICS
RUN_Q_PROCS    = 106
LABEL   "Procs in run q"
PRECISION 0;

BLOCKED_PROCS   = 107
LABEL   "Blocked Processes"
PRECISION 0;
```

クラスの仕様ファイルには 1 つ以上のクラスが含まれます。クラスの仕様ファイルの作成が完了すると、ファイルに名前を付けて保存します。DSI コンパイラ sdlcomp を実行する場合は、このファイルを使用してログ ファイル セットを作成します。クラスの仕様とメトリック記述の構文についての詳細は、第 14 章の「DSI クラスの仕様」を参照してください。

ログ ファイルの編成方法

ログ ファイルはクラスに編成されます。各クラスは、受信データ ソースの 1 つを表現していて、共に記録されるデータ項目やメトリックのグループで構成されます。クラス内にあるデータの各レコードまたは各ラインは、メトリック グループの値のサンプルを表現します。

各クラスのデータは、ログ ファイル セットの一部であるログ ファイル内のディスクに保存されます。ログ ファイル セットは、ルート ファイル、記述ファイル、1 つ以上のログ ファイルで構成されます。クラス内のデータはすべて常に単一のデータ ファイルに保持されます。ただし、ログ ファイル セット名を sdlcomp コンパイラに使用する場合は、単一のログ ファイル セットや個別のログ ファイル セットに複数のクラスを保存できます。次の図は、2 つのクラスを単一のログ ファイル セットに保存する方法を示します。



各クラスは循環ログ ファイルとして扱われるため、複数のクラスを単一のログ ファイル セットに保存するように指定した場合でも、各クラスの記憶容量を個別に設定できます。記憶容量に達するとクラスは「ロール」されます。「ロールする」とは、新しいデータの領域を確保するためにクラス内で最も古いレコードを削除することです。

アーカイブ ファイルに古いデータをエクスポートするなど、クラスがロールされたときに実行するアクションを指定できます。

ログ ファイル セットの作成

DSI コンパイラ sdlcomp は、クラスの仕様ファイルを使用して空のログ ファイル セットを作成または更新します。ログ ファイル セットは、dsilog プログラムから記録済みのデータを受信する場合に使用されます。

ログ ファイル セットを作成するには、次の作業を実行します。

1 適切な変数とオプションを使用して、sdlcomp を実行します。次に例を示します。

sdlcomp [-maxclass *value*] *specification_file*
[*logfile_set*[*log file*]] [options]

2 出力エラーを検査し、必要な場合は変更を行います。

sdlcomp についての詳細は、第 15 章の「コンパイラ構文」を参照してください。

クラスの仕様ファイルとログ処理のテスト (オプション)

DSI は sdlgendata プログラムを使用して、受信する生成済みのデータ ソースに対するクラスの仕様ファイルをテストします。このプロセスの出力を調べて、DSI がデータを仕様どおりに記録していることを確認できます。sdlgendata については、第 15 章の「sdlgendata によるログ処理のテスト」を参照してください。

次の手順で、ログ処理用のクラスの仕様ファイルをテストします。

1 sdlgendata が生成したデータを dsilog プログラムに送信します。構文は次のようになります。

sdlgendata *logfile_set class* **|** **dsilog** *logfile_set class* **-vo**

2 出力を検査して、クラスの仕様ファイルがデータ収集プロセスのフォーマットと一致していることを確認します。sdlgendata プログラムの出力がユーザーのプログラムの出力と異なる場合は、出力フォーマットかクラスの指定ファイルのいずれかにエラーがあります。

3 実際のデータを収集する前に、テスト プロセスで生成されたログ ファイルをすべて削除してください。

ログ ファイル セットへのデータの記録

ログ ファイル セットを作成し任意でファイルをテストした後、必要な場合は Performance Collection Component 構成ファイルを更新し、dsilog プログラムを実行して受信データを記録します。

1 データ ソース構成ファイル datasources を更新し、アラームを生成するデータ ソースとして DSI ログ ファイルを追加します。

2 特定の DSI メトリックにアラームを発生させる場合は、アラーム定義ファイル alarmdef を変更します。詳細情報は、第 15 章の「DSI メトリックのアラーム定義」を参照してください。

3 -vi オプション セットと共に dsilog プログラムにデータを渡して、ログ処理をテストします (このデータはクラスの仕様と一致する sdlgendata により生成されます)。このテストはオプションです。

4 データを検査して、正しく記録されたことを確認します。

5 テストが終了すると、テストに使用されたデータを削除します。

6 コマンド ラインから収集プロセスを起動します。

7 収集プロセスからのデータを適切な変数やオプション セットと共に dsilog (またはその他の方法で stdin) に渡します。次に例を示します。

< *プログラムまたはプロセスと変数*>| dsilog *logfile_set class*

dsilog プログラムは、データの連続的な流れを受け取るように考慮されています。したがって、dsilog が連続的な入力データを受け取るように、スクリプトを組み立てることが重要です。新しい入力データ ポイントに対して、新しい dsilog プロセスを作成するようなスクリプトは記述しないでください。そうした場合、複製のタイムスタンプが、dsilog ファイルに書かれ、ファイルを読む際に Performance Manager と perfalarm に不具合発生の原因となります。推奨されるスクリプトの例に関しては、第 16 章の「データ ソースの統合例」を参照してください。

dsilog オプションについての詳細は、第 15 章の「dsilog によるログ処理」を参照してください。

ログ データの使用

DSI ログ ファイルを作成した後、Performance Collection Component の `extract` プログラムを使用してデータをエクスポートできます。DSI メトリックが確定条件を超えたときにアラームを発生するようにも構成できます。

DSI ログ データを使用するには、次の方法を実行します。

- 表計算ソフトなどのツールのレポートに使用するデータをエクスポートします。
- Performance Manager などの分析ツールを使用して、エクスポートされた DSI データを表示します。
- HP Operations Manager または HP Network Node Manager を使用してアラームを監視します。

DSI ログ ファイルから抽出されたログ ファイルは作成できません。

14 DSI クラスの仕様

この章では、次の参照情報についての詳細を説明します。

- クラスの仕様
- クラスの仕様構文
- クラスの仕様におけるメトリックの記述

クラスの仕様

受信データの各ソースについて、受信データを保存するためのフォーマットを記述するクラスの仕様ファイルを作成する必要があります。ファイルを作成するには、次の項「クラスの仕様構文」で説明するクラスの仕様言語を使用します。クラスの仕様ファイルには次の記述が含まれます。

- クラスの記述。受信データ セットに名前と数値 ID を割り当て、保存するデータの量を決定し、新しいデータのための領域を確保するためにデータをロールするときを指定します。
- 各データ項目に関するメトリックの記述。メトリックの記述はデータ項目に名前を付けてそれを記述します。クラスに対して構成されている時間に複数のレコードが届く場合、データに適用する要約レベル (RECORDS PER HOUR) も指定します。

クラスの仕様ファイルを生成するには、ファイルを ASCII テキスト ファイルとして保存できるエディタまたはワード プロセッサを使用します。sdlcomp を実行してクラスの仕様ファイルをコンパイルする場合、対象となるファイル名を指定します。クラスの仕様をコンパイルすると、データの記憶領域としてログ ファイル セットが自動的に作成または更新されます。

クラスの仕様では、クラスに対して 1 時間あたりの保存レコード数よりも多くのレコードが届いた場合の要約方法を指定できます。たとえば、1 時間に 12 レコード (5 分間で 1 レコード) を保存するように指定したときにレコードが 1 分おきに届く場合、データ項目の一部の平均を取り、その他の項目を合計して実行件数を維持できます。

DSI コンパイラ sdlcomp はログ ファイル セット (名前は logfile_set_name) に対して次の名前のファイルを作成します。

```
logfile_set_name および logfile_set_name.desc
```

sldcomp はクラス (名前は class_name) に対して次のデフォルト名のファイルを作成します。

```
logfile_set_name.class_name
```

上記の命名規則に矛盾する名前をクラスの仕様ファイルに使用しないように注意してください。sdlcomp が正しく機能しなくなる場合があります。

クラスの仕様構文

[ ] (大かっこ) 内に示されている構文は、オプションです。{ } (中かっこ) 内に示されている複数の文は、それらの文のうちの 1 つを選択する必要があることを示します。斜体の語句は入力する変数名または数字を示します。クラスの仕様の最後や各メトリックの仕様の最後に付けるセミコロンの直前を除き、任意の場所でカンマを使用して構文をわかりやすく区切ることができます。文では、大文字と小文字を区別しません。

metric_label_name や *class_label_name* などのユーザー定義の記述は、DSI クラス仕様の文法のキーワード要素とは必ず違うものを設定してください。

コメントは、# または // から始まります。# または // の後に続く語句は、その行の終わりまですべて無視されます。クラスの記述の後や各メトリックの記述の後にはセミコロンが必要であることに注意してください。次にクラスの仕様の各部分についての詳細と例を示します。

```
CLASS class_name = class_id_number
[LABEL "class_label_name"]


 [INDEX BY {HOUR | DAY | MONTH} MAX INDEXES number
 [[ROLL BY {HOUR | DAY | MONTH} [ACTION "action" ]

 [ CAPACITY {maximum_record_number} ]
[ RECORDS PER HOUR  number ]
;

METRICS

metric_name = metric_id_number
[ LABEL "metric_label_name" ]
[ TOTALED | AVERAGED | SUMMARIZED BY metric_name ]
 [ MAXIMUM  metric_maximum_number ]
[PRECISION {0 | 1 | 2 | 3 | 4 | 5} ]
[TYPE TEXT LENGTH "length"]
;
```

クラスの記述

クラスの記述を作成するには、特定のデータ ソースからのメトリックのグループに名前を付け、クラスの容量を指定し、容量に達したときにクラス内のデータをロールする方法を指定します。

クラスの記述は必ず CLASS キーワードで開始します。クラスの仕様の最終パラメータの後には必ずセミコロンが付きます。

構文

```
CLASS class_name = class_id_number

[LABEL "class_label_name"]


[INDEX BY { HOUR | DAY | MONTH } MAX INDEXES number
[[ ROLL BY { HOUR | DAY | MONTH } [ACTION "action"]


[ CAPACITY {maximum_record_number} ]

[ RECORDS PER HOUR number]

;
```

デフォルトの設定

クラスの記述のデフォルトの設定は次のとおりです。

```
LABEL (class_name)
INDEX BY DAY
MAX INDEXES 9
RECORDS PER HOUR 12
```

デフォルトを使用するには、キーワード CLASS を *class_name* および数値 *class_id_number* と共に入力します。

クラス

クラス名とクラス ID は、特定のデータ ソースからのメトリックのグループを識別します。

構文

```
CLASS    class_name = class_id_number
```

使い方

class_name と *class_ID_number* は次の要件を満たす必要があります。

- *class_name* は英数字で、20 文字以下にします。名前は必ず英字で始まり、アンダースコアを使用できます (ただし、特殊文字は使用できません)。
- *class_ID_number* は 6 桁以内の数値です。
- *class_name* と *class_ID_number* はいずれも大文字と小文字の区別がありません。

- *class_name* と *class_ID_number* は定義されるクラスすべてで一意であり、Performance Collection Component の parm ファイル内で定義されているいずれのアプリケーションとも異なる必要があります。parm ファイルについての詳細は、『HP Operations エージェント for UNIX ユーザー マニュアル』の第 2 章を参照してください。

例

```
CLASS VMSTAT_STATS = 10001;
```

ラベル

クラス ラベルはクラス全体を識別し、Performance Manager ではクラス名の代わりに使用されます。

構文

[LABEL "*class_label_name***"]**

使い方

class_label_name は次の要件を満たす必要があります。

- 二重引用符で囲む必要があります。
- 48 文字以下にします。
- CAPACITY や ACTION など、DSI クラスの仕様構文のいずれのキーワード要素とも異なる必要があります。
- ラベルに二重引用符が含まれる場合、二重引用符の前に円記号 (**¥**) を付けます。たとえば、ラベルが **"my" data** である場合、**¥**"my**¥**" data と入力します。
- ラベルが指定されていない場合、*class_name* がデフォルトとして使用されます。

例

```
CLASS VMSTAT_STATS = 10001
LABEL "VMSTAT data";
```

INDEX BY, MAX INDEXES, ROLL BY

INDEX BY、MAX INDEXES、ROLL BY の各設定により、データを保存する方法や削除するときを指定できます。これらの設定により、保存するデータのブロック、保存するブロックの最大数、データが最大インデックス値に達した場合に廃棄するデータ ブロックのサイズを決定します。

構文

[NDEX BY {HOUR | DAY | MONTH} MAX INDEXES *number***]**
[[ROLL BY {HOUR | DAY | MONTH} [ACTION "*action***"]]**

使い方

INDEX BY の設定により、クラスの容量に達したときにデータ ブロックをロールアウトできます。INDEX BY と RECORDS PER HOUR の各オプションを使用して、後述の「ログ ファイル サイズの制御」で説明されているように、クラスの容量を間接的に設定できます。

INDEX BY の設定は ROLL BY の設定を超えられません。たとえば、INDEX BY DAY は ROLL BY HOUR と共に使用できませんが、INDEX BY HOUR は ROLL BY DAY と共に使用できます。

ROLL BY が指定されていない場合、INDEX BY 設定が使用されます。容量に達した場合、最も古いロール インターバルで記録されたすべてのレコードが解放され、領域が再使用されます。

ACTION を指定した場合、データが廃棄 (ロール) される前に、その ACTION が実行されます。このオプションの ACTION を使用して、クラスからデータを削除する前に別の場所にエクスポートできます。データのエクスポートについての詳細は、第 15 章の「DSI プログラムに関する参考資料」を参照してください。

ロール アクションに関する注記

ACTION 文で指定された UNIX コマンドはバックグラウンドでは実行できません。ACTION 文でのコマンドの指定は長い遅延の原因となり、その間、新しいデータが記録されないため避けてください。

コマンドの長さが 2 行以上になる場合は、各ラインの始めと終わりに二重引用符を付けます。複数のラインを結合したときにコマンド ラインの各オプションがつながらないように、引用符の内側には必要に応じてスペースを入れます。

コマンドに二重引用符が含まれる場合は、その直前に円記号 (¥) を付けます。

ACTION は 199 文字以下に制限されます。

ACTION 文でマクロを使用して、ログ ファイルからロール アウトされるデータのタイム ウィンドウを定義できます。これらのマクロは、dsilog により展開されます。PT_START を使用してロールアウトされるデータのブロックの先頭を UNIX 時間 (1/1/70 00:00:00 からの秒数) で指定することも、PT_END を使用してデータの終端を UNIX 時間で指定することも可能です。これらは、データを上書きする前にデータをエクスポートする extract プログラムと組み合わせて使用すると特に便利です。

マクロが使用される場合は、最大 199 文字まで拡張できます。

例

次の例は、INDEX BY、MAX INDEXES、および ROLL BY の各文節間の関係が明確になることを示します。

次の例は、CAPACITY を 144 レコード (1×12×12) に間接的に設定しています。

```
CLASS VMSTAT_STATS = 10001
LABEL "VMSTAT data"
INDEX BY HOUR
MAX INDEXES 12
RECORDS PER HOUR 12;
```

次の例は、CAPACITY を 1440 レコード (1×12×120) に間接的に設定しています。

```
CLASS VMSTAT_STATS = 10001
LABEL "VMSTAT data"
INDEX BY HOUR
MAX INDEXES  12
RECORDS PER HOUR  120;
```

次の例では、ROLL BY HOUR を示します。

```
CLASS VMSTAT_STATS = 10001
LABEL  "VMSTAT data"
INDEX BY HOUR
MAX INDEXES  12
ROLL BY HOUR
RECORDS PER HOUR  120;
```

次の例は、現在 (週末を除き) ロールの対象として識別されているすべてのデータを、上書きする前に sys.sdl と呼ばれるファイルにエクスポートしています。次に示した例の最後の数行は二重引用符で囲まれており、単一のコマンドを示していることに注意してください。

```
CLASS VMSTAT_STATS = 10001
LABEL  "VMSTAT data"
INDEX BY HOUR
MAX INDEXES  12
ROLL BY HOUR
ACTION "extract -xp -l sdl_new -C SYS_STATS "
"-B $PT_START$ -E $PT_END$ -f sys.sdl, purge -we 17 "
RECORDS PER HOUR  120;
```

その他の例

インデックスの設定例を次に示します。保存するデータの量を決定するときに参考にしてください。

INDEX BY	MAX INDEXES	保存データ量
HOUR	72	3 日分
HOUR	168	7 日分
HOUR	744	31 日分
DAY	365	1 年分
MONTH	12	1 年分

次の表では ROLL BY による設定の詳細を説明します。

INDEX BY	MAX INDEXES	ROLL BY	説明
DAY	9	DAY	9 日分のデータがログ ファイルに保存されます。10 日目が記録される前に、1 日目がロール アウトされます。これらは、インデックスおよび最大インデックス数のデフォルト値です。
HOUR	72	HOUR	72 時間 (3 日) 分のデータがログ ファイルに保存されます。73 時間目が記録される前に、1 時間目がロール アウトされます。その後、各時間の始まりに、「最も古い」時間がロール アウトされます。
HOUR	168	DAY	168 時間 (7 日) 分のデータがログ ファイルに保存されます。169 時間 (8 日) 目が記録される前に、1 日目がロール アウトされます。その後、各日の始まりに、「最も古い」日がロール アウトされます。
HOUR	744	MONTH	744 時間 (31 日) 分のデータがログ ファイルに保存されます。745 時間 (32 日) 目が記録される前に、1 か月目がロール アウトされます。その後、745 時間目が記録される前に、「最も古い」月がロール アウトされます。 たとえば、dsilog を 4 月 15 日に開始し、5 月 16 日まで (744 時間) のデータを記録する場合、745 時間目 (5 月 17 日の最初の 1 時間) のデータを記録する前に、dsilog は 4 月分 (4 月 15 日から 30 日) のデータをロール アウトします。

INDEX BY	MAX INDEXES	ROLL BY	説明
DAY	30	DAY	30 日分のデータがログ ファイルに保存されます。31 日目が記録される前に、1 日目がロール アウトされます。その後、各日の始まりに、「最も古い」月がロール アウトされます。 たとえば、dsilog を 4 月 1 日に開始し、4 月中のデータを記録する場合、5 月 1 日 (31 日目) のデータが記録される前に 4 月 1 日のデータがロール アウトされます。
DAY	62	MONTH	62 日分のデータがログ ファイルに保存されます。63 日目が記録される前に、1 か月目がロール アウトされます。その後、63 日目が記録される前に、「最も古い」月がロール アウトされます。 たとえば、dsilog を 3 月 1 日に開始し、3 月と 4 月のデータを記録する場合、ログ ファイルには 61 日分のデータが保存されます。dsilog が 5 月 1 日 (62 日目) のデータを記録すると、ログ ファイルはいっぱいになります。dsilog は、5 月 2 日のデータを記録する前に 3 月全体をロール アウトします。
MONTH	2	MONTH	2 か月分のデータがログ ファイルに保存されます。3 か月目が記録される前に、1 か月目がロール アウトされます。その後、各月の始まりに、「最も古い」月がロール アウトされます。 たとえば、dsilog を 1 月 1 日に開始し、1 月と 2 月のデータを記録する場合、dsilog は 3 月のデータを記録する前に 1 月全体をロール アウトします。

ログ ファイル サイズの制御

各クラスに保存するデータ量と新しいデータの領域を確保するために廃棄するデータ量を決定します。

クラスの容量は、INDEX BY (時間、日、月)、RECORDS PER HOUR、MAX INDEXES から計算されます。各種設定の結果を次の例に示します。

この例では、クラスの容量は 288 (24 インデックス ×1 時間あたり 12 レコード) になります。

```
INDEX BY HOUR
MAX INDEXES 24
RECORDS PER HOUR 12
```

この例では、クラスの容量は 504 (7 日 ×1 日あたり 24 時間 ×1 時間あたり 3 レコード) になります。

```
INDEX BY DAY
MAX INDEXES 7
RECORDS PER HOUR 3
```

この例では、クラスの容量は 14,880 (2 か月 ×1 か月あたり 31 日 ×1 日あたり 24 時間 ×1 時間あたり 10 レコード) になります。

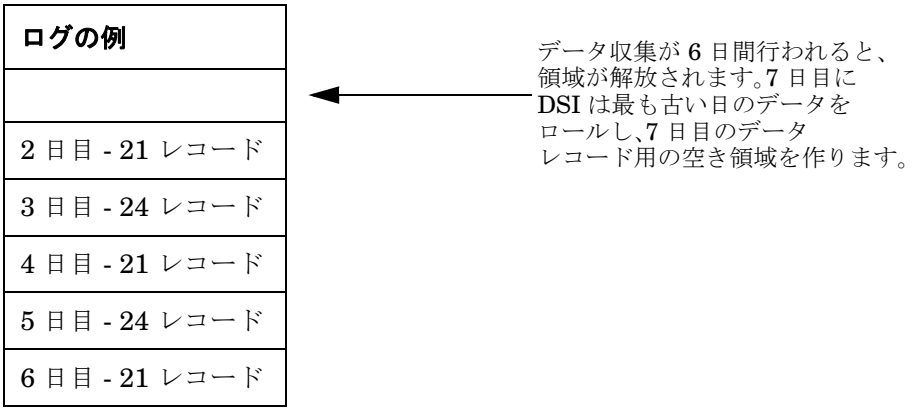
```
INDEX BY MONTH
MAX INDEXES 2
RECORDS PER HOUR 10
```

INDEX BY、RECORDS PER HOUR, および MAX INDEXES の値が指定されていない場合、DSI はクラスの記述にデフォルトを使用します。この章の最初に記載された「クラスの記述」の項にある「デフォルトの設定」を参照してください。

ROLL BY オプションを使用すると、クラスのレコード容量に達するたびに廃棄するデータの量を指定できます。ROLL BY に関する設定は、INDEX BY の設定に依存します。ROLL BY の単位 (時間、日、月) は必ず INDEX BY の単位よりも大きく設定されます。

次の例では、サンプル値をロール アウトする方法が示されます。

```
INDEX BY DAY
MAX INDEXES 6
ROLL BY DAY
```



上記の例では、クラスの容量が次の設定により 6 日分のデータに制限されています。

```
MAX INDEXES 6
```

データの削除は、次の設定により 1 日分に設定されています。

```
ROLL BY DAY
```

7 日目のデータが届くと、最も古い日のデータが廃棄されます。ログ処理の開始時には、廃棄されるデータがないことに注意してください。クラスが初めていっぱいになる 7 日目の終わりに、ロールが 1 日 1 回行われます。

RECORDS PER HOUR

RECORDS PER HOUR の設定は、1 時間あたりのログ ファイルへの書き込みレコード数を決定します。RECORDS PER HOUR のデフォルト値は、Performance Collection Component のサンプル データの測定インターバルである 5 分 (12 レコードあたり 60 分 = 5 分ごとに記録) に合わせて 12 になっています。

デフォルト値または入力する値によっては、データがログ ファイルの一部になる前にログ処理により要約される必要があります。各データ項目の要約に使用する方法はメトリック記述で指定します。詳細は、この章で後述している「要約方法」の項を参照してください。

構文

[RECORDS PER HOUR *number*]

使い方

ログ処理では、この値を使用して受信データを要約し、指定された数のレコードを生成します。たとえば、データが 1 分おきに到着し RECORDS PER HOUR を 6 (10 分ごと) に設定している場合、10 個のデータ ポイントが要約されて各レコードがクラスに書き込まれます。一般的な RECORDS PER HOUR の設定例のいくつかを次に示します。

```
RECORDS PER HOUR 6 --> 10 分あたり 1 レコード
RECORDS PER HOUR 12 --> 5 分あたり 1 レコード
RECORDS PER HOUR 60 --> 1 分あたり 1 レコード
RECORDS PER HOUR 120 --> 30 秒あたり 1 レコード
```

注記

RECORDS PER HOUR は、dsilog の -s seconds オプションにより無効にできます。ただし、元の設定を無効にすると Performance Manager によるデータのグラフ作成で問題が発生する場合があります。

dsilog が全記録期間中にメトリック データを受信しない場合、そのメトリックのデータがないことを示すインジケータが記録されます。dsilog の -asyn オプションを使用して、DSI に対して最後に記録された値を使用するように指定できます。-asyn オプションの詳細は、第 15 章の「dsilog によるログ処理」を参照してください。

例

この例では、レコードは 10 分おきに書き込まれます。

```
CLASS VMSTAT_STATS = 10001
LABEL  "VMSTAT data"
RECORDS PER HOUR  6;
```

CAPACITY

CAPACITY は、クラスに保存されるレコード数です。

構文

[CAPACITY {*maximum_record_number***}]**

使い方

クラスの容量は、RECORDS PER HOUR、INDEX BY および MAX INDEXES 内の設定から引き出されます。これらの他の設定から引き出される値より大きな容量を指定する場合を除き、CAPACITY の設定は無視されます。大きな容量を指定する場合、指定する容量を確保するために MAX INDEXES の設定が増やされます。

例

```
INDEX BY DAY
MAX INDEXES 9
RECORDS PER HOUR 12
CAPACITY 3000
```

上記の例では、クラスの容量として 2,592 レコード (9 日 ×1 日あたり 24 時間 ×1 時間あたり 12 レコード) が引き出されます。

3000 は 2592 よりも大きいため、sdlcomp は MAX INDEXES を 11 に増やします。その結果、クラス容量は 3168 になります。コンパイル後は、-decomp オプションと共に sdlutil を実行することで、最終的な MAX INDEXES および CAPACITY の値を確認できます。

メトリックの記述

クラスの仕様ファイルのメトリックの記述を使用して、クラスの個々のデータ項目を定義します。メトリックの記述によりメトリック名と数値の ID が対応付けられ、1 時間あたりのレコード数が RECORDS PER HOUR の設定で指定された値を超えた場合のデータ要約方法が指定されます。

metric_label_name などのユーザー定義の記述は、DSI クラス仕様の文法のキーワード要素とは必ず違うものを設定してください。

dsilog フォーマット ファイルでは、メトリックの数は 100 以下に制限されることに注意してください。

```
METRICS

metric_name = metric_id_number
[LABEL "metric_label_name"]
[TOTALED | AVERAGED | SUMMARIZED BY metric_name]
[MAXIMUM metric_maximum_number]
[PRECISION { 0 | 1 | 2 | 3 | 4 | 5 }]
TYPE TEXT LENGTH "length"
```

数値メトリックの場合、要約方法 (TOTALED、AVERAGED、SUMMARIZED BY) 、および PRECISION を指定できます。テキスト メトリックの場合、TYPE TEXT LENGTH のみを指定できます。

METRICS

メトリック名と ID 番号は、収集されるメトリックを識別します。

構文

METRICS
metric_name **=** *metric_id_number*

使い方

メトリック セクションは、最初のメトリックを定義する前に METRICS キーワードで開始する必要があります。各メトリックには、次の要件を満たす名前が必要です。

- 20 文字以下にします。
- 英字で始めます。
- 英数字とアンダースコアでのみ構成します。
- 大文字と小文字の区別はありません。

また、メトリックには 6 桁以下のメトリック ID があります。

metric_name および *metric_id_number* はそれぞれクラス内で定義したすべてのメトリックの間で一意である必要があります。*class_name*:*metric_name* の組み合わせはこのシステムにおいて一意であり、いずれの *application_name*:*metric_name* とも異なる必要があります。

各メトリックの記述の終わりにはセミコロン (;) を付けます。

定義が同一である場合、同じログ ファイル セット内でデータが保存されている別のクラスのメトリック名を再使用できます (第 13 章の「ログ ファイルの編成方法」を参照してください)。同じログ ファイル セット内の別のクラスですでに定義されているメトリックの定義を再使用するには *metric_name* のみを指定し、*metric_id_number* やその他の仕様を指定する必要はありません。いずれかのオプションを前回定義したメトリックと異なるように設定する場合、そのメトリックに一意の名前と数値 ID を割り当て、再定義する必要があります。

クラスの仕様のセクションにおけるメトリック名の順序は、ログ データをエクスポートするときのフィールドの順序を決定します。受信データの順序がこの仕様でリストされる順序と異なる場合や、受信データ ストリーム内のすべてのデータを記録する必要がない場合、第 15 章の「DSI プログラムに関する参考資料」に記載されている適正な位置にメトリックをマップする方法を参照してください。

タイムスタンプ メトリックは、各クラスの最初のメトリックとして自動的に挿入されます。エクスポートされたデータ内でタイムスタンプを別の位置に表示する場合、内部的に確定しているメトリック定義の短い形式 (DATE_TIME;) をタイムスタンプの表示位置に含めます。タイムスタンプを除き、受信データの一部である UNIX タイムスタンプ (1/1/70 00:00:00 からの秒数) を使用するには、dsilog プロセスを開始するときに -timestamp オプションを選択します。

メトリック名をラベルとして使用し、AVERAGED、MAXIMUM 100、および PRECISION 3 の小数点位置のデフォルトを使用する最も単純なメトリックの記述では、次の記述が必要です。

```
METRICS
metric_name = metric_id_number
```

メトリック名を再使用しているかどうかにかかわらず、sdlcomp を使用して各クラスをコンパイルし dsilog プロセスを使用して、そのクラスのデータの記録を開始する必要があります。

例

```
VM;
```

VM は、同じログ ファイル セット内の別のクラスですでに定義されているメトリックの定義を再使用しているメトリックの例です。

ラベル

メトリック ラベルは、Performance Manager グラフとエクスポート データのメトリックを識別します。

構文

[LABEL "*metric_label_name***"]**

使い方

グラフとエクスポート データのメトリックのラベルとなるテキスト文字列を、二重引用符で囲んで指定します。48 文字まで入力できます。ラベルを指定しない場合、メトリック名がメトリックを識別するために使用されます。

注記

ラベルに二重引用符が含まれる場合、二重引用符の前に円記号 (¥) を付けます。たとえば、ラベルが **"my" data** である場合、**¥"**my**¥"** data と入力します。

metric_label_name は、CAPACITY や ACTION など、DSI クラスの仕様構文のいずれのキーワード要素とも異なる必要があります。

例

```
METRICS
RUN_Q_PROCS  = 106
LABEL  "Procs in run q";
```

要約方法

要約方法では、CLASS セクションの RECORDS PER HOUR オプションで設定されたレコード数を超えたレコードが届いた場合に、データを要約する方法を決定します。たとえば、実際のメトリックを合計するときに割合を平均する場合があります。要約方法は数値メトリックに対してのみ有効です。

構文

[{TOTALED | AVERAGED | SUMMARIZED BY *metric_name*}]

使い方

SUMMARIZED BY は、メトリックを時間ではなくクラス内の別のメトリックで平均する場合に使用します。たとえば、TOTAL_ORDERS と LINES_PER_ORDER の各メトリックを定義していると仮定します。これらのメトリックは 5 分ごとにログ処理されますが、レコードが 1 時間に 1 回しか書き込まれないとしても、LINES_PER_ORDER は全ライン数を全オーダー数で割った値になるように正確に要約するため、ログ処理では次の計算を 5 分ごとに行う必要があります。

- 5 分のインターバルが終わるたびに、LINES_PER_ORDER と TOTAL_ORDERS を掛けて、その結果を全ラインの内部実行回数として保持します。
- TOTAL_ORDERS の実行回数を保持します。
- 1 時間後に全ライン数を TOTAL_ORDERS で割ります。

この種の計算を指定するには、LINES_PER_ORDER を SUMMARIZED BY TOTAL_ORDERS として指定します。

要約方法を指定しない場合、メトリックはデフォルトで AVERAGED になります。

例

```
METRICS
ITEM_1_3 = 11203
LABEL    "TOTAL_ORDERS"
TOTALED;
ITEM_1_5   = 11205
LABEL   "LINES_PER_ORDER"
SUMMARIZED BY ITEM_1_3;
```

PRECISION

PRECISION はメトリック値の小数点以下の桁数を識別します。PRECISION を指定しない場合は、指定した MAXIMUM に基づいて計算します。いずれの値も指定しない場合、PRECISION はデフォルトの 3 になります。この設定は数値メトリックに対してのみ有効です。

構文

[PRECISION{0|1|2|3|4|5}]

使い方

PRECISION の設定により記録可能な最大値が決まります。整数には PRECISION 0 を使用します。

PRECISION	小数点以下の桁数	最大許容値	MAXIMUM
0	0	2,147,483,647	> 10,000
1	1	214,748,364.7	1001 ～ 10,000
2	2	21,474,836.47	101 ～ 1,000
3	3	2,147,483.647	11 ～ 1,000
4	4	214,748.3647	2 ～ 10
5	5	21,474.83647	1

例

```
METRICS
RUN_Q_PROCS        = 106
LABEL      "Procs in run q"
PRECISION 1;
```

TYPE TEXT LENGTH

TYPE TEXT LENGTH の 3 つのキーワードは、メトリックが数値ではなくテキストであることを指定します。テキストは、^d、\n または区切り記号 (使用している場合) 以外の任意の文字として定義します。

dsilog に入力するデータ項目間のデフォルトの区切り記号はスペースであるため、テキストにスペースが含まれる場合、区切り記号を変更する必要があります。異なる区切り記号を指定するには、第 15 章の「DSI プログラムに関する参考資料」で説明するように dsilog -c char オプションを使用します。

構文

[TYPE TEXT LENGTH *length***]**

使い方

length はゼロより大きく、4096 より小さい数である必要があります。

注記

要約方法、MAXIMUM、および PRECISION はいずれもテキスト メトリックでは指定できません。テキストは要約できないため、dsilog はインターバル内で最初に記録された値を取り込み、残りを無視します。

例

```
METRICS
text_1 = 16
LABEL "first text metric"
TYPE TEXT LENGTH 20
;
```

クラス仕様のサンプル

```
CLASS VMSTAT_STATS = 10001
LABEL  "VMSTAT data"
INDEX BY HOUR
MAX INDEXES  12
ROLL BY HOUR
RECORDS PER HOUR  120;


METRICS

RUN_Q_PROCS       = 106
LABEL     "Procs in run q"
PRECISION 0;

BLOCKED_PROCS     = 107
LABEL     "Blocked Processes"
PRECISION 0;

SWAPPED_PROCS     = 108
LABEL     "Swapped Processes"
PRECISION 0;

AVG_VIRT_PAGES    = 201
LABEL     "Avg Virt Mem Pages"
PRECISION 0;

FREE_LIST_SIZE    = 202
LABEL     "Mem Free List Size"
PRECISION 0;

PAGE_RECLAIMS     = 303
LABEL     "Page Reclaims"
PRECISION 0;


ADDR_TRANS_FAULTS = 304
LABEL     "Addr Trans Faults"
PRECISION 0;

PAGES_PAGED_IN    = 305
LABEL     "Pages Paged In"
PRECISION 0;



PAGES_PAGED_OUT   = 306
LABEL     "Pages Paged Out"
PRECISION 0;

PAGES_FREED       = 307
LABEL     "Pages Freed/Sec"
PRECISION 0;
```

```
MEM_SHORTFALL     = 308
LABEL     "Exp Mem Shortfall"
PRECISION 0;

CLOCKED_PAGES     = 309
LABEL     "Pages Scanned/Sec"
PRECISION 0;

DEVICE_INTERRUPTS = 401
LABEL     "Device Interrupts"
PRECISION 0;

SYSTEM_CALLS      = 402
LABEL     "System Calls"
PRECISION 0;

CONTEXT_SWITCHES  = 403
LABEL     "Context Switches/Sec"
PRECISION 0;

USER_CPU          = 501
LABEL     "User CPU"
PRECISION 0;

SYSTEM_CPU        = 502
LABEL     "System CPU"
PRECISION 0;

IDLE_CPU          = 503
LABEL     "Idle CPU"
PRECISION 0;
```

15 DSI プログラムに関する参考資料

この章では、次の参照情報についての詳細を説明します。

- sdlcomp コンパイラ
- datasources 構成ファイルと alarmdef 構成ファイル
- dsilog によるログ処理
- Performance Collection Component の extract プログラムによる DSI データのエクスポート
- データ ソース管理ユーティリティ sdlutil

sdlcomp コンパイラ

sdlcomp コンパイラは、クラスの仕様ファイルのエラーを検査します。エラーが検出されない場合は、ユーザーが指定するログ ファイル セットの記述ファイルにクラスおよびメトリックの記述が追加されます。また、sdlcomp コンパイラは、データの保存に使用するログ ファイルに対してログ ファイル セットのルート ファイルのポインタをセットアップします。ログ ファイル セットまたはログ ファイルのいずれかがない場合は、コンパイラにより作成されます。

コンパイラ コマンド内でフル パスを指定すると、DSI ファイルをシステム上の任意の場所に配置できます。ただし、パスを指定した後に、DSI ログ ファイルを異なるディレクトリに*移動しないようにしてください*。SDL62 は第 17 章の「SDL エラー メッセージ」に記載されている関連クラス仕様のエラー メッセージです。クラス仕様のエラー メッセージ用に DSI で使用するフォーマットは、接頭辞 SDL (Self Describing Logfile) + 「メッセージ番号」です。

コンパイラ構文

```
sdlcomp [-maxclass value] specification_file
   [logfile_set[log file]] [options]
```

変数とオプション	定義
-maxclass *value*	新しいログ ファイル セットを作成するときに入力されるクラスの最大数を指定できます。このオプションは既存のログ ファイル セット名と同時に使用すると無視されます。各追加クラスは、そのクラスが使用されているかどうかにかかわらず、オーバーヘッド内で約 500 バイトのディスク領域を消費します。-maxclass を指定していない場合デフォルトは 10 です。
specification_file	クラスの仕様を含むファイル名です。このファイルが現在のディレクトリにない場合、完全修飾名を指定する必要があります。
logfile_set	対象となるクラスを追加するログ ファイル セットの名前です。
log file	対象となるクラスに関するデータを含むセット内のログ ファイルです。ログ ファイルを指定しない場合、新しいログ ファイルがクラスに対して作成され、自動的に名前が付けられます。
-verbose	コンパイラ出力の詳細説明を stdout に出力します。

変数とオプション	定義
-vers	バージョン情報を表示します。
-?	構文の説明を表示します。
-u	1 秒当りに複数のレコードを記録できます。このオプションは、要約されていないデータのみを記録するのに使用します。

コンパイラ出力のサンプル

次のコマンド ラインを仮定します。

```
->sdlcomp vmstat.spec sdl_new
```

次のコードは正常終了したコンパイルの出力サンプルです。vmstat.spec は第 3 章で示した仕様ファイルのサンプルであることに注意してください。

```
sdlcomp
Check class specification syntax.

CLASS VMSTAT_STATS = 10001
LABEL  "VMSTAT data"
INDEX BY HOUR
MAX INDEXES  12
ROLL BY HOUR
RECORDS PER HOUR  120;

METRICS

RUN_Q_PROCS       = 106
LABEL     "Procs in run q"
PRECISION 0;

BLOCKED_PROCS     = 107
LABEL     "Blocked Processes"
PRECISION 0;

SWAPPED_PROCS     = 108
LABEL     "Swapped Processes"
PRECISION 0;

AVG_VIRT_PAGES    = 201
LABEL     "Avg Virt Mem Pages"
PRECISION 0;

FREE_LIST_SIZE    = 202
LABEL     "Mem Free List Size"
PRECISION 0;

PAGE_RECLAIMS     = 303
LABEL     "Page Reclaims"
PRECISION 0;
```

```
ADDR_TRANS_FAULTS = 304
LABEL     "Addr Trans Faults"
PRECISION 0;

PAGES_PAGED_IN    = 305
LABEL     "Pages Paged In"
PRECISION 0;

PAGES_PAGED_OUT   = 306
LABEL     "Pages Paged Out"
PRECISION 0;

PAGES_FREED       = 307
LABEL     "Pages Freed/Sec"
PRECISION 0;

MEM_SHORTFALL     = 308
LABEL     "Exp Mem Shortfall"
PRECISION 0;

CLOCKED_PAGES     = 309
LABEL     "Pages Scanned/Sec"
PRECISION 0;

DEVICE_INTERRUPTS = 401
LABEL     "Device Interrupts"
PRECISION 0;

SYSTEM_CALLS      = 402
LABEL     "System Calls"
PRECISION 0;

CONTEXT_SWITCHES  = 403
LABEL     "Context Switches/Sec"
PRECISION 0;

USER_CPU          = 501
LABEL     "User CPU"
PRECISION 0;

SYSTEM_CPU        = 502
LABEL     "System CPU"
PRECISION 0;

IDLE_CPU          = 503
LABEL     "Idle CPU"
PRECISION 0;
Note:  Time stamp inserted as first metric by default.

Syntax check successful.

Update SDL sdl_new.
Open SDL sdl_new
Add class VMSTAT_STATS.
```

```
Check class VMSTAT_STATS.

Class VMSTAT_STATS successfully added to log file set.
```

エラー メッセージと回復に関する説明は、第 17 章の「エラー メッセージ」を参照してください。

構成ファイル

データの記録を開始する前に、2 つの Performance Collection Component 構成ファイルを更新する必要があります。

- `/var/opt/OV/conf/perf/datasources`
- `/var/opt/perf/alarmdef` – `alarmdef` 構成ファイルの使用に関する詳細情報は、次の項の「DSI メトリックのアラーム定義」を参照してください。

DSI メトリックのアラーム定義

Performance Collection Component を使用すると、DSI メトリックに関するアラームを定義できます。アラームは、DSI メトリックが定義条件を満たす場合や超える場合にユーザーに通知します。アラームを定義するには条件を指定し、その条件を満たす場合や超える場合に、アラートの通知またはアクションをトリガします。Performance Collection Component システム上の `alarmdef` ファイルにある別の Performance Collection Component メトリックのアラームと同様に、DSI により記録されたデータのアラームを定義します。`alarmdef` ファイルは Performance Collection Component の `var/opt/perf/` 構成ディレクトリに配置されています。

アラーム定義内の DSI メトリック名は必ず完全修飾名で指定します。次に示すように、DSI メトリック名の直前に *datasource_name* と *class_name* を指定します。

datasource_name:*class_name*:`metric_name`

- *datasource_name* は、`datasources` ファイル内でデータ ソースを構成するために使用している名前です。詳細は、『HP Operations エージェント インストール、構成ガイド』の「データ ソースの構成」の項を参照してください。
- *class_name* は、データ ソースのクラスの仕様でクラスを識別するために使用している名前です。クラスの仕様でメトリック名が一意である (再使用されていない) 場合、*class_name* を入力する必要はありません。
- `metric_name` は、データ ソースのクラスの仕様に含まれるデータ項目です。

ただし、メトリックに完全修飾名を指定しない場合は、使用するデータ ソースを識別するために `alarmdef` ファイルに USE 文を含む必要があります。USE 文に関する詳細は、『HP Operations エージェント for UNIX ユーザー マニュアル』第 7 章の「パフォーマンス アラーム」を参照してください。

アラーム ジェネレータが読み込めるように `alarmdef` ファイルの変更をアクティブにするには、コマンド ラインに **`ovpa restart alarm`** コマンドを入力します。

アラーム定義構文、アラームの処理方法、およびアラーム定義のカスタマイズに関する詳細は、『HP Operations エージェント for UNIX ユーザー マニュアル』第 7 章を参照してください。

アラーム処理

`dsilog` はデータを記録すると、そのデータを `alarmdef` ファイル内のアラート定義と比較して条件を満たしていること、または超えていることを確認します。データが条件を満たしている場合または超えている場合、アラートの通知またはアクションがトリガされます。

アラームの通知を送信する場所や、ローカル アクションの実行の有無が構成できます。アラームの通知を中央 Performance Manager 分析システムに送信し、そこでシステム パフォーマンスの特徴を示すメトリックのグラフを作成できます。SNMP トラップは、HP Network Node Manager に送信できます。Performance Collection Component システム上で、ローカルな動作を実行できます。アラーム情報も Operations Manager に送信できます。

dsilog によるログ処理

dsilog 処理では、データにアクセスするためにユーザーが独自のプログラムを作成するか、既存のプログラムを使用する必要があります。データを dsilog に渡すとデータをログ ファイル セットに記録できます。定義する各クラスに対して個別のログ処理を実行する必要があります。

dsilog プログラムは stdin からデータを受信することを予測します。ログ処理を開始するには、次の例で示すように、データを収集する処理の出力を dsilog に渡します。

```
vmstat 60 | dsilog logfile_set class
```

コマンド ラインで使用できるのは 1 つのパイプ (|) のみです。これは、パイプを 2 つ使用すると、UNIX バッファリングにより最初のコマンドの出力が遅延するためです。2 つ目のコマンドが実行されログ ファイルにデータが渡されるまで 8000 文字が書き込まれます。

fifo (指定済みのパイプ) を使用することもできます。次に例を示します。

```
mkfifo -m 777 myfifo
dsilog logfile_set class -i myfifo &
vmstat 60 > myfifo &
```

& により、バック グラウンドで処理が実行されます。

dsilog 処理を多数実行する場合、UNIX カーネル パラメータ shmmni と nflocks の値を増やす必要があることに注意してください。shmmni は共有メモリの最大セグメント数を指定し、nflocks はシステム上の最大ファイル ロック数を指定します。どちらのパラメータもデフォルト値は 200 です。アクティブな DSI ログ ファイル セットは、それぞれ 1 つの共有メモリのセグメント (shmmni) と 1 つ以上のファイル ロック (nflocks) を使用します。HP-UX では、システム管理および保守ユーティリティ (SAM) を使用して、shmmni と nflocks の設定を変更できます。

構文

dsilog logfile_set class [*options***]**

dsilog パラメータとオプションについては、次のページ以降で説明します。

表 1　　dsilog パラメータおよびオプション

変数とオプション	定義
logfile_set	データを保存するログ ファイル セットの名前です。現在のディレクトリにない場合、完全修飾名である必要があります。
class	記録されるクラスの名前です。
-asyn	データが RECORDS PER HOUR の比率で非同期的に届くように指定します。ログ インターバル中にデータが届かない場合、最後のログ インターバルのデータが繰り返されます。ただし、dsilog がまだデータを記録していない場合、記録されたメトリック値は欠落データとして扱われます。これにより直線がデータのグラフィック表示に描画され、データがエクスポートされるときに各レコードのデータが繰り返されます。
-c char	指定文字を文字列の区切り記号として使用します。整数、マイナス符号、^z、\n は区切り記号として使用できません。いずれかのテキスト メトリックにスペースが含まれている場合、このオプションを指定して、固有の区切り記号を指定する必要があります。
-f *format file*	ログ処理に入力されるデータを説明するファイルを指定します。このオプションを指定しない場合、dsilog は次の仮定を用いてクラスの仕様から入力のフォーマットを派生させます。詳細は、この章で後述している「フォーマット ファイルの作成」を参照してください。 入力レコード内の各データ項目は、クラスの仕様で定義されているメトリックに対応しています。 メトリックは、入力レコード内でデータ項目として表示される順序で、クラスの仕様に定義されています。 メトリックの定義より多くのデータ項目が入力レコードにある場合、dsilog は追加のデータ項目をすべて無視します。
-f *format file* (続き)	入力データ項目より多くのメトリックの定義をクラスの仕様でリストしている場合、このフィールドは、データをエクスポートするときに「欠落」データを表示し、分析ソフトウェアでデータをグラフ化するときには、そのメトリックにデータを使用できません。 フォーマット ファイル内のフィールドの数は、100 までに制限されています。

表 1　　dsilog パラメータおよびオプション

変数とオプション	定義
-i fifo または ASCII file	指定の fifo または ASCII ファイルから入力されることを示します。このオプションを使用しない場合、stdin から入力されます。このオプションを使用する場合は、収集処理を開始する前に dsilog を起動します。fifo の使用についての詳細は、man ページの mkfifo を参照してください。また、使用例については、第 16 章の「データ ソースの統合例」を参照してください。
-s seconds	データを要約するまでの秒数です。-s オプションは、クラス仕様中の RECORDS PER HOUR への要約間隔および要約レートのデフォルトを変更します。また、このオプションは RECORDS PER HOUR の値を変更します。 ゼロ (0) は要約を終了し、すべての受信 データが記録されます。-s 0 オプションは注意が必要です。dsilog は、そのポイントになった時、ログデータにタイムスタンプします。これは、通常の間隔でタイムスタンプを行なう Performance Manager と perfalarm の不具合になります。ログファイルが Performance Manager によってアクセスされる場合、-s 0 オプションは使用しないことをお勧めします。
-t	stdout に記録されるすべてのデータを ASCII フォーマットで印刷します。
-timestamp	ログ処理がタイムスタンプを提供せず、入力データですでに提供されているタイムスタンプを使用することを示します。受信データ内のタイムスタンプは、UNIX タイムスタンプ フォーマット (1/1/70 00:00:00 からの秒数) で、各地の時間を表す必要があります。
-vi	dsilog により入力をフィルタ処理し、エラーをログ ファイルではなく stdout に書き込みます。これは、記録された実データを stdout に書き込みません (次の -vo オプションを参照)。これを使用すると、入力の妥当性を確認できます。
-vo	dsilog により入力をフィルタ処理し、記録された実データとエラーをログ ファイルではなく stdout に書き込みます。これを使用すると、データ要約の妥当性を確認できます。
-vers	バージョン情報を表示します。
-?	構文の説明を表示します。

dsilog によるデータ処理方法

dsilog プログラムは各入力データ文字列を走査し、区切られたフィールドを個別の数値メトリックまたはテキスト メトリックに分割します。データの処理方法を予測するための重要な規則は、入力文字列の妥当性です。有効な入力文字列では、区切り記号がすべての指定メトリック タイプ (数値またはテキスト) 間に含まれる必要があります。デフォルトの区切り記号はスペースですが、dsilog -c char コマンド ライン オプションを使用すると別の区切り記号を指定できます。

DSI がレコードを正しく解釈するために、DSI に送り込まれるすべてのレコードの最後に新しい文字列を含む必要があります。

sdlgendata によるログ処理のテスト

データの記録を開始する前に sdlgendata プログラムを使用して、コンパイルされたログ ファイル セットとログ処理をテストできます。sdlgendata はクラスのメトリックを検出し、クラス内の各メトリックについてデータを作成します。

構文

sdlgendata logfile_set class [*options***]**

sdlgendata のパラメータとオプションを次に示します。

表 2　　Sdlgendata パラメータおよびオプション

変数とオプション	定義
logfile_set	データを生成するログ ファイル セットの名前です。
class	データを生成するデータ クラスです。
-timestamp [*number*]	データと共にタイムスタンプを示します。負の数字を入力する場合、または数字を入力しない場合、現在の時刻が timestamp に使用されます。正の数字を使用すると時間は 0 から始まり、個別の新規データ レコードについて *number* で示す数だけ増加します。
-wait *number*	生成されるレコード間で、*number* で示す秒数分待機します。
-cycle number	*number* で示すサイクル回数の後に、データを再利用します。
-vers	バージョン情報を表示します。
-?	構文の説明を表示します。

-vi または -vo オプションを使用して sdlgendata の出力を dsilog プロセスに渡すことにより、独自のプロセスまたはプログラムによる記録を開始する前に入力 (-vi) や出力 (-vo) を検査できます。

テストが終了すると、テストで作成したログ ファイルはすべて削除します。削除しない場合、これらのファイルはログ ファイル セットの中に残ります。

次のコマンドを使用して、sdlgendata からログ処理にデータを渡します。-vi オプションは、データが廃棄されてエラーが stdout に書き込まれることを指定します。データ生成を終了するには、**CTRL+C** またはその他の割り込み制御文字を押します。

```
sdlgendata logfile_set class -wait 5 | dsilog \
logfile_set class -s 10 -vi
```

上記のコマンドにより、次のようなデータが生成されます。

```
dsilog
I:  744996402  1.0000  2.0000  3.0000   4.0000   5.0000   6.0000    7.0000
I:  744996407  2.0000  3.0000  4.0000   5.0000   6.0000   7.0000    8.0000
I:  744996412  3.0000  4.0000  5.0000   6.0000   7.0000   8.0000    9.0000
I:  744996417  4.0000  5.0000  6.0000   7.0000   8.0000   9.0000   10.0000
I:  744996422  5.0000  6.0000  7.0000   8.0000   9.0000   10.0000  11.0000
I:  744996427  6.0000  7.0000  8.0000   9.0000   10.0000  11.0000  12.0000
I:  744996432  7.0000  8.0000  9.0000   10.000   11.000   12.0000  13.0000
I:  744996437  8.0000  9.0000  10.0000  11.0000  12.0000  13.0000  14.0000
```

また、dsilog の -vo オプションを使用して実際に実データを記録する前に、実データの入力および要約出力を検査できます。次のコマンドにより、vmstat は 5 秒間のインターバルで dsilog に渡され、10 秒間に要約されます。

```
->vmstat 5 | dsilog logfile_set class -s 10 -vo

dsilog
I: 744997230  0.0000  0.0000  21.0000  2158.0000  1603.0000  2.0000  2.0000
I: 744997235  0.0000  0.0000  24.0000  2341.0000  1514.0000  0.0000  0.0000
interval marker
L: 744997230  0.0000  0.0000  22.5000  2249.5000  1558.5000  1.0000  1.0000

I: 744997240  0.0000  0.0000  23.0000  2330.0000  1513.0000  0.0000  0.0000
I: 744997245  0.0000  0.0000  20.0000  2326.0000  1513.0000  0.0000  0.0000
interval marker
L: 744997240  0.0000  0.0000  21.5000  2328.0000  1513.0000  0.0000  0.0000

I: 744997250  0.0000  0.0000  22.0000  2326.0000  1513.0000  0.0000  0.0000
I: 744997255  0.0000  0.0000  22.0000  2303.0000  1513.0000  0.0000  0.0000
interval marker
L: 744997250  0.0000  0.0000  22.0000  2314.5000  1513.0000  0.0000  0.0000

I: 744997260  0.0000  0.0000  22.0000  2303.0000  1512.0000  0.0000  0.0000
I: 744997265  0.0000  0.0000  28.0000  2917.0000  1089.0000  9.0000 33.0000
interval marker
L: 744997260  0.0000  0.0000  25.0000  2610.0000  1300.5000  4.5000 16.5000
```

```
I: 744997270  0.0000  0.0000  28.0000  2887.0000  1011.0000  3.0000  9.0000
I: 744997275  0.0000  0.0000  27.0000  3128.0000   763.0000  8.0000  6.0000
interval marker
L: 744997270  0.0000  0.0000  27.5000  3007.5000   887.0000  5.5000 12.5000
```

dsilog -vo オプションを使用すると、古いデータのファイルもテストに使用できます。ただし、データに独自の UNIX タイムスタンプ (1/1/70 00:00:00 からの秒数) が含まれている場合に限ります。古いデータのファイルを使用するには、次のようなコマンドを入力します。

dsilog -timestamp -vo <*oldfile*>

フォーマット ファイルの作成

次の場合、フォーマット ファイルを作成して、データ入力をクラスの仕様にマップします。

- データ入力が、クラスの仕様に含まれないデータを含んでいる場合
- 受信データのメトリックの順序が、クラスの仕様で指定した順序とは異なる場合

フォーマット ファイルは、vi または任意のテキスト エディタで作成できる ASCII テキスト ファイルです。dsilog の -f オプションを使用して、フォーマット ファイルの完全修飾名を指定します。

次のメトリックを開始するために、区切り記号 (デフォルトのスペース、または dsilog の -c オプションによるユーザー定義の記号) の後の最初の有効な文字を検索することによりログ処理は実行されるため、フォーマット ファイルはスキップするフィールドとスキップしないフィールドと関連するメトリック名のみをログ処理に通知します。

$numeric は、数値メトリック フィールドを 1 つスキップして次に進むようにログ処理に通知します。$any は、テキスト メトリック フィールドを 1 つスキップして次に進むようにログ処理に通知します。フォーマット ファイルは 100 フィールドまでに制限されることに注意してください。

たとえば、受信データ ストリームには次の情報が含まれています。

```
ABC 987 654 123 456
```

ここで、最初の数値フィールドのみをメトリック metric_1 に記録する場合、フォーマット ファイルは次のようになります。

```
$any metric_1
```

これは、最初の数値フィールドの情報のみを記録し、残りのデータを廃棄するようにログ処理に通知します。3 番目の数値フィールドの情報のみを記録する場合、フォーマット ファイルは次のようになります。

```
$any $numeric $numeric metric_1
```

4 つの数値データ項目すべてを逆の順序で記録する場合、フォーマット ファイルは次のようになります。

```
$any metric_4 metric_3 metric_2 metric_1
```

受信データ ストリームには次の情報が含まれています。

```
/users    15.9    3295    56.79%    xdisk1 /dev/dsk/
c0d0s*
```

最初のテキスト メトリックと最初の 2 つの数値フィールドを、それぞれ *text_1*、*num_1*、*num_2* と命名してメトリック フィールドに記録する場合、フォーマット ファイルは次のようになります。

```
text_1    num_1    num_2
```

これは、最初の 3 つのフィールドの情報のみを記録し、残りのデータを廃棄するようにログ処理に通知します。

すべてのデータを記録するが、3 番目のメトリックの後の「%」は廃棄する場合、フォーマット ファイルは次のようになります。

```
text_1    num_1    num_2    num_3    $any    text_2    text_3
```

現在、数値フィールドを記録しており、「%」はテキスト フィールドとみなされるため、「%」の後のテキスト フィールドを正しく記録するには「%」をスキップする必要があります。

データ項目を異なる順序で記録する場合、フォーマット ファイルは次のようになります。

```
text_3    num_2    num_1    num_3    $any    text_2    text_1
```

クラスの仕様で *text_1* の長さが 6 文字であると宣言されている場合、*text_3* の最初の 6 文字のみが記録されることに注意してください。*text_3* のすべてを最初の値として記録するには、クラスの仕様を変更して余分な領域を使用できるようにデータ ストリームを変更します。

クラス仕様の変更

クラスの仕様ファイルを変更するには、次のようにログ ファイル セット全体を再度作成する必要があります。

1. `dsilog` プロセスを終了します。
2. データを保存する場合または古いデータと記録する新しいデータを統合する場合、UNIX タイムスタンプ オプションを使用して既存のログ ファイルからデータをエクスポートします。この方法については、この章で後述している「DSI データのエクスポート」を参照してください。
3. `sdlutil` を実行してログ ファイル セットを削除します。この方法については、この章で後述している「sdlutil によるデータの管理」を参照してください。
4. クラスの仕様ファイルを更新します。
5. `sdlcomp` を実行して、クラスの仕様を再コンパイルします。
6. 任意で `dsilog` で `-i` オプションを使用し、手順 2 でエクスポートした古いデータに統合します。場合によっては、`-f` *format_file* オプションを使用して、新しいデータと統合するデータを変更する必要があります。
7. `dsilog` を実行して、新しいクラスの仕様に基づく記録を開始します。
8. ログ ファイル セットの名前または位置を変更しない限り、`datasources` ファイルを更新する必要はありません。

DSI データのエクスポート

DSI ログ ファイルからデータをエクスポートするには、Performance Collection Component の extract プログラムの export 機能を使用します。extract プログラムを使用してデータをエクスポートする方法についての詳細は、『HP Operations エージェント for UNIX ユーザー マニュアル』第 5 章および第 6 章を参照してください。コマンド ラインの引数を使用して DSI データをエクスポートする例を次のページに示します。

DSI ログ ファイルからエクスポートできるクラスとメトリックを検出するには、いくつかの方法があります。この章で後述する「sdlutil によるデータの管理」で説明されているように、sdlutil を使用してこの情報をリストします。または、extract guide コマンドを使用して DSI ログ ファイルにクラスとメトリックをリストするエクスポート テンプレート ファイルを作成します。vi を使用して、ファイルを編集、命名、保存できます。『HP Operations エージェント for UNIX ユーザー マニュアル』第 5 章および第 6 章で記述した方法で、エクスポート テンプレート ファイルを使用してエクスポート フォーマットを指定します。

DSI ログ ファイル データをエクスポートするには、ユーザーが root またはログ ファイルの作成者であることが必要です。

DSI ログファイル データをエクスポートする extract プログラムの使用例

```
extract -xp -l logfile_set -C class [options]
```

extract コマンド ライン オプションを使用して、次の動作が実行できます。

- エクスポート出力ファイルの指定
- エクスポート対象の最初と最後のインターバルを開始する日時および終了する日時の設定
- 特定時間 (シフト) に限定したデータのエクスポート
- 特定曜日 (週末など) のデータの除外
- レポートのメトリック間に挿入する区切り記号の指定
- 到着するデータがないインターバルや表示された値が欠落データやヌル データ用であるインターバルに対して、見出しや空白レコードを表示するかどうかの選択
- エクスポートされた日付と時刻の表示 (UNIX フォーマットまたは日付フォーマット)
- 追加の要約レベルの設定

Performance Manager でのデータの表示

Performance Manager で DSI ログ ファイルのデータを表示するには、DSI ログ ファイルを Performance Collection Component データ ソースとして構成する必要があります。データの記録を開始する前に、Performance Collection Component システムの datasources ファイルにデータを追加してデータ ソースを構成します。

Performance Manager を使用すると、DSI データの傾向の表示、監視、分析、比較、予測を集中処理できます。Performance Manager は、現在の問題や将来起こる可能性のある問題を識別するときに役立ちます。Performance Manager は、ユーザーの生産性に影響が生じる前に、問題解決に必要な情報を提供します。

sdlutil によるデータの管理

DSI ログ ファイルのデータを管理するには、sdlutil プログラムを使用して次の作業のいずれかを実行します。

- 現在定義されているクラスとメトリック情報を stdout にリストします。出力をファイルにリダイレクトできます。
- クラスの詳しい統計値を stdout にリストします。
- リストしたすべてのメトリックに関するメトリックの記述を表示します。
- ログ ファイル セット内のファイルをリストします。
- ログ ファイル セットからクラスとデータを削除します。
- ログ ファイル セット内の情報からクラスの仕様を再度作成します。
- バージョン情報を表示します。

構文

sdlutil logfile_set [*option***]**

変数とオプション	定義
logfile_set	クラスの仕様をコンパイルして作成したログ ファイル セットの名前です。
-classes *classlist*	リストされたすべてのクラスに関するクラスの記述を提示します。クラスがリストされていない場合、すべてのクラスの記述が提示されます。*classlist* 内の項目はスペースで区切ります。
-stats *classlist*	リストされたすべてのクラスの詳しい統計値を提示します。クラスがリストされていない場合、すべてのクラスの記述が提示されます。*classlist* 内の項目はスペースで区切ります。
-metrics *metriclist*	*metriclist* 内のすべてのメトリックに関するメトリックの記述を提示します。クラスがリストされていない場合、すべてのクラスの記述が提示されます。*metriclist* 内の項目はスペースで区切ります。
-id	ログ ファイルが使用する共有メモリ セグメント ID を表示します。
-files	ログ ファイル セット内のすべてのファイルをリストします。

変数とオプション	定義
-rm all	ログ ファイルからすべてのクラスとデータ、およびクラスに関するデータと共有メモリ ID を削除します。
-decomp *classlist*	ログ ファイル セット内の情報からクラスの仕様を再度作成します。この結果は stdout に書き込まれます。ファイルを変更して再使用する場合、ファイルに結果をリダイレクトします。*classlist* 内の項目はスペースで区切ります。
-vers	バージョン情報を表示します。
-?	構文の説明を表示します。

16 データ ソースの統合例

データ ソース統合は極めて強力で柔軟な技術です。単純なものから非常に複雑なものまで DSI を実装できます。

この章では、DSI を使用して次の作業を使用する例を示します。

- `dsilog` スクリプトの記述
- `vmstat` データの記録
- `sar` データの記録
- `who` ワード カウントの記録

dsilog スクリプトの記述

dsilog コードは、入力として、連続したデータ列の流れを扱うように考慮されています。入力されたデータ列は、各クラスの仕様指示による dsilog と、要求された要約間隔で記録される 1 つの要約されたデータ列によって要約されます。ログの中に書き込まれたタイムスタンプが、予想した要約割合 (レコード / 時) に一致する場合、Performance Manager と perfalarm は最良な動作になります。これは dsilog が要約を行うことを許された場合、自動的に発生します。

スクリプトによっては各到着する入力列ごとに新しい dsilog プロセスが実行されます。これは、Performance Manager と perfalarm に関する問題を引き起こす原因になるために推奨しません。

- 問題となる dsilog スクリプト
- 推奨する dsilog スクリプト

例 1 - 問題となる dsilog スクリプト

次のスクリプトでは、新しいプロセス dsilog が各到着する入力列で実行されます。

```
while :
do
   feed_one_data_row | dsilog sdlname classname
   sleep 50
done
```

例 2 - 推奨する dsilog スクリプト

次のスクリプトでは、1 つの dsilog プロセスが、連続した入力データ列を受け付けます。feed_one_data_row は、連続的なデータ列を単一の dsilog プロセスに供給する機能として書かれています。

```
# Begin data feed function
feed_one_data_row()
  {
   while :
   do
#  Perform whatever operations necessary to produce one row
#  of data for feed to a dsilog process
   sleep 50
   done
  }
#  End data feed function


#  Script mainline code
   feed_one_data_row | dsilog sdlname classname
```

vmstat データの記録

この例は、vmstat からレポートされる最初の 2 つの値を記録するために、デフォルト設定を使用してデータ ソース統合を準備する方法を示します。データ ソース統合処理機能の概要としてこの項を参照する、または各作業を実行してシステム上で等しい DSI ログ ファイルを作成することもできます。

データ ソース統合を実装するには、次の手順を実行することが必要です。

- クラスの仕様ファイルの作成
- クラスの仕様ファイルのコンパイル
- dsilog によるログ処理の開始

クラス仕様のファイルの作成

クラスの仕様ファイルは、クラス内でメトリックとして記録する個別の数字と、クラス、すなわち受信データのセットを説明するために作成するテキスト ファイルです。ファイルは好みのテキスト エディタを使用して作成されます。ここで示すデータ ソース統合例のファイルは /tmp/ ディレクトリに作成されます。

次の例では、VMSTAT_STATS の名前が付いたクラス内で最初の 2 つの vmstat 数値を記述するために必要なクラスの仕様ファイルを示します。このクラスに定義されたメトリックは 2 つであるため、ログ プロセスは各 vmstat 出力レコードの残りを無視します。ファイル内の各ラインの内容は次の注釈ラインで説明されます。

```
CLASS VMSTAT_STATS = 10001;
  # Assigns a unique name and number to vmstat class data.
  # The semicolon is required to terminate the class section
  # of the file.

METRICS
  # Indicates that everything that follows is a description
  # of a number (metric) to be logged.

RUN_Q_PROCS = 106;
      # Assigns a unique name and number to a single metric.
      # The semicolon is required to terminate each metric.

BLOCKED_PROCS = 107;
      # Assigns a unique name and number to another metric.
# The semicolon is required to terminate each metric.
```

クラス仕様ファイルのコンパイル

sdlcomp を使用してクラスの仕様ファイルをコンパイルする場合、ファイルを検査して構文エラーを検出します。エラーが検出されない場合、sdlcomp はログ ファイル セットを作成または更新して、クラスのデータを保持します

クラスの仕様ファイルに付けたファイル名を使用し、*logfile_set_name* の名前を指定します。これにより、ログ ファイルに含まれるデータの種類が容易に記憶できます。次に示すコマンドおよびコンパイラ出力の例で、/tmp/vmstat.spec はファイル名として使用され、/tmp/VMSTAT_DATA はログ ファイル セットに使用されます。

```
-> sdlcomp /tmp/vmstat.spec /tmp/VMSTAT_DATA

sdlcomp X.01.04
Check class specification syntax.

CLASS VMSTAT_STATS = 10001;

METRICS
RUN_Q_PROCS       = 106;
BLOCKED_PROCS     = 107;

NOTE: Time stamp inserted as first metric by default.

Syntax check successful.

Update SDL VMSTAT_DATA.
Shared memory ID used by vmstat_data=219

Class VMSTAT_STATS successfully added to log file set.
```

この例では /tmp/ ディレクトリに VMSTAT_DATA ログ ファイル セットが作成され、データ ファイルにルート ファイルと記述ファイルが追加されます。ログ ファイル セットは記録されたデータをいつでも受信できます。クラスの仕様ファイルに構文エラーがある場合は障害を示すメッセージが表示され、ログ ファイル セットは作成されません。

dsilog によるログ処理の開始

vmstat の出力を直接 dsilog ログ処理に渡します。次のコマンドを使用します。

```
vmstat 60 | dsilog /tmp/VMSTAT_DATA VMSTAT_STATS &
```

このコマンドは 60 秒ごとに vmstat を実行し、出力を VMSTAT_DATA ログ ファイル セット内の VMSTAT_STATS クラスに直接渡します。コマンドはバック グラウンドで実行されます。また、remsh を使用してリモート システムから vmstat を送信できます。

ログ処理の始めに、次のメッセージが生成されることに注意してください。

```
Metric null has invalid data
Ignore to end of line, metric value exceeds maximum
```

このメッセージは、vmstat の出力で dsilog が記録できないヘッダー ラインの結果です。このメッセージが画面に表示されても dsilog は実行を続行し、最初の有効な入力ラインからデータの記録を開始します。

データへのアクセス

sdlutil プログラムを使用して、次のようにクラスの内容をレポートできます。

```
sdlutil /tmp/VMSTAT_DATA -stats VMSTAT_STATS
```

デフォルトでは、データは 5 分ごとに要約されて記録されます。

extract プログラムのコマンド ラインの引数を使用して、クラスからデータをエクスポートできます。次に例を示します。

```
extract -xp -l /tmp/VMSTAT_DATA -C VMSTAT_STATS -ut -f stdout
```

DSI データをエクスポートするには、ユーザーが root またはログ ファイルの作成者であることが必要です。

単一ファイルの sar データの記録

この例では、データを提供する標準の sar (システム アクティビティ レポート) ユーティリティを使用して、複数の DSI データ収集をセットアップする方法を示します。

システム ユーティリティを使用する場合は、ユーティリティがデータをレポートする方法を正確に把握することが重要です。たとえば、次の 2 つの sar コマンドの違いに注意してください。

```
sar -u 1 1

   HP-UX hpptc99 A.11.00 E 9000/855    04/10/99

   10:53:15    %usr    %sys    %wio   %idle
   10:53:16       2       7       6      85

sar -u 5 2

   HP-UX hpptc99 A.11.00 E 9000/855    04/10/99

   10:53:31    %usr    %sys    %wio   %idle
   10:53:36          4       5       0      91
   10:53:41          0       0       0      99

Average             2       2       0      95
```

上記の例が示すように、1 よりも大きい反復値を指定すると sar はインターバル間の平均値を表示します。この平均値は、対象になるかどうかにかかわらず DSI クラスの仕様ファイルとデータ変換に影響を与えます。異なる UNIX のプラットフォームで実行する場合は sar の出力、もしくはシステムのユーティリティの出力が異なる場合があることに注意してください。DSI クラスの仕様ファイルを作成する前に、使用するユーティリティを十分に認識しておくことが必要です。

最初の例では sar を使用し、その -u オプションを介して CPU の利用を監視します。sar の man ページで示すように、-u オプションはユーザー モードでの実行時間 (%usr)、システム モードでの実行時間 (%sys)、プロセスの一部がブロック I/O を待つアイドル時間 (%wio)、それ以外のアイドル時間 (%idle) をレポートします。長期間にわたって CPU のアクティビティを監視することを目的とするために、平均値を表示しない sar の形式を使用します。

クラス仕様ファイルの作成

最初に、DSI クラスの仕様ファイルを作成します。次の例は、受信データを記述する場合に使用するクラスの仕様を示します。

```
#  sar_u.spec
#
#  sar -u class definition for HP systems.
#
#  ==> 1 minute data; max 24 hours; indexed by hour; roll by day
#

CLASS sar_u = 1000
LABEL "sar -u data"
INDEX BY         hour
MAX INDEXES      24
ROLL BY          day
ACTION "./sar_u_roll $PT_START$ $PT_END$"
RECORDS PER HOUR 60
;

METRICS

hours_1 = 1001
LABEL "Collection Hour"
PRECISION 0;

minutes_1 = 1002
LABEL "Collection Minute"
PRECISION 0;

seconds_1 = 1003
LABEL "Collection Second"
PRECISION 0;

user_cpu = 1004
LABEL "%user"
AVERAGED
MAXIMUM 100
PRECISION 0
;

sys_cpu = 1005
LABEL "%sys"
AVERAGED
MAXIMUM 100
PRECISION 0
;


wait_IO_cpu = 1006
LABEL "%wio"
AVERAGED
MAXIMUM 100
PRECISION 0
;
```

```
idle_cpu = 1007
LABEL "%idle"
AVERAGED
MAXIMUM 100
PRECISION 0
;
```

クラス仕様ファイルのコンパイル

次に、次のコマンドを使用してクラスの仕様ファイルをコンパイルします。

```
sdlcomp sar_u.spec sar_u_log
```

sar -u コマンドの出力は、システムのヘッダー ライン、空白ライン、オプション ヘッダーライン、データ ラインです。これらのラインはタイムスタンプで構成されていて、収集するデータが後に続きます。対象となるのは最終ラインのみです。したがって、sar -u コマンドから出力の最終ラインのみを保存し、そのデータを DSI に送信する仕様が必要です。

dsilog プログラムは stdin からデータを受信することを予測します。ログ処理を開始するために、使用中の処理の出力を dsilog に渡します。ただし、コマンド ラインで使用できるのは 1 つのパイプ (|) のみです。2 つのパイプを使用した場合、UNIX バッファリングにより最初のコマンドの出力が遅延するためです。2 つ目のコマンドが実行されログ ファイルにデータが渡されるまで 8000 文字が書き込まれます。次のような結果が表示されます。

```
sar -u 60 1 | tail -1 | dsilog
```

この動作は有効でないため、DSI の入力ソースとして fifo を使用します。ただし、fifo には独自の問題があります。

次のスクリプトを使用していることを前提とします。

```
#!/bin/ksh                    sar_u_feed

# sar_u_feed script that provides sar -u data to DSI via
# a fifo(sar_u.fifo)

while :                          # (infinite loop)
do

# specify a one minute interval using tail to extract the
# last sar output record(contains the time stamp and data),
# saving the data to a file.

/usr/bin/sar -u 60 1 2>/tmp/dsierr | tail -1 > /usr/tmp/sar_u_data

# Copy the sar data to the fifo that the dsilog process is
# reading.

cat /usr/tmp/sar_u_data > ./sar_u.fifo

done
```

残念なことに、現在ではこのスクリプトから希望する結果は得られません。これは、cat コマンドが fifo をオープンし、データ レコードを書き込んだ後、fifo をクローズするためです。fifo のクローズにより、ログに書き込むデータがないことが dsilog に示されるため、dsilog はこの 1 つのデータ レコードを書き込み終了します。fifo をオープン状態で「保持」するダミーの処理が必要です。そのためダミーの fifo が必要であり、入力のためにダミーの fifo をオープンし、出力のために sar_u.fifo をオープンする処理も必要となります。これにより、sar_u.fifo がオープン状態で保持されるため、dsilog が終了することはありません。

DSI ログ処理の開始

次の手順を踏んで、sar -u データを dsilog に送信します。

1. fifo を 2 つ作成します。1 つはダミーの fifo で、実際の入力 fifo を「オープン状態に保持」するために使用します。

```
# Dummy fifo.
mkfifo ./hold_open.fifo
# Real input fifo for dsilog
mkfifo ./sar_u.fifo
```

2. -i オプションを使用して dsilog を開始し、fifo からの入力を指定します。sar データ フィード (sar_u_feed) を開始する前に、dsilog を開始することが重要です。

```
dsilog ./sar_u_log sar_u \
-i ./sar_u.fifo &
```

3. ダミーの処理を開始し、入力 fifo をオープン状態に保持します。

```
cat ./hold_open.fifo  \
 > ./sar_u.fifo &
```

4. sar のデータ フィードのスクリプト (sar_u_feed) を開始します。

```
./sar_u_feed &
```

5. sar_u_feed スクリプトはデータが抹消される、または fifo をオープン状態に保持する cat が抹消されるまでデータを dsilog に供給します。クラスの仕様ファイルにより sar_u_log (最大 24 時間分) は 1 時間ごとにインデックスが付けられ、翌日の最初に (1 日ごとにロール) スクリプト sar_u_roll が実行されることが通知されます。

```
!/bin/ksh                 sar_u_roll
#
# Save parameters and current date in sar_u_log_roll_file.
# (Example of adding comments/other data to the roll file).

mydate=`date`
echo "$# $0 $1 $2" >> ./sar_u_log_roll_file
echo $mydate       >> ./sar_u_log_roll_file

extract -l ./sar_u_log -C sar_u -B $1 -E $2 -1 -f \
stdout -xp >> ./sar_u_log_roll_file
```

6. ロール スクリプトは ASCII テキスト ファイルにロールアウトされるデータを保存します。この ASCII テキスト ファイルでは、テキスト エディタによる検査やプリンタへの印刷が可能です。

複数ファイルの sar データの記録

CPU の利用のみでなく他のデータも対象とする場合は、そのデータを記述するクラスの仕様ファイルを 1 つ保持する、または各オプションにクラスの仕様ファイルを 1 つ保持して、それらを 1 つのログ ファイル セットにコンパイルできます。次に示す最初の例では、1 つのログ ファイル セットにコンパイルされた個別のクラスの仕様ファイルを示します。

次の例では、CPU の利用、バッファ アクティビティ (sar -b)、システム コール (sar -c) を監視します。この方法でデータを記録する場合は、3 つのクラスの仕様ファイル、3 つの dsilog 処理、3 つの dsilog 入力 fifo、sar データを提供する 3 つのスクリプトが必要です。

複数のクラスの仕様ファイルの作成

各オプションのクラスの仕様ファイルを次に示します。

```
#  sar_u_mc.spec
#
#  sar -u class definition for log files on HP systems.
#
#  ==> 1 minute data; max 24 hours; indexed by hour; roll by day
#

CLASS sar_u = 1000
LABEL "sar -u data"
INDEX BY          hour
MAX INDEXES       24
ROLL BY           day
ACTION "./sar_u_mc_roll $PT_START$ $PT_END$"
RECORDS PER HOUR 60
;

METRICS

hours_1 = 1001
LABEL "Collection Hour"
PRECISION 0
;

minutes_1 = 1002
LABEL "Collection Minute"
PRECISION 0
;

seconds_1 = 1003
LABEL "Collection Second"
PRECISION 0
;
user_cpu = 1004
LABEL "%user"
AVERAGED
MAXIMUM 100
PRECISION 0
;
```

```
sys_cpu = 1005
LABEL "%sys"
AVERAGED
MAXIMUM 100
PRECISION 0
;

wait_IO_cpu = 1006
LABEL "%wio"
AVERAGED
MAXIMUM 100
PRECISION 0
;

idle_cpu = 1007
LABEL "%idle"
AVERAGED
MAXIMUM 100
PRECISION 0
;

#  sar_b_mc.spec
#
#  sar -b class definition for log files on HP systems.
#
#  ==> 1 minute data; max 24 hours; indexed by hour; roll by day
#

CLASS sar_b = 2000
LABEL "sar -b data"
INDEX BY         hour
MAX INDEXES      24
ROLL BY          day
ACTION "./sar_b_mc_roll $PT_START$ $PT_END$"
RECORDS PER HOUR 60
;

METRICS

hours_2 = 2001
LABEL "Collection Hour"
PRECISION 0
;

minutes_2 = 2002
LABEL "Collection Minute"
PRECISION 0
;

seconds_2 = 2003
LABEL "Collection Second"
PRECISION 0
;
```

```
bread_per_sec = 2004

LABEL "bread/s"
PRECISION 0
;

lread_per_sec = 2005
LABEL "lread/s"
PRECISION 0
;

read_cache = 2006
LABEL "%rcache"
MAXIMUM 100
PRECISION 0
;

bwrit_per_sec = 2007
LABEL "bwrit/s"
PRECISION 0
;

lwrit_per_sec = 2008
LABEL "lwrit/s"
PRECISION 0
;

write_cache = 2009
LABEL "%wcache"
MAXIMUM 100
PRECISION 0
;

pread_per_sec = 2010
LABEL "pread/s"
PRECISION 0
;

pwrit_per_sec = 2011
LABEL "pwrit/s"
PRECISION 0
;

#  sar_c_mc.spec
#
#  sar -c class definition for log files on HP systems.
#
#  ==> 1 minute data; max 24 hours; indexed by hour; roll by day
#

CLASS sar_c = 5000
LABEL "sar -c data"
INDEX BY         hour
MAX INDEXES      24
```

```
ROLL BY          day
ACTION "./sar_c_mc_roll $PT_START$ $PT_END$"
RECORDS PER HOUR 60


;

METRICS

hours_5 = 5001
LABEL "Collection Hour"
PRECISION 0
;

minutes_5 = 5002
LABEL "Collection Minute"
PRECISION 0
;

seconds_5 = 5003
LABEL "Collection Second"
PRECISION 0
;

scall_per_sec = 5004
LABEL "scall/s"
PRECISION 0
;

sread_per_sec = 5005
LABEL "sread/s"
PRECISION 0
;

swrit_per_sec = 5006
LABEL "swrit/s"
PRECISION 0
;

fork_per_sec = 5007
LABEL "fork/s"
PRECISION 2
;

exec_per_sec = 5008
LABEL "exec/s"
PRECISION 2
;

rchar_per_sec = 5009
LABEL "rchar"
PRECISION 0
;

wchar_per_sec = 5010
LABEL "wchar/s"
```

```
PRECISION 0
;
```

sar データを提供するために必要な 2 つの追加スクリプトを次に示します。

```
#!/bin/ksh

# sar_b_feed script that provides sar -b data to DSI via
# a fifo (sar_b.fifo)

while :                       # (infinite loop)
do

# specify a one minute interval using tail to extract the
# last sar output record(contains the time stamp and data),
# saving the data to a file.

/usr/bin/sar -b 60 1 2>/tmp/dsierr | tail -1 &> \
/usr/tmp/sar_b_data


# Copy the sar data to the fifo that the dsilog process is reading.

cat /usr/tmp/sar_b_data > ./sar_b.fifo

done

#!/bin/ksh                            sar_c_feed

# sar_c_feed script that provides sar -c data to DSI via
# a fifo(sar_c.fifo)

while :                                 # (infinite loop)
do

# specify a one minute interval using tail to extract the
# last sar output record(contains the time stamp and data),
# saving the data to a file.

/usr/bin/sar -c 60 1 2>/tmp/dsierr | tail -1 > /usr/tmp/sar_c_data


# Copy the sar data to the fifo that the dsilog process is reading.


cat /usr/tmp/sar_c_data > ./sar_c.fifo

done
```

複数のクラスの仕様ファイルのコンパイル

3 つの仕様ファイルを 1 つのログ ファイル セットにコンパイルします。

```
sdlcomp ./sar_u_mc.spec sar_mc_log
sdlcomp ./sar_b_mc.spec sar_mc_log
sdlcomp ./sar_c_mc.spec sar_mc_log
```

DSI ログ処理の開始

sar データについては、再度、次の手順を実行します。

1 fifo を 2 つ作成します。1 つはダミーの fifo で、実際の入力 fifo を「オープン状態に保持」するために使用します。

```
# Dummy fifo.
mkfifo ./hold_open.fifo

# sar -u input fifo for dsilog.
mkfifo ./sar_u.fifo

# sar -b input fifo for dsilog.
mkfifo ./sar_b.fifo

# sar -c input fifo for dsilog.
mkfifo ./sar_c.fifo
```

2 -i オプションを使用して dsilog を開始し、fifo からの入力を指定します。sar データ フィードを開始する前に dsilog を開始することが重要です。

```
dsilog ./sar_mc_log sar_u \
-i ./sar_u.fifo &

dsilog ./sar_mc_log sar_b \
-i ./sar_b.fifo &

dsilog ./sar_mc_log sar_c \
-i ./sar_c.fifo &
```

3 ダミーの処理を開始し、入力 fifo をオープン状態に保持します。

```
cat ./hold_open.fifo \
 > ./sar_u.fifo &

cat ./hold_open.fifo \
 > ./sar_b.fifo &

cat ./hold_open.fifo \
> ./sar_c.fifo &
```

4 sar データ フィードのスクリプトを開始します。

```
./sar_u_feed &

./sar_b_feed &

./sar_c_feed &
```

複数オプションの sar データの記録

sar を使用してデータを DSI に提供するための最後の例では、1 つの仕様ファイルを使用して複数の sar オプション (ubycwavm) からデータを定義しています。

```
#  sar_ubycwavm.spec
#
#  sar -ubycwavm class definition for HP systems.
#
#  ==> 1 minute data; max 24 hours; indexed by hour; roll by day
#

CLASS sar_ubycwavm = 1000
LABEL "sar -ubycwavm data"
INDEX BY         hour
MAX INDEXES      24
ROLL BY          day
ACTION "./sar_ubycwavm_roll $PT_START$ $PT_END$"
RECORDS PER HOUR 60
;

METRICS
hours = 1001
LABEL "Collection Hour"
PRECISION 0;

minutes = 1002
LABEL "Collection Minute"
PRECISION 0;

seconds = 1003
LABEL "Collection Second"
PRECISION 0;

user_cpu = 1004
LABEL "%user"
AVERAGED
MAXIMUM 100
PRECISION 0
;

sys_cpu = 1005
LABEL "%sys"
AVERAGED
MAXIMUM 100
PRECISION 0
;

wait_IO_cpu = 1006
LABEL "%wio"
AVERAGED
MAXIMUM 100
PRECISION 0
;
```

```
idle_cpu = 1007
LABEL "%idle"
AVERAGED
MAXIMUM 100
PRECISION 0
;

bread_per_sec = 1008
LABEL "bread/s"
PRECISION 0
;

lread_per_sec = 1009
LABEL "lread/s"
PRECISION 0
;

read_cache = 1010
LABEL "%rcache"
MAXIMUM 100
PRECISION 0
;

bwrit_per_sec = 1011
LABEL "bwrit/s"
PRECISION 0
;

lwrit_per_sec = 1012
LABEL "lwrit/s"
PRECISION 0
;

write_cache = 1013
LABEL "%wcache"
MAXIMUM 100
PRECISION 0
;
pread_per_sec = 1014
LABEL "pread/s"
PRECISION 0
;

pwrit_per_sec = 1015
LABEL "pwrit/s"
PRECISION 0
;

rawch = 1016
LABEL "rawch/s"
PRECISION 0
;
```

```
canch = 1017
LABEL "canch/s"

PRECISION 0
;

outch = 1018
LABEL "outch/s"
PRECISION 0
;

rcvin = 1019
LABEL "rcvin/s"
PRECISION 0
;

xmtin = 1020
LABEL "xmtin/s"
PRECISION 0
;

mdmin = 1021
LABEL "mdmin/s"
PRECISION 0
;

scall_per_sec = 1022
LABEL "scall/s"
PRECISION 0
;

sread_per_sec = 1023
LABEL "sread/s"
PRECISION 0
;

swrit_per_sec = 1024
LABEL "swrit/s"
PRECISION 0
;

fork_per_sec = 1025
LABEL "fork/s"
PRECISION 2
;

exec_per_sec = 1026
LABEL "exec/s"
PRECISION 2
;

rchar_per_sec = 1027
LABEL "rchar/s"
PRECISION 0
;
```

```
wchar_per_sec = 1028
LABEL "wchar/s"

PRECISION 0
;

swpin = 1029
LABEL "swpin/s"
PRECISION 2
;

bswin = 1030
LABEL "bswin/s"
PRECISION 1
;

swpot = 1031
LABEL "swpot/s"
PRECISION 2
;

bswot = 1032
LABEL "bswot/s"
PRECISION 1
;
blks = 1033
LABEL "pswch/s"
PRECISION 0
;

iget_per_sec = 1034
LABEL "iget/s"
PRECISION 0
;

namei_per_sec = 1035
LABEL "namei/s"
PRECISION 0
;

dirbk_per_sec = 1036
LABEL "dirbk/s"
PRECISION 0
;

num_proc = 1037
LABEL "num proc"
PRECISION 0
;

proc_tbl_size = 1038
LABEL "proc tbl size"
PRECISION 0
;
```

```
proc_ov = 1039
LABEL "proc ov"
PRECISION 0

;

num_inode = 1040
LABEL "num inode"
PRECISION 0
;

inode_tbl_sz = 1041
LABEL "inode tbl sz"
PRECISION 0
;

inode_ov = 1042
LABEL "inode ov"
PRECISION 0
;

num_file = 1043
LABEL "num file"
PRECISION 0
;

file_tbl_sz = 1044
LABEL "file tbl sz"
PRECISION 0
;

file_ov = 1045
LABEL "file ov"
PRECISION 0
;

msg_per_sec = 1046
LABEL "msg/s"
PRECISION 2
;

LABEL "sema/s"
PRECISION 2
;
```

この時点で、生成された出力を検査する必要があります。

```
sar -ubycwavm 1 1:
HP-UX hpptc16 A.09.00 E 9000/855    04/11/95

   12:01:41    %usr    %sys    %wio   %idle
          bread/s lread/s %rcache  bwrit/s  lwrit/s %wcache pread/s
         pwrit/s
         rawch/s canch/s outch/s   cvin/s  xmtin/s  mdmin/s
```

```
          scall/s sread/s swrit/s   fork/s   exec/s  rchar/s wchar/s
          swpin/s bswin/s swpot/s  bswot/s  pswch/s
           iget/s namei/s dirbk/s
          text-sz  ov  proc-sz  ov  inod-sz  ov  file-sz  ov
            msg/s  sema/s

12:01:42    22      48       30      0
              0     342      100     33       81      59   0    0
              0       0      470      0        0       0
            801     127       71      1.00    1.00    975872 272384
           0.00     0.0     0.00    0.0     251
             28     215      107
          N/A   N/A  131/532    0  639/644   0   358/1141   0
          40.00    0.00
```

この出力は、ヘッダーとデータに複数ラインが追加された sar -u 出力に類似しています。再度、tail コマンドを使用してデータ ラインを抽出しますが、このデータ ラインを「1 つの」データ レコードとして dsilog に提供する必要があります。次のスクリプトはデータを取り込み、tr (変換文字) ユーティリティを使用して改行を「削除」します。これは改行が dsilog により入力データの 1 行とみなされるためです。

```
#!/bin/ksh                    Sar_ubycwavm_feed

# Script that provides sar data to DSI via a fifo(sar_data.fifo)

while :                        # (infinite loop)
do

# specify a one minute interval using tail to extract the
# last sar output records (contains the time stamp and data)
# and pipe that data to tr to strip the new lines converting
# the eight lines of output to one line of output.


/usr/bin/sar -ubycwavm 60 1 2>/tmp/dsierr | tail -8 | \
tr "\012" " " > /usr/tmp/sar_data

# Copy the sar data to the fifo that the dsilog process is reading.

cat /usr/tmp/sar_data > ./sar_data.fifo

# Print a newline on the fifo so that DSI knows that this is
# the end of the input record.

print "\012" > ./sar_data.fifo

done
```

順番に処理を実行すると、最初に示した sar -u の例は上記の結果となります。ただし、ログ ファイル セット名、クラス名、fifo 名 (sar_ubycwavm.fifo)、sar データを提供する上記にリストしたスクリプトは除きます。

システム ユーザー数の記録

次の例では、who を使用してシステム ユーザー数を監視します。再度、クラスの仕様ファイルから開始します。

```
#  who_wc.spec
#
#  who word count DSI spec file
#

CLASS who_metrics = 150
LABEL "who wc data"
INDEX BY         hour
MAX INDEXES      120
ROLL BY          hour
RECORDS PER HOUR 60
;

METRICS
who_wc = 151
label "who wc"
averaged
maximum 1000
precision 0
;
```

仕様ファイルをコンパイルして、ログ ファイルを作成します。

```
sdlcomp ./who_wc.spec ./who_wc_log.
```

sar と違って、who ではインターバルや反復値を指定できないため、最小限にインターバル制御を提供するスクリプトを作成します。

```
#!/bin/ksh         who_data_feed

while :
do
    # sleep for one minute (this should correspond with the
    # RECORDS PER HOUR clause in the specification file).

sleep 60

    # Pipe the output of who into wc to count
    # the number of users on the system.

who | wc -l > /usr/tmp/who_data

# copy the data record to the pipe being read by dsilog.

cat /usr/tmp/who_data > ./who.fifo

done
```

データを dsilog に提供するために fifo とスクリプトが必要となるため、再度、次の手順を実行します。

1 fifo を 2 つ作成します。1 つはダミーの fifo で、実際の入力 fifo を「オープン状態に保持」するために使用します。

```
# Dummy fifo.
mkfifo ./hold_open.fifo

# Real input fifo for dsilog.
mkfifo ./who.fifo
```

2 -i オプションを使用して dsilog を開始し、fifo からの入力を指定します。who データフィードを開始する前に、dsilog を開始することが重要です。

```
dsilog ./who_wc_log who_metrics \
-i ./who.fifo &
```

3 ダミーの処理を開始し、入力 fifo をオープン状態に保持します。

```
cat ./hold_open.fifo \
> ./who.fifo &
```

4 who のデータ フィードのスクリプト (who_data_feed) を開始します。

```
./who_data_feed  &
```

17 エラー メッセージ

DSI エラー メッセージには、クラスの仕様、dsilog ログ処理、一般の 3 つのタイプがあります。

- クラスの仕様エラー メッセージのフォーマットは、接頭辞 SDL +「メッセージ番号」です。
- dsilog ログ処理メッセージのフォーマットは、接頭辞 DSILOG +「メッセージ番号」です。
- 一般エラー メッセージは、その他の作業と同様、上記のいずれかにより生成されます。これらのメッセージには、マイナス符号 (-) の接頭辞とメッセージ番号が付きます。

DSI エラー メッセージは、この章に記載されています。SDL エラー メッセージと DSILOG エラー メッセージは、数値順に、エラー状態から回復するための対応方法と共に記載されています。一般エラー メッセージは内容の説明であり、回復するための対応方法は記載されていません。

SDL エラー メッセージ

SDL エラー メッセージは Self Describing Logfile クラス仕様のエラー メッセージで、SDL<*message number*> というフォーマットで表します。

メッセージ SDL1

```
ERROR: Expected equal sign, "=".
```

ここには「=」が必要です。

対応 : 詳細は第 14 章の「クラスの仕様構文」を参照してください。

メッセージ SDL2

```
ERROR: Expected semi-colon, ";".
```

セミコロン (;) はクラスの仕様の終わりと各メトリックの仕様の終わりを示します。セミコロンがあるべき場所に間違った語句やスペルの誤りがある場合も、このメッセージが表示されます。

次に例を示します。

```
class xxxxx = 10
   label "this is a test"
   metric 1000;
```

上記の記述の代わりに次のように入力するとエラー メッセージが表示され、語句「metric」が示されます。

```
class xxxxx = 10
   label "this is a test"
   capacity 1000;
```

対応 : 詳細は第 14 章の「クラスの仕様構文」を参照してください。

メッセージ SDL3

```
ERROR: Precision must be one of {0, 1, 2, 3, 4, 5}
```

精度は、数字を内部で整数に変換するときや、メトリック値の数値表現に戻すときに使用する小数点位置を示す数字を特定します。

対応：詳細は第 14 章の「PRECISION」を参照してください。

メッセージ SDL4

```
ERROR: Expected quoted string.
```

テキストの文字列が必要です。

対応：詳細は、第 14 章の「クラスの仕様構文」を参照してください。

メッセージ SDL5

```
ERROR: Unterminated string.
```

文字列の終わりには二重引用符が必要です。

対応：詳細は第 14 章の「クラスの仕様構文」を参照してください。

メッセージ SDL6

```
NOTE: Time stamp inserted at first metric by default.
```

タイムスタンプ メトリックは、各クラスの最初のメトリックとして自動的に挿入されます。

対応：詳細は第 14 章の「クラスの仕様構文」を参照してください。

メッセージ SDL7

```
ERROR: Expected metric description.
```

メトリック セクションは、最初のメトリック定義の前に METRICS キーワードで開始する必要があります。

対応：詳細は第 14 章の「クラスの仕様構文」を参照してください。

メッセージ SDL8

```
ERROR: Expected data class specification.
```

クラスの仕様のクラス セクションは CLASS キーワードで開始する必要があります。

対応：詳細は第 14 章の「クラスの仕様構文」を参照してください。

メッセージ SDL9

```
ERROR: Expected identifier.
```

メトリックまたはクラスの識別子が必要です。識別子は必ず英字で始まり、英数字とアンダースコアを使用できます。大文字と小文字の区別はありません。

対応：詳細は第 14 章の「クラスの仕様構文」を参照してください。

メッセージ SDL10

```
ERROR: Expected positive integer.
```

数値形式が正しくありません。

対応：正の整数のみを入力してください。

メッセージ SDL13

```
ERROR: Expected specification for maximum number of indexes.
```

クラスの容量を計算するには、インデックスの最大数が必要です。

対応：詳細は第 14 章の「クラスの仕様構文」を参照してください。

メッセージ SDL14

```
ERROR: Syntax Error.
```

入力した構文が間違っています。

対応：構文を確認し、必要に応じて修正します。詳細は、第 14 章の「クラスの仕様構文」を参照してください。

メッセージ SDL15

```
ERROR: Expected metric description.
```

メトリックの記述がありません。

対応：メトリックの記述を入力して、クラスの各データ項目を定義します。詳細は、第 14 章の「クラスの仕様構文」を参照してください。

メッセージ SDL16

```
ERROR: Expected metric type.
```

各メトリックには、*metric_name* と数値の *metric_id* が必要です。

対応：詳細は第 14 章の「メトリックの記述」を参照してください。

メッセージ SDL17

```
ERROR: Time stamp metric attributes may not be changed.
```

timestamp メトリックは、各クラスの最初のメトリックとして自動的に挿入されます。タイムスタンプの位置は変更できます。また、タイムスタンプを取り除き、UNIX タイムスタンプを使用してもかまいません。

対応：詳細は第 14 章の「メトリックの記述」を参照してください。

メッセージ SDL18

```
ERROR: Roll action limited to 199 characters.
```

ROLL BY アクションの上限は 199 文字です。

対応：詳細は第 14 章の「INDEX BY, MAX INDEXES, ROLL BY」を参照してください。

メッセージ SDL19

```
ERROR: Could not open specification file (file).
```

コマンド ライン sdlcomp specification_file では、仕様ファイルをオープンできませんでした。次に示すように、次の行にエラーが続きます。

```
$/usr/perf/bin/sdlcomp /xxx
ERROR: Could not open specification file /xxx.
```

対応：ファイルが読み取り可能であることを確認します。ファイルが読み取り可能な場合は、ファイルの名前と、ファイル名が正しく入力されていることを確認してください。

MessageSDL20

```
ERROR: Metric descriptions not found.
```

メトリックの記述が正しい形式ではありません。

対応：クラス文のメトリック セクションを METRICS キーワードで開始したことを確認します。詳細は、第 14 章の「メトリックの記述」を参照してください。

メッセージ SDL21

```
ERROR: Expected metric name to begin metric description.
```

メトリック名がありません。またはメトリックの記述が正しい形式ではありません。

対応：メトリック名がありません。またはメトリックの記述が正しい形式ではありません。

メッセージ SDL24

```
ERROR: Expected MAX INDEXES specification.
```

INDEX BY を指定するときは、MAX INDEXES の値が必要です。

対応：必要な値を入力します。詳細は、第 14 章の「INDEX BY, MAX INDEXES, ROLL BY」を参照してください。

メッセージ SDL25

```
ERROR: Expected index SPAN specification.
```

INDEX BY の値がありません。

対応：INDEX BY を指定するときは、修飾子を入力してください。詳細は、第 14 章の「INDEX BY, MAX INDEXES, ROLL BY」を参照してください。

メッセージ SDL26

```
ERROR: Minimum must be zero.
```

この数字はゼロ以上でなければなりません。

メッセージ SDL27

```
Expected positive integer.
```

正の値がありません。

対応：正の整数のみを入力してください。

メッセージ SDL29

```
ERROR: Summarization metric does not exist.
```

要約方法として SUMMARIZED BY を使用しましたが、*metric_name* を指定しませんでした。

対応：詳細は第 14 章の「メトリックの記述」を参照してください。

メッセージ SDL30

```
ERROR: Expected 'HOUR', 'DAY', or 'MONTH'.
```

項目の修飾子がありません。

対応 : これらの修飾子のうちの 1 つを入力する必要があります。詳細は、第 14 章の「INDEX BY, MAX INDEXES, ROLL BY」を参照してください。

メッセージ SDL33

```
ERROR: Class id number must be between 1 and 999999.
```

class-id は 6 桁以下の数値でなければなりません。

対応 : クラスについて 6 桁を超えないクラス ID 番号を入力します。詳細は、第 14 章の「クラスの仕様構文」を参照してください。

メッセージ SDL35

```
ERROR: Found more than one index/capacity statement.
```

INDEX BY 文または CAPACITY 文は、各 CLASS セクションに 1 つのみ記述できます。

対応 : 第 14 章の「クラスの仕様構文」のフォーマット制限に従って入力します。

メッセージ SDL36

```
ERROR: Found more than one metric type statement.
```

METRICS キーワードは各メトリックの定義に 1 つのみ記述できます。

対応 : フォーマットの詳細は第 14 章の「メトリックの記述」を参照してください。

メッセージ SDL37

```
ERROR: Found more than one metric maximum statement.
```

MAXIMUM 文は各メトリックの定義に 1 つのみ記述できます。

対応 : 詳細は第 14 章の「メトリックの記述」を参照してください。

メッセージ SDL39

```
ERROR: Found more than one metric summarization specification.
```

要約方法 (TOTALED、AVERAGED、SUMMARIZED BY のいずれか) は、各メトリックの定義に 1 つのみ記述できます。

対応 : 詳細は第 14 章の「要約方法」を参照してください。

メッセージ SDL40

```
ERROR: Found more than one label statement.
```

LABEL は、各メトリックまたはクラスの定義に 1 つのみ記述できます。

対応 : 詳細は第 14 章の「クラスの仕様構文」を参照してください。

メッセージ SDL42

```
ERROR: Found more than one metric precision statement.
```

PRECISION 文は各メトリックの定義に 1 つのみ記述できます。

対応 : 詳細は第 14 章の「PRECISION」を参照してください。

メッセージ SDL44

ERROR: SCALE, MINIMUM, MAXIMUM, (summarization) are inconsistent with text metrics

クラスの仕様構文のこれらの要素は数値メトリック値でのみ有効です。

対応 : 詳細は第 14 章の「クラスの仕様構文」を参照してください。

メッセージ SDL46

ERROR: Inappropriate summarization metric (!).

タイムスタンプ メトリックでは要約できません。

対応 : 詳細は第 14 章の「クラスの仕様構文」を参照してください。

メッセージ SDL47

ERROR: Expected metric name.

各 METRICS 文には *metric_name* が必要です。

対応 : 詳細は第 14 章の「メトリックの記述」を参照してください。

メッセージ SDL47

ERROR: Expected metric name.

各 METRICS 文には *metric_name* が必要です。

対応 : 詳細は第 14 章の「メトリックの記述」を参照してください。

メッセージ SDL48

ERROR: Expected positive integer.

CAPACITY 文には正の整数が必要です。

対応 : 詳細は第 14 章の「CAPACITY」を参照してください。

メッセージ SDL49

ERROR: Expected metric specification statement.

METRICS キーワードが最初のメトリックの定義の前に必要です。

対応 : 詳細は第 14 章の「メトリックの記述」を参照してください。

メッセージ SDL50

Object name too long.

metric_name または *class_name* は 20 文字以下にする必要があります。

対応 : 詳細は第 14 章の「クラスの仕様構文」を参照してください。

メッセージ SDL51

ERROR: Label too long (max 20 chars).

class_label または *metric_label* は 20 文字以下にする必要があります。

対応：詳細は第 14 章の「クラスの仕様構文」を参照してください。

メッセージ SDL53

ERROR: Metric must be between 1 and 999999.

metric_id は 6 桁以下の数字でのみ構成されます。

対応：詳細は第 14 章の「メトリックの記述」を参照してください。

メッセージ SDL54

ERROR: Found more than one collection rate statement.

RECORDS PER HOUR 文は各クラスの記述に 1 つのみ記述できます。

対応：詳細は第 14 章の「RECORDS PER HOUR」を参照してください。

メッセージ SDL55

ERROR: Found more than one roll action statement.

ROLL BY 文は各クラスの仕様に 1 つのみ記述できます。

対応：詳細は第 14 章の「INDEX BY, MAX INDEXES, ROLL BY」を参照してください。

メッセージ SDL56

ERROR: ROLL BY option cannot be specified without INDEX BY option.

ROLL BY 文の前には INDEX BY 文が必要です。

対応：詳細は第 14 章の「INDEX BY, MAX INDEXES, ROLL BY」を参照してください。

メッセージ SDL57

ERROR: ROLL BY must specify time equal to or greater than INDEX BY.

ロール インターバルは、廃棄するデータを識別するためのインデックス インターバルに依存するため、ROLL BY の時間は、INDEX BY の時間と同じかそれより長くなければなりません。

対応：詳細は第 14 章の「INDEX BY, MAX INDEXES, ROLL BY」を参照してください。

メッセージ SDL58

ERROR: Metric cannot be used to summarize itself.

SUMMARIZED BY メトリックは *metric_name* と異なる必要があります。

対応：詳細は第 14 章の「クラスの仕様構文」を参照してください。

メッセージ SDL62

ERROR: Could not open SDL (name).

説明のメッセージがこのエラーの後に続きます。これは、次のようにファイル システム エラーを示すことがあります。

```
$/usr/perf/bin/sdlutil xxxxx – classes
ERROR: Could not open SDL xxxxx.
ERROR: Could not open log file set.
```

または、次のように内部エラーを示すことがあります。

```
$/usr/perf/bin/sdlutil xxxxx – classes
ERROR: Could not open SDL xxxxx.
ERROR: File is not SDL root file or the
description file is not accessible.
```

このエラーは、ログ ファイルが移動された場合にも表示されることがあります。パス名情報は DSI ログ ファイルに保存されているため、ログ ファイルは別のディレクトリに移動できません。

対応 : 上記の説明または後に続くメッセージが明確な問題を示さない場合、sdlutil を使用してログ ファイル セットを削除して、再度生成してください。

メッセージ SDL63

```
ERROR: Some files in log file set (name) are missing.
```

ログ ファイル セットを構成するファイルのリストを調査しましたが、操作の成功に必要な 1 つ以上のファイルがありませんでした。

対応 : 現在の状況を正確に把握できない場合、sdlutil を使用してログ ファイル セットを削除し、最初からやり直すことをお勧めします。

メッセージ SDL66

```
ERROR: Could not open class (name).
```

説明のメッセージが後に続きます。

対応 : 問題が明確でない場合、sdlutil を使用してログ ファイル セットを削除し、最初からやり直してください。

メッセージ SDL67

```
ERROR: Add class failure.
```

説明のメッセージが後に続きます。

コンパイラが新しいクラスをログ ファイル セットに追加できませんでした。

対応 : ログ ファイル セット内の適正なすべてのクラスにアクセスできる場合、新しいログ ファイル セットまたは別のログ ファイル セットを指定します。ログ ファイル セット内の適正なすべてのクラスにアクセスできない場合、sdlutil を使用してログ ファイル セットを削除し、最初からやり直してください。

メッセージ SDL72

```
ERROR: Could not open export files (name).
```

エクスポートされたデータが書き込まれるファイルをオープンできませんでした。

対応 : エクスポート ファイル パスが存在することと、保持しているアクセス権限を確認してください。

メッセージ SDL73

```
ERROR: Could not remove shared memory ID (name).
```

説明のメッセージが後に続きます。

対応 : 共有メモリ ID を削除するには、ログ ファイル セットの作成者またはルート ユーザーである必要があります。UNIX コマンド `ipcrm -m` *id* を使用して、共有メモリ ID を削除してください。

メッセージ SDL74

`ERROR: Not all files could be removed.`

ログ ファイル セット内のすべてのファイルを削除できませんでした。

説明のメッセージが後に続きます。

対応 : 次のように入力して、ファイルと共有メモリ ID をリストします。

```
sdlutil (logfile set) -files
sdlutil (logfile set) -id
```

ファイルを削除するには、UNIX コマンド **rm** *filename* を使用します。共有メモリ ID を削除するには UNIX コマンド **ipcrm -m** *id* を使用します。ログ ファイル セットがクローズされていて 、共有メモリ ID が `sdlutil` によって適切に削除されなかった場合、共有メモリ ID は存在するだけであり、削除する必要があることに注意してください。

メッセージ SDL80

`ERROR: Summarization metric (metric) not found in class.`

`SUMMARIZED BY` メトリックは `METRIC` のセクションでは定義されていません。

対応 : 詳細は第 14 章の「メトリックの記述」を参照してください。

メッセージ SDL81

`ERROR: Metric id (id) already defined in SDL.`

metric_id は一度だけ定義する必要があります。別のクラスですでに定義されているメトリックの定義を再利用するには、*metric_id* やその他の仕様を指定せずに *metric_name* のみを指定します。

対応 : 詳細は第 14 章の「METRICS」を参照してください。

メッセージ SDL82

`ERROR: Metric name (name) already defined in SDL.`

metric_name は一度だけ定義する必要があります。別のクラスですでに定義されているメトリックの定義を再利用するには、*metric_id* やその他の仕様を指定せずに *metric_name* のみを指定します。

対応 : 詳細は第 14 章の「METRICS」を参照してください。

メッセージ SDL83

`ERROR: Class id (id) already defined in SDL.`

class_id は一度だけ定義する必要があります。スペルを確認して、正確に入力したことを確かめてください。

対応 : 詳細は、第 14 章の「クラスの仕様構文」を参照してください。

メッセージ SDL84

```
ERROR: Class name (name) already defined in SDL.
```

class_name は一度だけ定義する必要があります。スペルを確認して、正確に入力したことを確かめてください。

対応： 詳細は、第 14 章の「クラスの仕様構文」を参照してください。

メッセージ SDL85

```
ERROR: Must specify class to de-compile.
```

-decomp を使用するときは *class list* を指定する必要があります。

対応：詳細は第 15 章の「sdlutil によるデータの管理」を参照してください。

メッセージ SDL87

```
ERROR: You must specify maximum number of classes with -maxclass.
```

-maxclass オプションを使用するときは、新しいログ ファイル セットの作成時に提示するクラスの最大数を指定する必要があります。

対応： 詳細は、第 15 章の「sdlcomp コンパイラ」を参照してください。

メッセージ SDL88

```
ERROR: Option \"!\" is not valid.
```

コマンド ラインの入力が無効です。

対応：入力内容を確認して、正しい構文に従っていることを確認してください。

メッセージ SDL89

```
ERROR: Maximum number of classes (!) for -maxclass is not valid.
```

-maxclass の数字はゼロより大きくなければなりません。

対応：詳細は第 15 章の「sdlcomp コンパイラ」を参照してください。

メッセージ SDL90

```
ERROR: -f option but no result file specified.
```

-f オプションを使用するときは、フォーマット ファイルを指定する必要があります。

対応：-f オプションを使用するときは、フォーマット ファイルを指定する必要があります。

メッセージ SDL91

```
ERROR: No specification file named.
```

クラスの仕様ファイルに名前が指定されていません。

対応：sdlcomp を使用するときは、*specification_file* を入力する必要があります。詳細は、第 15 章の「sdlcomp コンパイラ」を参照してください。

メッセージ SDL92

```
ERROR: No log file set named.
```

sdlcomp を使用するときは、*logfile_set* を入力する必要があります。

対応 : 詳細は第 15 章の「sdlcomp コンパイラ」を参照してください。

メッセージ SDL93

`ERROR: Metric ID already defined in class.`

metric_id は一度だけ定義する必要があります。

対応 : 別のクラスですでに定義されているメトリックの定義を再利用するには、*metric_id* やその他の仕様を指定せずに *metric_name* のみを指定します。

詳細は、第 14 章の「メトリックの記述」を参照してください。

メッセージ SDL94

`ERROR: Metric name already defined in class.`

metric-name は、一度だけ定義する必要があります。

対応 : 別のクラスですでに定義されているメトリックの定義を再利用するには、*metric_name* やその他の仕様を指定せずに *metric_id* のみを指定します。詳細は、第 14 章の「メトリックの記述」を参照してください。

メッセージ SDL95

`ERROR: Text found after complete class specification.`

sdlcomp コンパイラがクラスの仕様の一部として認識されないテキストを見つけました。

対応 : 仕様を再度入力して、もう一度実行してください。

メッセージ SDL96

`ERROR: Collection rate statement not valid.`

正しいフォーマットは `RECORDS PER HOUR (` *数字*`)` です。キーワードはこの順序で示す必要があり、省略できません。

対応 : 必要なフォーマットに従って、キーワードを修正します。

メッセージ SDL97

`ERROR: Expecting integer between 1 and 2,147,483,647.`

この範囲の数字を使用する必要があります。

対応 : 範囲内の数字を入力します。

メッセージ SDL98

`ERROR: Action requires preceding ROLL BY statement.`

入力内容が適切ではありません。またはクラスの仕様ファイルにありません。

対応 : ACTION はログ ファイルがロールするときの動作を指定します。ロールする必要があるときをまず把握することが重要です。`ROLL BY` が `ACTION` の前に必要です。

次に例を示します。

```
class xxxxx = 10
   index by month max indexes 12
   action "ll *";
```

以上の記述は、次のようになります。

```
class xxxxx = 10
   index by month max indexes 12
   roll by month
   action "ll *";
```

メッセージ SDL99

ERROR: MAX INDEXES requires preceeding INDEX BY statement.

入力内容が適切ではありません。またはクラスの仕様ファイルにありません。

対応：インデックスの最大数を指定するには、INDEX BY を記述している場所をプログラムが認識する必要があります。INDEX BY 文は MAX INDEXES の前に必要です。

次に例を示します。

```
class xxxxx = 10
   max indexes 12
   label "this is a test";
```

以上の記述は、次のようになります。

```
class xxxxx = 10
   index by month
   max indexes 12
   label "this is a test";
```

メッセージ SDL100

WARNING: CAPACITY UNLIMITED not implemented, derived value used.(SDL-100)

メッセージ SDL101

ERROR: Derived capacity too large.(SDL-101)

メッセージ SDL102

ERROR: Text Length should not exceed 4096.

テキスト メトリックの長さは 4096 文字以内にする必要があります。

メッセージ SDL103

ERROR: RECORDS PER HOUR should not be greater than 3600 for logging summarized data.

対応：RECORDS PER HOUR は要約されていないデータでのみ 3600 文字以上にすることができます。コンパイルするには、-u オプションを使用します。

DSILOG エラー メッセージ

DSILOG エラー メッセージは dsilog ログ処理メッセージで、

DSILOG<*メッセージ番号*> というフォーマットで表します。

メッセージ DSILOG1

ERROR: Self describing log file not specified.

対応 : コマンド ラインを修正して、再度実行してください。

メッセージ DSILOG2

ERROR: Data class name not specified.

データ クラスは dsilog に渡される 2 番目のパラメータである必要があります。

対応 : コマンド ラインを修正して、再度実行してください。

メッセージ DSILOG3

ERROR: Could not open data input file (name).

コマンド ラインで指定したファイルをオープンできませんでした。エラー メッセージの次の行に UNIX ファイル システム エラーが表示されます。

メッセージ DSILOG4

ERROR: OpenClass (\"name\") failed.

指定したクラスをオープンできませんでした。指定したクラスは指定されたログ ファイル セットにありません。またはそのクラスのデータ ファイルにアクセスできません。

対応 : 説明のメッセージに続いて、内部エラーの説明またはファイル システム エラーのいずれかが表示されます。

メッセージ DSILOG5

ERROR: Open of root log file (name) failed.

ログ ファイル セットのルート ファイルがオープンできませんでした。その理由が説明のメッセージに表示されます。

メッセージ DSILOG6

ERROR: Time stamp not defined in data class.

クラスが生成されましたが、タイムスタンプが含まれていませんでした。

対応 : sdlutil を使用してログ ファイル セットを削除し、最初からやり直してください。

メッセージ DSILOG7

ERROR: (Internal error) AddPoint ( ) failed.

dsilog がレコードをデータ ファイルに書き込もうとしましたが、できませんでした。説明のメッセージが後に続きます。

メッセージ DSILOG8

```
ERROR: Invalid command line parameter (name).
```

示されているパラメータが有効なコマンド ライン オプションとして認識されなかったか、コマンド ライン内の所定の位置にありません。

対応 : コマンド ライン パラメータを修正して、再度実行してください。

メッセージ DSILOG9

```
ERROR: Could not open format file (name).
```

受信メトリックとデータ クラス内の受信メトリックの照合を指示するファイルが検出されないか、またはそのファイルにアクセスできませんでした。UNIX ファイル システム エラーと共に説明のメッセージが後に続きます。

対応 : クラスの仕様ファイルがあることを確認します。

メッセージ DSILOG10

```
ERROR: Illegal metric name (name).
```

フォーマット ファイルに最大サイズより長いメトリック名が含まれていたか、メトリック名がメトリック名構文に従っていませんでした。

対応 : クラスの仕様のメトリック名を修正して、再度 dsilog を実行します。

メッセージ DSILOG11

```
ERROR: Too many input metrics defined.Max 100.
```

フォーマット ファイルで指定できるのは、最大 100 メトリックです。

対応 : dsilog への入力を外部的に再フォーマットするか、データ ソースを 2 つ以上のデータ ソースに分割してください。

メッセージ DSILOG12

```
ERROR: Could not find metric (name) in class.
```

フォーマット ファイルにあるメトリック名がデータ クラス内にありませんでした。

対応 : 訂正して、もう一度実行してください。

メッセージ DSILOG13

```
ERROR: Required time stamp not found in input specification.
```

-timestamp コマンド ライン オプションが使用されましたが、フォーマット ファイルはタイムスタンプを受信データ内で検出する場所を指定していませんでした。

対応 : タイムスタンプを検出する場所を指定してください。

メッセージ DSILOG14

```
ERROR: (number) errors, collection aborted.
```

収集の設定時に重大なエラーが検出されました。

対応 : エラーを訂正してもう一度実行してください。また、-vi と -vo の各オプションを使用して、データの受信時および記録時にデータを検証できます。

メッセージ DSILOG15

```
ERROR: Self describing log file and data class not specified.
```

コマンド ラインで、データを記録するログ ファイル セットとデータ クラスを指定する必要があります。

対応 : コマンド ラインの入力を修正して、再度実行してください。

メッセージ DSILOG16

```
ERROR: Self describing log file set root file (name) could not be accessed.
error=(number).
```

ログ ファイル セットのルート ファイルをオープンできませんでした。

対応 : 後に続く、障害に関する説明のメッセージを確認してください。

メッセージ (番号なし)

```
Metric null has invalid data
Ignore to end of line, metric value exceeds maximum
```

この警告メッセージは、dsilog が入力の特定行のデータを記録しない場合に発生します。これは、入力に空白行またはヘッダー 行が含まれる場合や、メトリック値が指定された精度を超えている場合など、DSI ログ ファイルが予期するフォーマットに入力が適合しないときに発生します。この場合、誤りのある行はスキップされます (記録されません)。dsilog は、次の有効な入力行のデータの記録を再開始します。

メッセージ DSILOG17

```
ERROR: Logfile set is created to log unsummarized data, could not log
summarized data.
```

対応 : コンパイル中に -u オプションを使用してログ ファイルのセットが作成された場合、dsilog を使用してログを記録するには -s 0 オプションを使用します。このオプションを使用すると、記録されたデータは要約されないことを示します。

一般エラー メッセージ

エラー	説明
-3	max-class で許可されている数より多くのクラスを追加しようとしています。
-5	クラス データを含むファイルをオープンできませんでした。
-6	ファイルを読み取れませんでした。
-7	ファイルに書き込めませんでした。
-9	書き込みアクセスが要求されていないときに、ログ ファイルへの書き込みを行おうとしています。
-11	クラスへのポインタが見つかりませんでした。
-13	ファイルまたはデータ構造が初期化されていません。
-14	クラスの記述ファイルを読み取れませんでした。
-15	クラスの記述ファイルに書き込めませんでした。
-16	クラスの定義に必要なすべてのメトリックが、メトリックの記述クラスにありませんでした。
-17	ログ ファイル セット内のファイルのパス名の長さが、1024 文字を超えています。
-18	クラス名の長さが、20 文字を超えています。
-19	ファイルが、ログ ファイル セットのルート ファイルではありません。
-20	ファイルが、lod ファイル セットの一部ではありません。
-21	現在のソフトウェアは、ログ ファイル セットにアクセスできません。
-22	共有メモリ セグメントまたは ID が得られませんでした。
-23	共有メモリ セグメントに結び付けられませんでした。
-24	ログ ファイル セットをオープンできません。
-25	現在の作業ディレクトリを特定できませんでした。
-26	クラス データ ファイルからクラス ヘッダーを読み取れませんでした。
-27	ログ ファイル セット内のファイルをオープンできませんでした。
-28	データ クラスをオープンできませんでした。
-29	Lseek を実行できませんでした。
-30	ログ ファイルから読み取れませんでした。
-31	ログ ファイルに書き込めませんでした。
-32	削除できませんでした。

エラー	説明
-33	shmctl (REM_ID) を実行できませんでした。
-34	ログ ファイル セットが不完全です。ルート ファイルまたは記述ファイルがありません。
-35	クラスを追加するターゲット ログ ファイルが、現在のログ ファイル セットにありません。

18 トランザクション追跡の紹介

この章では、次の内容について説明します。

- パフォーマンス管理の改善
- リアルタイムでの注文処理のシナリオ
- トランザクション データの監視

パフォーマンス管理の改善

HP Operations エージェント および HP GlancePlus のトランザクション追跡機能によって、システム性能を管理する能力を改善できます。

分散した重要度の高いビジネス アプリケーションの数が増加するにつれて、アプリケーション マネージャおよびシステム マネージャはそれらの分散した情報技術 (IT: Information Technology) がどのように実行されているかを知るために、次のようなより多くの情報を必要とします。

- アプリケーションは応答を停止しているのか。
- アプリケーション応答時間は問題にならないか。
- サービス レベルの目標 (SLO) を達成しているか。

Performance Collection Component および GlancePlus のトランザクション追跡機能は、IT マネージャが、ビジネス トランザクションを遂行するクライアント / サーバー IT 環境の終端間を管理しやすいように構築することを可能にします。Performance Collection Component により、ビジネス トランザクションがどのようなものかを知り、ビジネスの遂行に重要なトランザクション データを定義し、収集できます。

標準的なアプリケーション応答測定 (ARM: Application Response Measurement) API コールがアプリケーションに組み込まれている場合、これらの製品はマルチベンダ プラットフォーム間での広範囲な処理内容の追跡および終端間の管理機能を提供します。

トランザクション追跡の利点

- トランザクションの開始から終了までの経過時間をクライアントの視点で提供します。
- トランザクション データを提供します。
- サービス レベル協定 (SLA: Service Level Agreement) の管理を支援します。

これらの内容は、この後で詳細に説明します。

ユーザーの視点から見たトランザクション時間

トランザクション追跡は、トランザクションの開始から終了までの経過時間をエンド ユーザーの視点で提供します。IT 環境でトランザクション追跡を使用すると、次の利点があります。

- 各トランザクションの実行回数を正確に追跡できます。
- 現在のような概算の時間ではなく、トランザクションが完了するまでの詳細な所要時間がわかります。
- トランザクション時間とシステム リソースの利用を関連させることができます。
- 容量立案、パフォーマンス管理、アカウンティング、課金などのシステム管理アプリケーションのデータを利用できます。
- システムとネットワーク リソースの抽象的な定義によって実際の作業を表すのではなく、具体的な作業単位 (トランザクション) に基づいた、アプリケーションの最適化と詳細なパフォーマンス トラブルシューティングが可能になります。

トランザクション データ

各ビジネス トランザクションの開始と終了をマークするために、アプリケーション応答測定 (ARM) API コールをアプリケーションに挿入すると、次のリソースおよびパフォーマンスの監視ツールを使用して、トランザクション データを監視できます。

- Performance Collection Component は、トランザクション データのアラームの検知、記録、報告に必要な登録機能を提供します。トランザクション データは Performance Manager、Glance で表示することも、Performance Collection Component のログ ファイルから、スプレッドシートやレポーティング ツールによりアクセスできるファイルにデータをエクスポートして表示することもできます。
- Performance Manager によりグラフ化されたパフォーマンス データは、短期的なトラブルシュートや、長期的な傾向や分析などの調査に使用します。
- Glance は、システムおよびトランザクションの状態を監視するための詳細なリアルタイム データを逐次表示します。
- Performance Manager、Glance または HP Operations Manager のメッセージ ブラウザは、サービス レベルのコンプライアンス関するアラームを監視します。

各トランザクションのメトリックについては、第 22 章の「トランザクション メトリック」を参照してください。

サービス レベルの目標

サービス レベルの目標 (SLO: Service Level Objective) は、ビジネス アプリケーションのユーザーから要求される規定されたサービス レベルから算出されます。SLO は、一般的にサービス レベル協定 (SLA: Service Level Agreement) の作成に基づくものです。SLO からは、ビジネス アプリケーション ユーザーに対して合意したサービス レベルが満たされているかどうかを判断するために、IT リソース マネージャが収集、監視、保存、報告する必要がある実際のメトリックが作成されます。

SLO は、簡単なトランザクションの応答時間を監視するような単純な場合もありますが、システムの可用性の追跡のように複雑な場合もあります。

リアルタイムでの注文処理のシナリオ

大勢の電話オペレータが、さまざまな商品の注文を視聴者から受けるテレビ ショッピングを想像します。この企業は、注文の情報を入力し、商品の納期をチェックし、在庫のインベントリを更新するために、コンピュータ プログラムを使用すると仮定します。この架空の企業を例に、顧客との契約および SLO を達成するために、トランザクション追跡がどのように役立つかを説明します。

リソース マネージャは、重要な作業、顧客満足度の要因、生産性の要因、最大応答時間に基づき、顧客に提供するサービス レベルを決定できます。

第 23 章の「トランザクション追跡の例」には、トランザクション データを Performance Collection Component および Glance で監視できるように、ARM API コールを注文処理アプリケーションに挿入する方法の擬似コード例が示されています。

リアルタイム注文処理の必要事項

上記のリアルタイム注文処理例において SLO を達成するために、リソース マネージャは次の重要な作業を完了するために必要な時間を監視する必要があります。

- 注文情報の入力
- 商品の納期の問い合わせ
- 在庫のインベントリの更新

ここで、顧客にとっての重要な顧客満足の要素は、オペレータが注文をどれくらい迅速に処理できるかです。

企業にとっての重要な生産性の要素は、オペレータが 1 時間に処理できる注文の数です。

顧客満足および生産性の目標を達成するには、設定した SLO のコンプライアンスのために、インベントリ データベースへのアクセス、インベントリの調整、レコードの書き戻しの各トランザクションの応答時間を監視する必要があります。たとえば、リソース マネージャがトランザクションの 90 パーセントが 5 秒以内で終了するように、このアプリケーションの SLO を設定している場合などです。

注文処理アプリケーションの準備

インベントリの応答および**インベントリの更新**のトランザクションを作成するために、ARM API コールを注文処理アプリケーションに挿入できます。アプリケーション プログラマーは、アプリケーションをコンパイルをする前に、ARM API コールをアプリケーションに挿入する必要があります。さまざまなトランザクション監視を定義する ARM API コールを含む注文処理プログラムの例 (擬似コードに記載) は、第 23 章の「トランザクション追跡の例」を参照してください。

ARM API コールのアプリケーションへの組み込みについての情報は、『*アプリケーション応答測定 2.0 API ガイド*』を参照してください。

トランザクション データの監視

ARM API コールが組み込まれているアプリケーションをシステムにインストールして実行すると、Performance Collection Component、GlancePlus、または Performance Manager でトランザクション データを監視できます。

Performance Collection Component での使用

Performance Collection Component を使用すると、指定したトランザクションのデータを収集および記録し、経時的に SLO の傾向を監視し、SLO を超過したときにアラームを生成できます。これらの傾向が特定されたら、トランザクションの量に基づいて IT コストを割り当てることができます。Performance Collection Component のアラームは、技術者のページャーに通知するように構成できるため、問題を直ちに調査して解決することができます。詳細は、第 24 章の「高度な機能」を参照してください。

Performance Manager でトランザクション データを表示するには、Performance Collection Component が必要です。

Performance Manager での使用

Performance Manager は Performance Collection Component からアラームとトランザクション データを受信します。たとえば、注文処理アプリケーションによる在庫確認に時間がかかりすぎる場合、Performance Manager はアラームを受信し、潜在的な問題として警告をリソース マネージャのコンソールに送信するように Performance Collection Component を構成できます。

Performance Manager で、データ ソースの [Class List (クラス リスト)] ウィンドウから **TRANSACTION** を選び、さまざまなトランザクションの**トランザクション メトリックをグラフ化**できます。詳細は、Performance Manager のオンライン ヘルプを参照してください。

GlancePlus の使用

GlancePlus を使用すると、最新のトランザクション応答時間と、設定した SLO 内でトランザクションが実行されているかどうかを監視できます。GlancePlus は、トランザクションのパフォーマンスに影響を与えている可能性のあるリソースのボトルネックの特定および解決に役立ちます。詳細は、GlancePlus ヘルプ メニューから利用できる GlancePlus オンライン ヘルプを参照してください。

ARM の使用のガイドライン

ARM API をアプリケーションに組み込む場合、いくつかの注意点があります。さらに、ARM データ収集の特徴と制限を理解していれば、ARM の機能が組み込まれたアプリケーションの環境をより簡単に管理できます。これらを十分に理解していない場合に混乱が生じることのある点を次に示します。

1. ARM メトリックを取得するには、`ttd` と `midaemon` が動作している必要があります。Performance Collection Component の場合、ARM メトリックを記録するには `scope` コレクタが動作している必要があります。`ovpa start` スクリプトは、必要なプロセスをすべて起動します。同様に Glance は、`ttd` と `midaemon` が開始していない場合、それらを開始します（第 19 章の「トランザクション追跡デーモン (ttd)」を参照）。
2. 新しく定義したトランザクション名を取得するには、トランザクション設定ファイル `ttd.conf` を再度読み込みます（第 19 章の「トランザクション構成ファイル (ttd.conf)」を参照）。
3. 新規または変更されたトランザクションの範囲とサービス レベルの目標 (SLO) を取得するには、Performance Collection Component、ユーザー アプリケーション、`ttd` を再起動する必要があります（第 19 章の「新しいアプリケーションの追加」を参照）。
4. Performance Collection Component では、ユーザー定義メトリック内の文字列は無視されます。最初の 6 つの非文字列ユーザー定義メトリックのみが記録されます（第 24 章の「データ型の使用方法」を参照）。
5. トランザクションのアラーム状態を指定する場合、トランザクション名へのダッシュ (-) の使用には制限があります（第 20 章の「アラーム」のセクションの「Performance Collection Component の使用」を参照）。
6. Performance Collection Component は、アプリケーション名およびトランザクション名の最初の 60 文字のみを表示します（第 19 章の「アプリケーション名とトランザクション名の指定」を参照）。
7. 組み込まれる固有のトランザクション名の数を制限します（第 20 章の「固有のトランザクションの制限」を参照）。
8. エンドユーザーから見て、アプリケーションの実行に影響を与えるような ARM API 関数コールを許可しません（第 19 章の「ARM API コール ステータス リターン」を参照）。
9. リンクには、共有ライブラリを使用します（381 ページの「プラットフォームごとの C コンパイル オプション例」のセクションを参照）。

19 トランザクション追跡の動作

次に示す Performance Collection Component と GlancePlus のコンポーネントを一緒に使用すると、アプリケーション応答測定 (ARM: Application Response Measurement) コールが組み込まれたアプリケーションからのトランザクション データの定義や追跡に役立ちます。

- 測定インターフェイス デーモン (midaemon) は、Performance Collection Component、Performance Manager、GlancePlus によってアクセスおよびレポートされる情報のある共有メモリー セグメントで、トランザクション データを監視してレポートするデーモン プロセスです。また、HP-UX システムでは、midaemon はシステム パフォーマンス データも監視しています。
- 構成ファイル (/var/opt/perf/ttd.conf) は、ユーザーがトランザクションを定義し、各トランザクションで監視する情報を識別するために使用されます。
- トランザクション追跡デーモン (ttd) は、midaemon と共に、トランザクション構成ファイル (ttd.conf) からのトランザクション定義の読み込み、登録、同期を実行します。

ARM 2.0 のサポート

ARM 2.0 は、以前の ARM バージョンのスーパーセットです。ARM 2.0 で提供される新しい機能は、ユーザー定義メトリック、トランザクション相関、ロギング エージェントです。Performance Collection Component および GlancePlus では、ユーザー定義メトリック、トランザクション相関はサポートされていますが、ロギング エージェントはサポートされていません。

ただし、アプリケーションへの組み込みをテストするために、ロギング エージェントを使用することはできます。ロギング エージェントのソース コード (`logagent.c`) は ARM 2.0 Software Developers Kit (SDK) に含まれています。これは、次の Web サイトから入手できます。

http://regions.cmg.org/regions/cmgarmw

ロギング エージェントの使用に関する情報は、『*アプリケーション応答測定 2.0 API ガイド*』を参照してください。

『*アプリケーション応答測定 2.0 API ガイド*』では「ユーザー定義メトリック」ではなく、「アプリケーション定義メトリック」という用語で説明されています。

ARM API コールのサポート

次に示す ARM API コールは、Performance Collection Component および GlancePlus でサポートされています。

`arm_init()`	アプリケーションと (オプションとして) ユーザーを指定して登録します。
`arm_getid()`	トランザクション クラスを指定して登録し、関連するトランザクション情報を提供します。ユーザー定義メトリックのコンテキストを定義します。
`arm_start()`	固有のトランザクション インスタンスの開始を通知します。
`arm_update()`	固有のトランザクション インスタンスの値を更新します。
`arm_stop()`	固有のトランザクション インスタンスの終了を送信します。
`arm_end()`	アプリケーションの終了を送信します。

ARM API コールのアプリケーションへの組み込みやコールとパラメータの詳細については、現在の『*アプリケーション応答測定 2.0 API ガイド*』および *arm (3)* のマニュアル ページを参照してください。業務用アプリケーションでは、ARM API コールがアプリケーションにすでに 組み込まれているかどうかを、その製品ドキュメントで確認してください。

必要なライブラリに関する重要な情報は、このマニュアルの後ろの 375 ページの「トランザクション ライブラリ」を参照してください。

arm_complete_transaction コール

HP の ARM エージェントは、ARM 2.0 API の標準の他に、arm_complete_transaction コールもサポートしています。このコールは ARM 標準への HP 特有の拡張であり、トランザクションの開始が arm_start コールで区切られなかった場合に、完了したトランザクションの終了をマークするために使用できます。arm_complete_transaction コールは、完了したトランザクション インスタンスの応答時間をパラメータとして使用します。

トランザクション インスタンスの終了を通知する他に、オプションのデータバッファには、トランザクションに関する追加情報を含めることができます。このコールとパラメータについての詳細情報に加え、このオプション データの情報については、*arm (3)* のマニュアル ページを参照してください。

ARM 組み込みアプリケーションの例

組み込まれた ARM API コールの例については、/< インストール ディレクトリ >/examples/arm/ ディレクトリにある armsample1.c、armsample2.c、armsample3.c、armsample4.c の ARM 組み込みアプリケーション、およびそのビルド スクリプト (Make.armsample) を参照してください。

- armsample1.c は簡単な標準 ARM API コールの使用例を示しています。
- armsample2.c も、簡単な標準 ARM API コール の使用例を示しています。これは、armsample1.c に似た構造を持ちますが、双方向の特性があります。
- armsample3.c は、ARM API のバージョン 2.0 で提供された、ユーザー定義メトリックとトランザクション相関係数を使用する方法の例です。この例は、サーバーとクライアントの両方がいくつかのトランザクションを実行するクライアント / サーバー アプリケーションをシミュレーションします (通常は、アプリケーション クライアントとサーバー コンポーネントは別々のプログラムに存在しますが、分かりやすくするために同じ場所に置かれています)。

  クライアント プロシージャはトランザクションを開始し、arm_start コールから ARM 相関係数を要求します。この相関係数はクライアントによって保存され、サーバーが arm_start をコールしたときに使用できるように、サーバーに渡されます。サーバーで実行されているパフォーマンス ツールは、この相関係数情報を使用して、このサーバーを使用している異なるクライアントを識別できます。

  また、このプログラムには、ユーザー定義のメトリック値を ARM API に渡すメカニズムが含まれています。これにより、パフォーマンス ツールの応答時間やサービス レベル情報が分かるだけではなく、アプリケーション自体にとって重要なデータを参照することもできます。たとえば、あるトランザクションが異なるサイズの要求を処理していて、その要求のサイズをユーザー定義のメトリックで設定できる場合もあります。応答時間値が高い場合、長い応答時間が大きいサイズのトランザクション インスタンスに関係するかどうかを確認するために、このユーザー定義メトリックを使用できます。
- armsample4.c は、ARM コールのユーザー定義メトリックの使用例を提供します。arm_start、arm_update、arm_stop の各コールを使用して、異なるメトリック値を渡すことができます。または、arm_complete_transaction は、tran が start/stop コールで区切られない場合に使用できます。

アプリケーション名とトランザクション名の指定

ARM の `arm_init` と `arm_getid` の API コール内のアプリケーション名とトランザクション名はそれぞれ最大 128 文字ですが、Performance Collection Component には最大 60 文字のみが表示されます。最初の 60 文字を超えた文字は表示されません。ただし、GlancePlus では 128 文字まで表示されます。

Performance Collection Component には、アプリケーション名およびトランザクション名が抽出またはエクスポートされたトランザクション データ内の表示方法に制限があります。これらの規則は、Performance Manager でアプリケーション名およびトランザクション名を表示するときにも適用されます。

アプリケーション名はトランザクション名より常に先行されます。たとえば、65 文字のアプリケーション名、および 40 文字のトランザクション名を持つトランザクション データをエクスポートすると、アプリケーション名のみが表示されます。アプリケーション名の最後の 5 文字は表示されません。

別の例としては、アプリケーション名が 32 文字で、トランザクション名が 40 文字の場合、Performance Collection Component はアプリケーション名をすべて表示しますが、トランザクション名は切り詰められ、合計で 60 文字が表示されます。アプリケーション名およびトランザクション名に 59 文字が割り当てられ、2 つの名前を区分する下線 (_) に 1 文字が割り当てられます。アプリケーション名 「`WarehouseInventoryApplication`」およびトランザクション名「`CallFromWestCoastElectronicSupplier`」が Performance Collection Component または Performance Manager に表示される方法を次に示します。

```
WarehouseInventoryApplication_CallFromWestCoastElectronicSup
```

Performance Manager でデータを表示する場合、アプリケーション名およびトランザクション名の 60 文字の組合せは固有である必要があります。

トランザクション追跡デーモン (ttd)

トランザクション追跡デーモン (ttd) は、midaemon と共に ttd.conf からのトランザクション定義の読み込み、登録、同期を実行します。

ttd は、ovpa start コマンドで Performance Collection Component の scope データ コレクタを起動すると開始します。ディスパッチされると、ttd はバックグラウンド モードで動作し、エラーは /var/opt/perf/status.ttd ファイルに書き込まれます。

また、トランザクションを処理し、これらのトランザクションに関連したパフォーマンス メトリックを収集するためには、midaemon を実行する必要があります (次のページを参照)。

ttd を停止しないことを強くお勧めします。

ttd を停止させる必要がある場合、ttd と Performance Collection Component のプロセスを再起動する前に、実行中の ARM 組み込みアプリケーションをすべて終了させる必要があります。ttd は、システム上で実行されている arm_init および arm_getid コールを取得するために、動作している必要があります。ttd が停止して再起動すると、これらのコールにより返されたトランザクション ID は繰り返されるため、ARM メトリックは無効になります。

Performance Collection Component のプロセスを起動して、プロセスが正確な順序で開始されたことを確認するには、ovpa スクリプトを使用します。ovpa stop では ttd をシャットダウンできません。任意のパフォーマンス ソフトウェアを再インストールするために ttd をシャットダウンする必要がある場合、/<**インストール ディレクトリ**>/bin/ttd -k コマンドを使用します。ただし、Performance Collection Component を再インストールする以外では、ttd を停止させることはお勧めしません。

Performance Collection Component がシステム上にない場合、GlancePlus は midaemon を開始します。実行されていない場合、midaemon は、midaemon が測定データの処理を開始する前に、ttd を開始させます。

すべてのプログラム オプションについては、ttd のマニュアル ページを参照してください。

ARM API コール ステータス リターン

トランザクションを登録するには、ttd プロセスが常に実行されている必要があります。ttd が何らかの理由で終了し、実行されていない場合、arm_init または arm_getid コールは「失敗」というリターン コードを返します。ttd がその後再起動すると、新しい arm_getid コールは、他のプログラムがすでに使用しているものと同じトランザクション ID を再登録する可能性があります。そうすると、無効なデータが記録されます。

ttd を停止して再起動すると、ARM 組み込みアプリケーションは、-2 (TT_TTDNOTRUNNING) の戻り値と ARM API コールの EPIPE errno エラーを受信し始めることがあります。アプリケーションが初めて起動すると、最初の ARM API コールでクライアント接続ハンドルが作成されます。このクライアント ハンドルにより、アプリケーションは ttd プロセスと通信することができます。ttd が終了するとこの接続も無効になるため、次にアプリケーションが ARM API コールの使用を試みると、TT_TTDNOTRUNNING の戻り値が送信されることがあります。このエラーは、たとえ他の ttd プロセスが実行されていても、前の ttd プロセスは実行されていないことを意味します (ARM API コールのリターンについては、*arm (3)* のマニュアル ページを参照してください)。

このエラーを回避するには、ttd が停止したら、ARM 組み込みアプリケーションを再起動する必要があります。初めに、ARM 組み込みアプリケーションを終了させます。次に、(/< **インストール ディレクトリ** >/bin/ovpa start または /< **インストール ディレクトリ** >/bin/ttd を使用して) ttd を再起動し、その後にアプリケーションを再起動します。アプリケーションを再起動すると、アプリケーションと ttd プロセスの間に新しいクライアント接続ハンドルが作成されます。

midaemon にエラーがあっても、いくつかの ARM API コールはエラーを返しません。たとえば、midaemon がその共有メモリー セグメント内の領域を使い果たした場合です。パフォーマンス メトリック GBL_TT_OVERFLOW_COUNT は 0 より大きくなります。オーバーフローの状況が発生すると、(ttd 以外の) 実行中のパフォーマンス ツールをシャットダウンし、共有メモリー セグメント内でより多くの領域を指定するために、-smdvss オプションを使用して、midaemon を再起動することもできます (詳細は、*midaemon* のマニュアル ページを参照してください)。

ARM エラーが発生しても、アプリケーションが実行し続けられるようにアプリケーションを作成することをお勧めします。ARM の状態がプログラムの実行に影響しないようにします。

arm_getid コールによってトランザクションを ttd に登録することができるアクティブなクライアント プロセスの数は、maxfiles カーネル パラメータに制限されます。このパラメータは、オープンするファイル数をプロセスごとに制御します。各クライアント登録の要求により、ttd で RPC 接続用のソケット (オープンされた 1 つのファイル) が開かれます。クライアント アプリケーションが終了すると、ソケットは閉じられます。そのため、この制限は、arm_getid コールによってトランザクションを登録した、稼働中のクライアントの数にのみ影響します。この制限に達すると、ttd はクライアントの arm_getid 要求に対して TT_TTDNOTRUNNING を返します。maxfiles カーネル パラメータを増やすと、ttd にトランザクションを登録する、稼働中のアプリケーション数の上限値を上げることができます。

測定インターフェイス デーモン (midaemon)

測定インターフェイス デーモン (Measurement Interface Daemon: midaemon) は、システム パフォーマンス情報を継続的に収集する、負荷の小さいプロセスです。midaemon は、Performance Collection Component がトランザクション データを収集し、GlancePlus がトランザクション データを報告するために実行されている必要があります。GlancePlus を開始すると、または ovpa start コマンドで scope を実行すると、midaemon は動作し始めます。

トランザクションを登録および追跡できるように、Performance Collection Component と GlancePlus では、midaemon と ttd の両方が実行されている必要があります。ovpa スクリプトは、mideamon を含む Performance Collection Component のプロセスを正しい順序で起動および終了させます。GlancePlus は mideamon が実行されていない場合にそれを起動します。midaemon は、ttd が実行されていない場合にそれを起動します。

midaemon の CPU オーバーヘッドについての情報は、このマニュアルの「CPU オーバーヘッド」を参照してください。

すべてのプログラム オプションについては、*midaemon* のマニュアル ページを参照してください。

トランザクション構成ファイル (ttd.conf)

トランザクション構成ファイル /var/opt/perf/ttd.conf により、アプリケーション名、トランザクション名、パフォーマンス区分範囲、および各トランザクションに適用するサービスレベルの目標を定義できます。ttd は、各トランザクションを登録する方法を決定するために ttd.conf を読み込みます。

ttd.conf のカスタマイズはオプションです。Performance Collection Component に付属しているデフォルトの構成ファイルは、アプリケーションに組み込まれているすべてのトランザクションを監視します。

業務用アプリケーションを使用していて、どのようなトランザクションがアプリケーションに組み込まれているかが分からない場合、デフォルトの ttd.conf ファイルを使用して、いくつかのデータを収集します。次に、利用可能なトランザクションを知るために、そのデータを調べます。その後、ttd.conf を変更して、そのアプリケーションのトランザクション データの収集をカスタマイズできます。

新しいアプリケーションの追加

新しい ARM 組み込みアプリケーションを ttd.conf ファイル内のデフォルトの slo と tran=* 行の range 値でシステムに追加する場合、新しいトランザクションを組み込む操作は不要です (tran、range、slo の詳細は、「構成ファイルのキーワード」のセクションを参照してください)。新しいトランザクションは自動的に取り出され、ttd.conf ファイル内の tran 行の slo および range の値が、新しいトランザクションに使用されます。

新しいトランザクションの追加

ttd.conf ファイルに追加を行ったら、次の手順を実行して、追加を有効にする必要があります。

- すべてのアプリケーションを終了する。
- ttd -hup -mi コマンドを **root** として実行する。

以上の操作により、ttd.conf ファイルが再度読み込まれ、新しいトランザクションが slo と range の値と共に ttd と midaemon に登録されます。再度読み込んでも、再読み込み前に ttd.conf ファイルにあったトランザクションの slo と range の値は変更されません。

range 値または slo 値の変更

ttd.conf ファイルにある既存のトランザクションの slo と range の値を変更する場合、次を実行します。

- すべての ARM 組み込みアプリケーションを終了する。
- **ovpa stop** を使用して scope コレクタを終了する。
- Glance の使用をすべて停止する。
- **ttd -k** コマンドを発行して ttd を停止する。
- ttd.conf ファイルを変更した後、
- **ovpa start** を使用して、scope を再起動する。

- ARM 組み込みアプリケーションを再起動する。

構成ファイルのキーワード

構成ファイル /var/opt/perf/ttd.conf は、表 1 に示すキーワードによって定義されるトランザクション属性に、トランザクション名を関連付けます。

表 1　　構成ファイルのキーワード

キーワード	構文	使用方法
tran	**tran=***transaction_name*	必要
slo	**slo=***sec*	オプション
range	**range=***sec [,sec,...]*	オプション

次に、これらのキーワードの詳細について説明します。

tran

tran はトランザクション名の定義に使用します。この名前は、組み込みアプリケーション内の arm_getid API コールで定義されているトランザクションに対応する必要があります。オプションの属性 range または slo を指定する前に、tran キーワードを使用する必要があります。tran は構成ファイルに必要な唯一のキーワードです。トランザクション名の末尾をアスタリスク (*) にすると、このエントリに対して登録要求が行われたときに、ワイルドカード パターン マッチが行われます。トランザクション名にはダッシュ (-) を使用できます。ただし、トランザクション名にスペースは使用できません。

トランザクション名は最大 128 文字まで使用できます。ただし、最初の 60 文字だけが Performance Collection Component に表示されます。GlancePlus は専用の画面で 128 文字を表示できます。

デフォルトの ttd.conf ファイルには、いくつかのエントリが含まれています。最初のエントリは、ARM API コールに組み込まれている Performance Collection Component のデータ コレクタ scope で使用されるトランザクションを定義します。また、このファイルにはエントリ tran=* が含まれています。これは、ARM API または トランザクション追跡 API コールに組み込まれているアプリケーション内のその他のトランザクションをすべて登録します。

range

range はトランザクション パフォーマンス区分範囲を指定するために使用します。パフォーマンス区分範囲により、各トランザクションが完了するまでの各所用時間の区分と、各所用時間内にトランザクションが成功した回数を知ることができます。定義された範囲は GlancePlus の [トランザクション追跡] ウィンドウに表示されます。

sec に入力されたそれぞれの値は、その範囲のトランザクションの上限を秒単位で表します。値は小数点以下最大 6 桁で、整数または実数です。HP-UX では、1 マイクロ秒 (0.000001 秒) の精度が許容されます。ただし、他のプラットフォームでの精度は 10 ミリ秒 (0.01 秒) です。したがって、許容されるのは小数点以下 2 桁のみです。

最大 10 個の範囲を各トランザクションごとに定義できます。ユーザーは最大 9 個の範囲を指定できます。残り 1 個の範囲は、ユーザーが定義した最大の範囲を超えるトランザクション データを収集する、オーバーフロー範囲として確保されています。9 個以上の範囲を指定した場合、最初の 9 個が使用され、その後は無視されます。

9 個未満の範囲を指定した場合、最初の非指定範囲がオーバーフロー範囲になります。他の残りの非指定範囲は使用されません。非指定範囲のメトリックは、0.000 としてレポートされます。対応する最初の非指定カウント メトリックはオーバーフロー カウントになります。残りの非指定範囲のメトリックは常に ゼロ (0) です。

範囲は昇順で定義する必要があります (この章の後の例を参照)。

slo

slo は、パフォーマンス サービス レベル協定 (SLA) を監視するために必要なサービス レベルの目標 (SLO) を、秒単位で指定するために使用します。

range キーワードと同じように、値は小数点以下最大 6 桁の数字で、整数または実数です。HP-UX では、1 マイクロ秒 (0.000001 秒) の精度が許容されます。ただし、他のプラットフォームでの精度は 10 ミリ秒 (0.01 秒) です。したがって、許容されるのは小数点以下 2 桁のみです。

HP-UX で、マイクロ秒でトランザクションを並べ替えることができても、トランザクション時間は 100 マイクロ秒単位で表示されることに注意してください。

構成ファイルのフォーマット

ttd.conf ファイルには、一般的なトランザクションとアプリケーション固有のトランザクションの 2 種類のエントリが含まれています。

一般的なトランザクションは、アプリケーションが定義される前に、ttd.conf ファイルで定義する必要があります。これらのトランザクションは、定義されるすべてのアプリケーションに関係しています。デフォルトの ttd.conf ファイルには一般的なトランザクション エントリが 1 つと、ARM API コールに組み込まれた scope コレクタのエントリが複数含まれます。

```
tran=* range=0.5, 1, 2, 3, 5, 10, 30, 120, 300 slo=5.0
```

オプションとして、各アプリケーションはトランザクション名の固有のセットを持つことができます。これらのトランザクションはそのアプリケーションのみに関係します。指定するアプリケーション名は、組み込みアプリケーション内の arm_init API コールに定義されているアプリケーション名に対応する必要があります。アプリケーション固有のエントリの各グループは、次に示すように、[ ] で囲まれたアプリケーションの名前から始まる必要があります。例 :

```
[AccountRec]
tran=acctOne range=0.01, 0.03, 0.05
```

アプリケーション名は最大 128 文字まで使用できます。ただし、最初の 60 文字だけが Performance Collection Component に表示されます。Glance は、専用の画面で 128 文字を表示できます。

一般的なトランザクションと同じ名前のトランザクションがある場合、アプリケーションに記述されたトランザクションが使用されます。

次に例を示します。

```
tran=abc range=0.01, 0.03, 0.05 slo=0.10
tran=xyz range=0.02, 0.04, 0.06 slo=0.08
tran=t* range=0.01, 0.02, 0.03

[AccountRec}
tran=acctOne range=0.04, 0.06, 0.08
tran=acctTwo range=0.1, 0.2
tran=t* range=0.03, 0.5

[AccountPay]

[GenLedg]
tran=GenLedgOne range=0.01
```

上記の例では、最初の 3 つのトランザクションは、指定される 3 つのアプリケーションのすべてに当てはまります。

アプリケーション [AccountRec] には、トランザクション acctOne、acctTwo、abc、xyz、t* があります。一般的なトランザクション セット内のエントリのうちの 1 つには、「t*」という名前のワイルドカード トランザクションがあります。この場合、AccountRec アプリケーションの「t*」トランザクション名が使用され、一般的なトランザクション セット内のトランザクション名は無視されます。

アプリケーション [AccountPay] には一般的なトランザクション セットからのトランザクションのみがあります。

アプリケーション [GenLedg] には、トランザクション GenLedgOne、abc、xyz、t* があります。

トランザクション名の順序によって、アプリケーションで違いが生じることはありません。

アプリケーション名およびトランザクション名のその他の情報は、この章の「アプリケーション名とトランザクション名の指定」を参照してください。

構成ファイル例

例 1

```
tran=* range=0.5,1,2,3,5,10,30,12,30 slo=5.0
```

「*」エントリは、登録されているトランザクション名とエントリが一致していない場合に、デフォルト値として使用されます。これらのデフォルト値は、各システム上で「*」エントリを修正して変更できます。「*」エントリが見当たらない場合、上記の「*」エントリに割り当てられている初期値パラメータと一致する登録パラメータのデフォルトのセットが使用されます。

例 2

```
[MANufactr]
tran=MFG01 range=1,2,3,4,5,10 slo=3.0
tran=MFG02 range=1,2.2,3.3,4.0,5.5,10 slo=4.5
tran=MFG03
tran=MFG04 range=1,2.2,3.3,4.0,5.5,10
```

MANufctr アプリケーションのトランザクションです。MFG01、MFG02、MFG04 は、それぞれ固有のパラメータを使用します。MFG03 トランザクションは、時間区分およびサービス レベルの目標を追跡する必要がないため、パラメータを指定しません。

例 3

```
[Financial]
tran=FIN01
tran=FIN02 range=0.1,0.5,1,2,3,4,5,10,20 slo=1.0
tran=FIN03 range=0.1,0.5,1,2,3,4,5,10,20 slo=2.0
```

Financial アプリケーションのトランザクションです。FIN02 および FIN03 は、それぞれ固有のパラメータを使用します。FIN01 トランザクションは、時間区分およびサービス レベルの目標を追跡する必要がないため、パラメータを指定しません。

例 4

```
[PERSONL]
tran=PERS*  range=0.1,0.5,1,2,3,4,5,10,20 slo=1.0
tran=PERS03 range=0.1,0.2,0.5,1,2,3,4,5,10,20 slo=0.8
```

PERS03 アプリケーションのトランザクションです。PERSONL は固有のパラメータを使用します。残りの人事アプリケーション トランザクションは、PERSONL アプリケーションに固有のデフォルトのパラメータ セットを使用します。

例 5

```
[ACCOUNTS]
tran=ACCT_*     slo=1.0
tran=ACCT_REC range=0.5,1,2,3,4,5,10,20 slo=2.0
tran=ACCT_PAY range=0.5,1,2,3,4,5,10,20 slo=2.0
```

ACCOUNTS アプリケーションのトランザクションです。ACCT_REC および ACCT_PAY トランザクションは、それぞれ固有のパラメータを使用します。残りの会計トランザクションは、会計アプリケーションに固有のデフォルトのパラメータ セットを使用します。買掛金および売掛金トランザクションだけが時間区分を追跡する必要があります。トランザクション名の順序によって、アプリケーションで違いが生じることはありません。

ARM を使用する際のオーバーヘッドの考慮事項

現在のバージョンの Performance Collection Component および GlancePlus には、ARM 2.0 に必要な追加のデータをサポートする測定インターフェイスの修正が含まれています。これらの修正により、パフォーマンス管理上のオーバーヘッドの増加が発生します。アプリケーションに ARM を組み込む場合、オーバーヘッドについて考慮してください。

オーバーヘッドについては、この章の後で説明されています。

ガイドライン

アプリケーションに ARM API を組み込むためのガイドラインを次に示します。

- 個別のトランザクション ID の数の合計は最高 4,000 に制限されます。一般に、異なるトランザクションの単一インスタンスを持つよりも、同じトランザクションの複数のインスタンスを持つより方が負荷は少なくなります。実際に監視されるトランザクションのみを登録してください。

- arm_start および arm_stop の API コールによるオーバーヘッドは非常に小さいものですが、多くのトランザクション インスタンスがある場合、オーバーヘッドが増加することがあります。ほとんどのシステム上で、1 秒単位あたりに数千を超える arm_start および arm_stop の API コールは、全体的なパフォーマンスに著しく影響を与えます。
- ARM 2.0 の機能を使用する場合にのみ ARM 相関係数を使用してください。ARM 相関係数に関する情報は、『*アプリケーション応答測定 2.0 API ガイド*』の「高度なトピック」の項を参照してください。相関係数情報の作成、移動、監視などのオーバーヘッドは、ARM 2.0 相関係数機能を使用しないトランザクションの監視よりも著しく高くなります。
- より大きな文字列サイズ (長いトランザクション名、アプリケーション名およびユーザー定義の文字列メトリックを登録するアプリケーション) は、さらにオーバーヘッドを増加させます。

ディスク I/O オーバーヘッド

パフォーマンス管理ソフトウェアは、システム上で大きなディスク オーバーヘッドにはなりません。通常 Glance は、ディスクにそのデータを記録しません。Performance Collection Component の収集デーモン (scope) はディスク ログ ファイルを生成しますが、ARM 2.0 によってそれらのサイズが影響を受けることはほとんどありません。logtran scope ログ ファイルは ARM データを格納するために使用されます。

CPU オーバーヘッド

ARM コールを組み込んだプログラムは通常、ARM コールによる速度の低下はありません。これは、arm_getid コールの割合が毎秒 1 回未満のコールであり、また、arm_start および arm_stop コールの割合は毎秒数千より少ないと仮定した場合です。これ以上に頻繁な ARM API へのコールは避けてください。

ARM をサポートために発生する追加の CPU オーバーヘッドの多くは、パフォーマンス ツール プログラムおよびデーモンの内部で発生します。midaemon CPU のオーバーヘッドはわずかに増加しますが、ARM 1.0 に比較して 2 パーセント以下です。midaemon がトランザクションごとのリソース メトリックを追跡するように要求されている場合、トランザクション インスタンスごとのオーバーヘッドは、トランザクションごとのリソース メトリックを追跡しない場合の 2 倍程度です (parm ファイル内で log transaction=resource フラグを設定することにより、トランザクションごとのリソース メトリックの追跡が可能です)。さらに、Glance および scope の CPU オーバーヘッドは、ARM 2.0 コールが組み込まれたアプリケーションのあるシステムでわずかに高くなります。相関係数やユーザー定義メトリックを拡張利用する、ARM 2.0 コールが組み込まれたアプリケーションだけが、midaemon、scope または Glance 上でのパフォーマンスに大きく影響します。

利用可能なデフォルトの共有メモリーを使用率が超過した場合、midaemon オーバーフロー状態が発生することがあります。この結果、次を引き起こします。

- オーバーフローの状態が発生すると、ARM コールからのリターン コードがない。
- 空白のプロセス名など、不正なメトリックが表示される。
- 「out of space」などのエラーが status.mi に記録される。

メモリー オーバーヘッド

ARM API コールを生成するプログラムは、ARM 2.0 相関係数およびユーザー定義メトリック情報を渡すために使用されるスペースを除いて、それらのメモリー仮想セット サイズに大きな影響はありません。『*アプリケーション応答測定 2.0 API ガイド*』で説明されているこれらのバッファは、プロセスのメモリーの必要条件に影響を与えません。

ARM 2.0 をサポートするパフォーマンス ツールでは、追加の仮想セット サイズのオーバーヘッドがあります。Performance Collection Component および Glance によって使用されるために ARM データが内部に保持される場合、midaemon プロセスは共有メモリー セグメントを作成します。この共有メモリー セグメントのサイズは、ARM 2.0 により使用される可能性を含めたために、ARM 1.0 リリースでのサイズに比べて大きくなりました。ほとんどのシステム上のデフォルトでは、この共有メモリー セグメントのサイズは、およそ 11 メガバイトです。必要でない場合、このセグメントが物理メモリーに必ずあるとは限りません。したがって、まだメモリーの制約を受けていないほとんどのシステムには大きな影響を与えません。midaemon のメモリー オーバーヘッドは、特別な開始パラメータを使用して調整できます (*midaemon* のマニュアル ページを参照してください)。

20 トランザクションについて

この章では、トランザクション追跡とサービス レベルの目標管理の開始に必要な情報を提供します。詳細は、第 19 章の「トランザクション追跡の動作」を参照してください。また、使用例については、第 23 章の「トランザクション追跡の例」を参照してください。

開始する前に

Performance Collection Component は、`libarm.*` 共有ライブラリを次の場所に配置します。

プラットフォーム	パス
IBM RS/6000	`/usr/lpp/perf/lib/`
その他の UNIX プラットフォーム	`/opt/perf/lib/`

Performance Collection Component がシステムにインストールされていない場合、または `libarm.*` が上記のプラットフォームのパスに存在しない場合、このマニュアルの最後の 381 ページの「プラットフォームごとの C コンパイル オプション例」を参照してください。また、入手については『*アプリケーション応答測定 2.0 API ガイド*』の「ARM 共有ライブラリ (`libarm`)」を参照してください。`libarm` については、このマニュアルの最後の 375 ページの「ARM ライブラリ (libarm)」を参照してください。

トランザクション追跡のセットアップ

アプリケーションにトランザクション追跡を設定するには、次の手順を行います。これらの手順については、この章の後で詳細を説明します。

1. 監視する主要なトランザクションおよび期待する応答レベルを決定し、SLO を定義します (オプション)。
2. Performance Collection Component および Performance Manager のトランザクションを監視する場合、Performance Collection Component の `parm` ファイルのトランザクション ログがオンになっていることを確認してください。その後、Performance Collection Component を開始または再起動して、最新の `parm` ファイルを読み込みます。

   GlancePlus でトランザクションを見るために `parm` ファイルを編集する必要はありません。ただし、GlancePlus でトランザクション データを見るには、ttd が実行されている必要があります。GlancePlus を開始すると、自動的に `ttd` が起動します。

3 このマニュアルと『*アプリケーション応答測定 2.0 API ガイド*』で説明されている ARM API コールが組み込まれたアプリケーションを実行します。

4 収集されたトランザクション データを見るには、Performance Collection Component または Performance Manager を使用します。または、現在のデータを見るには GlancePlus を使用します。Performance Manager でデータが表示されない場合、データ ソースを閉じてから再度接続します。

5 アプリケーションのトランザクション データを収集する方法を修正するには、構成ファイル `ttd.conf` をカスタマイズします (オプション)。

6 `ttd.conf` ファイルに追加を行ったら、次の手順を実行して、追加を有効にする必要があります。

   a すべての ARM 組み込みアプリケーションを終了する。

   b `ttd -hup -mi` コマンドを **`root`** として実行する。

   これらの操作を実行すると、`ttd.conf` ファイルが再度読み込まれ、新しいトランザクションが `slo` と `range` の値と共に `ttd` および `midaemon` に登録されます。再度読み込んでも、再読み込み前に `ttd.conf` ファイルにあったトランザクションの `slo` と `range` の値は変更されません。

7 `ttd.conf` ファイルで、既存のトランザクションの `slo` と `range` の値を変更するには、次を実行します。

   a すべての ARM 組み込みアプリケーションを終了する。

   b `ovpa stop` を使用して `scope` コレクタを終了する。

   c Glance の使用をすべて停止する。

   d **`ttd -k`** コマンドを使用して `ttd` を終了する。

変更を加えたら、

   a **`ovpa start`** を使用して、`scope` を再起動する。

   b ARM を含むアプリケーションを開始する。

サービス レベルの目標 (SLO) の定義

トランザクション追跡を実装する最初のステップは、顧客の期待に応えるために必要となる主要なトランザクションと、トランザクションの応答レベル値を決定することです。要求される応答レベル値がサービス レベルの目標 (SLO) になります。サービス レベルの目標は、トランザクション構成ファイル `ttd.conf` に定義します。

サービス レベルの目標 (SLO) の定義は、IT 部門のサービス レベル協定 (SLA) を見直して、サービス レベル協定 (SLA) を満たすために監視する必要のあるトランザクションを確認する程度の簡単なものでもかまいません。SLA がなければ、導入されることをお勧めします。ただし、SLA の作成はトランザクションを追跡するために必須ではありません。

parm ファイルの変更

必要に応じて、Performance Collection Component の parm ファイルを変更し、Performance Manager および Performance Collection Component で使用するために、記録する項目のリストにトランザクションを追加ます。トランザクションを追加するには、次の例のように parm ファイルの log パラメータに transaction オプションを追加します。

```
log global application process transaction device=disk
```

log transaction パラメータのデフォルトは、no resource と no correlator です。リソース データ収集または相関係数データの収集を行うには、log transaction=resource または log transaction=correlator を指定します。両方とも、log transaction=resource、correlator を指定することで記録できます。

Performance Collection Component および Performance Manager で使用するトランザクション データを収集する前に、次の方法で最新の parm ファイルを起動させる必要があります。

Performance Collection Component の状態	トランザクション追跡の起動コマンド
実行中	ovpa restart
停止中	ovpa start

トランザクション データの収集

アプリケーションを開始します。トランザクション追跡デーモン (ttd)、および測定インターフェイス デーモン (midaemon) は、アプリケーションの実行に同期してトランザクション データを収集します。データは、Performance Collection Component または GlancePlus が使用できるように、midaemon の共有メモリー セグメントに格納されます。アプリケーションの各トランザクション データを表示する各ツールについては、359 ページの「パフォーマンス データの監視」を参照してください。

エラーの対応

パフォーマンスの考慮事項にあるように、問題のあるすべての ARM API コールまたは TT API コールが、リアルタイムにエラーを返すとは限りません。エラーが期待どおりに返らない場合の例を次に示します。

- 初期化されていない変数など、問題のある id パラメータを指定して arm_start をコールした場合。
- 前回有効な arm_start コールを行わずに、arm_stop をコールした場合。

Performance Collection Component - アプリケーションに ARM コールを組み込むときにこれらの状況をデバッグするには、十分な量のトランザクション データを生成および収集できるだけの時間、アプリケーションを実行します。Performance Collection Component でデータを収集した後に、extract プログラムの export コマンドを使用して、logtran ファイルからデータをエクスポートします。データを検査して、必要なトランザクションがすべて記録されていることを確認します。また、エラーに関しては、/var/opt/perf/status.ttd ファイルを参照してください。

GlancePlus - アプリケーションに ARM コールを組み込むときにこれらの状況をデバッグするには、十分な量のトランザクション データを生成および収集できるだけの時間、アプリケーションを実行します。次に、GlancePlus を使用して、すべてのトランザクションが期待どおりに表示されることを確認します。

固有のトランザクションの制限

特定のシステム リソースおよびカーネルの設定によっては、アプリケーションで許容される固有のトランザクションの数が制限される場合があります。この上限は通常、数千回の固有の `arm_getid` コールです。

`midaemon` によって使用される共有メモリー セグメントがいっぱいになると、固有のトランザクションの数が制限を超える場合があります。この場合、GlancePlus にオーバーフロー メッセージが表示されます。Performance Collection Component ではこのメッセージは表示されず、以後の新しいトランザクション データは記録されません (ただし、オーバーフロー メッセージは `/var/opt/perf/status.scope` で確認できます)。GlancePlus では、以後の新しいトランザクション データは表示されません。すでに登録されているトランザクションは引き続き記録および報告されます。測定できなかった新しいトランザクションの数は、GlancePlus の `GBL_TT_OVERFLOW_COUNT` メトリックで報告されます。

この状態は、`-smdvss` オプションを使用してより大きな共有メモリー セグメントのサイズを指定し、`midaemon` プロセスを終了して再起動することにより改善できます。現在の共有メモリー セグメントのサイズは、`midaemon -sizes` のコマンドを使用して確認できます。システムでの midaemon の最適化の詳細は、*midaemon* のマニュアル ページを参照してください。

構成ファイルのカスタマイズ (オプション)

アプリケーションからのトランザクション データを確認した後、アプリケーションのトランザクション データの収集方法を変更する場合、トランザクション構成ファイル `/var/opt/perf/ttd.conf` をカスタマイズできます。デフォルトの構成ファイル `ttd.conf` は、アプリケーションで定義されたすべてのトランザクションを取り扱うため、変更はオプションです。`ttd.conf` ファイルをカスタマイズする場合は、アプリケーションを実行するシステム上でこの作業を行ってください。`ttd.conf` を変更するには、**`root`** としてログオンする必要があります。

構成ファイル キーワード `tran`、`range`、`slo` についての情報は、第 19 章の「トランザクション追跡の動作」を参照してください。各キーワードの使用例を次に示します。

```
tran=Example:     tran=answerid
                  tran=answerid*
                  tran=*

range=Example:    range=2.5,4.2,5.0,10.009

slo=Example:       slo=4.2
```

トランザクションとそれに関連する属性をすべて含むように、構成ファイルを編集します。`tran` キーワードは、`range` キーワードまたは `slo` キーワードの前に使用する必要があります。`ttd.conf` 構成ファイルの例を次に示します。

```
tran=*
tran=my_first_transaction slo=5.5

[answerid]
tran=answerid1 range=2.5, 4.2, 5.0, 10.009 slo=4.2
```

```
[orderid]
tran=orderid1 range=1.0, 1.5, 2.0, 2.5, 3.0, 3.5, 4.0
```

ttd.conf ファイルに追加が必要な場合、次のように行ってください。

- すべての ARM 組み込みアプリケーションを終了する。
- ttd -hup -mi コマンドを **root** として実行する。

上記の操作を実行すると、ttd.conf ファイルが再度読み込まれ、新しいトランザクションが slo と range の値と共に ttd および midaemon に登録されます。再度読み込んでも、再読み込み前に ttd.conf ファイルにあったトランザクションの slo と range の値は変更されません。

ttd.conf ファイルで、既存のトランザクションの slo と range の値を変更するには、次を実行します。

1. すべての ARM 組み込みアプリケーションを終了する。
2. **ovpa stop** を使用して scope コレクタを終了する。
3. Glance の使用をすべて停止する。
4. **ttd -k** コマンドを使用して ttd を終了する。

変更を加えたら、

1. **ovpa start** を使用して、scope を再起動する。
2. ARM を含むアプリケーションを開始する。

パフォーマンス データの監視

次のリソースと Performance Collection Component、Performance Manager、GlancePlus などのパフォーマンス管理製品を使用して、トランザクション データを監視できます。

Performance Collection Component での使用

Performance Collection Component で、長期間のデータ収集および記録を行うことにより、システム パフォーマンスの分析および傾向分析が可能となります。Performance Collection Component のデータは、Performance Manager Agent で表示できる他、さまざまな他のパフォーマンス監視、会計報告、モデリング、計画ツールなどに使用するためにエクスポートできます。

Performance Collection Component の extract プログラムにより、データはスプレッドシート、および分析プログラムに使用するようにエクスポートできます。データはアーカイブと分析のために抽出できます。

Performance Collection Component でトランザクション データを監視するには、Performance Collection Component および ttd が実行されている必要があります。ovpa スクリプトを使用して Performance Collection Component を開始することで、GlancePlus によるトランザクション データの監視に必要な ttd および midaemon プロセスが、正しい順序で開始されます。

Performance Manager での使用

Performance Manager は Performance Collection Component のデータをインポートし、カスタマイズされたグラフ形式やスプレッド シート形式に変換して表示します。Performance Manager により、トランザクション データの履歴傾向の分析、将来予測をより正確に実行できます。

Performance Manager のデータ ソースとして、[Class List (クラス リスト)] ウィンドウから **TRANSACTION** を選び、さまざまなトランザクション メトリックをグラフ化できます。詳細は、Performance Manager のヘルプ メニューからアクセスできる Performance Manager のオンライン ヘルプを参照してください。Performance Manager で目的のトランザクションが表示されない場合、現在のデータ ソースを閉じてから再度接続してください。

GlancePlus での使用

GlancePlus を使用してシステムを監視することにより、リソースのボトルネックの識別や、コンピュータ システムに関するパフォーマンス情報をすぐにで得られます。GlancePlus には、定義したすべてのトランザクションに関する情報を表示する [トランザクション追跡] ウィンドウ、および個別のトランザクションに関する特定の情報を表示する [トランザクション グラフ] ウィンドウがあります。たとえば、定義した SLO に対する各トランザクションの実行状況が分かります。GlancePlus の使用方法の詳細は、ヘルプ メニューからアクセスできるオンライン ヘルプを参照してください。

アラーム

Performance Collection Component、Performance Manager、GlancePlus のリソースおよびパフォーマンス管理製品により、トランザクション データに関するアラームを生成することができます。

Performance Collection Component での使用

Performance Collection Component でアラームを生成するには、アラーム定義ファイル `alarmdef` でアラーム状態を定義する必要があります。E-mail メッセージの送信やページャの呼び出しなど、さまざまな方法でアラーム状態を通知するように Performance Collection Component を設定できます。

`alarmdef` ファイルの構文チェックをパスさせるには、ログ ファイルにそのアプリケーション名とトランザクション名のデータを記録するか、または `ttd.conf` ファイルに名前の登録をしておく必要があります。

名称にダッシュ (-) を含むトランザクションにアラーム状態を定義する場合には、制限があります。この制限を回避するには、`alarmdef` ファイル内で `ALIAS` コマンドを使用してトランザクション名を再定義します。

GlancePlus での使用

トランザクション パフォーマンスに関するアラームを生成するようにアドバイザ シンタックスを設定することができます。たとえば、アラーム状態が満たされた場合、情報を `stdout` に表示したり、UNIX コマンド (`mailx` など) を実行したり、GlancePlus メイン ウィンドウ上の [アラーム] ボタンを黄色や赤色に変えたりするように、GlancePlus に指示できます。GlancePlus のアラームの詳細は、[アドバイザ シンタックスの編集] ウィンドウのヘルプ メニューから [**ウィンドウについて**] を選択してください。

21 トランザクション追跡のメッセージ

アプリケーションに ARM API コールまたは TT API コールを組み込む場合、表 2 に示すエラーコードが返されるため、アプリケーション開発者はそれらを利用することができます。

表 2　　エラー コード

エラー コード	Errno 値	説明
-1	EINVAL	無効な引数
-2	EPIPE	ttd (登録デーモン) が実行されていない
-3	ESRCH	ttd.conf ファイルにトランザクション名がない
-4	EOPNOTSUPP	オペレーティング システムのバージョンがサポートされていない

ARM API コールまたは TT API コールが組み込まれたアプリケーションが実行されている場合、発生するエラーのリターン コードはトランザクション追跡デーモン ttd から返されます。測定インターフェイス デーモン midaemon はいかなるエラー リターン コードも生成しません。

midaemon エラーが発生する場合、内容の詳細は /var/opt/perf/status.mi ファイルを参照してください。

22 トランザクション メトリック

ARM エージェントは、GlancePlus と Performance Collection Component の共有コンポーネントとして提供され、さまざまなトランザクション メトリックを作成します。次の方法で、メトリックとその説明についての詳細なリストを表示できます。

- インストールされている GlancePlus のメトリックについては、GlancePlus のオンライン ヘルプを使用するか、または次の場所にある『*GlancePlus for HP-UX Dictionary of Performance Metrics*』を参照してください。

  UNIX/Linux 上の /< インストール ディレクトリ >/paperdocs/gp/C/ にある gp-metrics.txt。

  InstallDir は、Performance Collection Component がインストールされているディレクトリです。

- 特定のプラットフォームにインストールされている Performance Collection Component のメトリックについては、次の場所にある、プラットフォームに対応した「*HP Operations エージェント Dictionary of Operating System Performance Metrics*」ファイルを参照してください。

  UNIX/Linux 上の /< インストール ディレクトリ >/paperdocs/ovpa/C/ にある met< *プラットフォーム* >.txt。

  Windows 上の *%ovinstalldir%*paperdocs\ovpa\C にある met< *プラットフォーム* >.txt。

23 トランザクション追跡の例

この章には、アプリケーションに ARM API コールを組み込み、アプリケーションに定義されたトランザクションを Performance Collection Component または GlancePlus で監視できるようにする擬似コード例が含まれています。この擬似コード例は、第 18 章の「トランザクション追跡の紹介」で説明されているリアルタイム注文処理のシナリオに対応します。

この章には、リアルタイム注文処理シナリオに対応する構成ファイルの例など、いくつかのトランザクション構成ファイルの例が含まれています。

リアルタイム注文処理の擬似コード

この擬似コード例には、第 18 章の「トランザクション追跡の紹介」で説明されている、リアルタイム注文処理シナリオのトランザクションを定義するために使用された ARM API コールが含まれています。オペレータが電話注文を受けるたびに、このルーチンが処理されます。この例では、ARM API コールを含む行は太字で示されます。

```
routine answer calls()
{
*****************************************************
*  Register the transactions if first time in       *
*****************************************************
   if (transactions not registered)
   {
      appl_id = arm_init("Order Processing Application","*", 0,0,0)
      answer_phone_id = arm_getid(appl_id,"answer_phone","1st tran",0,0,0)
      if (answer_phone_id < 0)
         REGISTER OF ANSWER_PHONE FAILED - TAKE APPROPRIATE ACTION
      order_id = arm_getid(appl_id,"order","2nd tran",0,0,0)
      if (order_id < 0)
         REGISTER OF ORDER FAILED - TAKE APPROPRIATE ACTION
      check_id = arm_getid(appl_id,"check_db","3rd tran",0,0,0)
      if (check_id < 0)
         REGISTER OF CHECK DB FAILED - TAKE APPROPRIATE ACTION
      update_id = arm_getid(appl_id,"update","4th tran",0,0,0)
      if (update_id < 0)
         REGISTER OF UPDATE FAILED - TAKE APPROPRIATE ACTION

   } if transactions not registered
*****************************************************
*  Main transaction processing loop
*****************************************************
   while (answering calls)
   {
      if (answer_phone_handle = arm_start(answer_phone_id,0,0,0) < -1)
         TRANSACTION START FOR ANSWER_PHONE NOT REGISTERED
```

```
*****************************************************
*  At this point the answer_phone transaction has    *
*  started.If the customer does not want to order, *
*  end the call; otherwise, proceed with order.    *
*****************************************************
      if (don't want to order)
          arm_stop(answer_phone_handle,ARM_FAILED,0,0,0)
          GOOD-BYE - call complete
      else
      {
*****************************************************
* They want to place an order - start an order now  *
*****************************************************
         if (order_handle = arm_start(order_id,0,0,0) < -1)
            TRANSACTION START FOR ORDER FAILED
  take order information: name, address, item, etc.
*****************************************************
* Order is complete - end the order transaction    *
*****************************************************
          if (arm_stop(order_handle,ARM_GOOD,0,0,0) < -1)
             TRANSACTION END FOR ORDER FAILED
*****************************************************
*  order taken - query database for availability     *
*****************************************************
         if (query_handle = arm_start(query_id,0,0,0) < -1)
            TRANSACTION QUERY DB FOR ORDER NOT REGISTERED
         query the database for availability
*****************************************************
* database query complete - end query transaction  *
*****************************************************
          if (arm_stop(query_handle,ARM_GOOD,0,0,0) < -1)
             TRANSACTION END FOR QUERY DB FAILED

*****************************************************
* If the item is in stock, process order, and        *
* update inventory.                                  *
*****************************************************
         if (item in stock)
            if (update_handle = arm_start(update_id,0,0,0) < -1)
               TRANSACTION START FOR UPDATE NOT REGISTERED
            update stock
*****************************************************
* update complete - end the update transaction     *
*****************************************************
            if (arm_stop(update_handle,ARM_GOOD,0,0,0) < -1)
               TRANSACTION END FOR ORDER FAILED
*****************************************************
* Order complete - end the call transaction          *
*****************************************************
         if (arm_stop(answer_phone_handle,ARM_GOOD,0,0,0) < -1)
            TRANSACTION END FOR ANSWER_PHONE FAILED
      } placing the order
   GOOD-BYE - call complete
```

```
   sleep("waiting for next phone call...zzz...")
   } while answering calls
   arm_end(appl_id, 0,0,0)
} routine answer calls
```

構成ファイル例

このセクションには、いくつかのトランザクション構成ファイル /var/opt/perf/ttd.conf の例が含まれています。ttd.conf ファイルおよび構成ファイルのキーワードの詳細は、第 19 章の「トランザクション追跡の動作」を参照してください。

例 1 (注文処理の擬似コード例)

```
# The "*" entry below is used as the default if none of the
# entries match a registered transaction name.

tran=* range=0.5,1,1.5,2,3,4,5,6,7 slo=1
tran=answer_phone* range=0.5,1,1.5,2,3,4,5,6,7 slo=5
tran=order* range=0.5,1,1.5,2,3,4,5,6,7 slo=5
tran=query_db* range=0.5,1,1.5,2,3,4,5,6,7 slo=5
```

例 2

```
# The "*" entry below is used as the default if none of the
# entries match a registered transaction name.

tran=* range=1,2,3,4,5,6,7,8 slo=5

# The entry below is for the only transaction being
# tracked in this application.The "*" has been inserted
# at the end of the tran name to catch any possible numbered
# transactions.For example "First_Transaction1",
# "First_Transaction2", etc.

tran=First_Transaction* range=1,2.2,3.3,4.0,5.5,10 slo=5.5
```

例 3

```
# The "*" entry below is used as the default if none of the
# entries match a registered transaction name.

tran=*
tran=Transaction_One range=1,10,20,30,40,50,60 slo=30
```

例 4

```
tran=FactoryStor* range=0.05, 0.10, 0.15 slo=3

# The entries below shows the use of an application name.
# Transactions are grouped under the application name.This
# example also shows the use of less than 10 ranges and
# optional use of "slo."

[Inventory]
tran=In_Stock range=0.001, 0.004, 0.008
tran=Out_Stock range=0.001, 0.005
tran=Returns range=0.1, 0.3, 0.7

[Pers]
tran=Acctg range=0.5, 0.10, slo=5
tran=Time_Cards range=0.010, 0.020
```

24 高度な機能

この章では、次の ARM 2.0 API 機能を Performance Collection Component で使用する方法を説明します。

- データ型
- ユーザー定義メトリック
- scope の組み込み

データ型の使用方法

表 3 は、Performance Collection Component でのデータ型の使用方法を示します。これは『*アプリケーション応答測定 2.0 API ガイド*』の「高度なトピック」のセクションの「データ型の定義」の補足です。

表 3　　データ型の使用方法 Performance Collection Component

ARM_Counter32	データは 32 ビットの整数で記録されます。
ARM_Counter64	データは 32 ビットの整数で、型変換されて記録されます。
ARM_CntrDivr32	計算された結果が 32 ビットの整数として記録されます。
ARM_Gauge32	データは 32 ビットの整数で記録されます。
ARM_Gauge64	データは 32 ビットの整数で、型変換されて記録されます。
ARM_GaugeDivr32	計算された結果が 32 ビットの整数として記録されます。
ARM_NumericID32	データは 32 ビットの整数で記録されます。
ARM_NumericID64	データは 32 ビット整数で、型変換されて記録されます。
ARM_String8	無視されます。
ARM_String32	無視されます。

Performance Collection Component は文字列データを記録しません。Performance Collection Component は 5 分ごとにデータを記録し、その間隔でのアクティビティの要約を記録するためです。アプリケーションから提供される文字列は要約できません。

Performance Collection Component は使用可能な最初の 6 つのユーザー定義メトリックから、その最大、最小、平均を記録します。ARM が組み込まれたアプリケーションは、Counter32、String8、NumericID 32、Gauge32、Gauge64、Counter64、NumericID64、String32、GaugeDivr32 を渡し、Performance Collection Component は 32 ビットの整数として、Counter32、NumericID32、Gauge32、Gauge64、NumericID32、NumericID64 の Min、Max、Average を 5 分間隔で記録します。Performance Collection Component では文字列を要約できないため、String8 と String32 は無視されます。最初の 6 つのユーザー定義メトリックのみが記録されるため、GaugeDivr32 も無視されます (その他の例は、次のセクションの「ユーザー定義メトリック」を参照してください)。

ユーザー定義メトリック

このセクションは『*アプリケーション応答測定 2.0 API ガイド*』の「高度なトピック」の「アプリケーション定義メトリック」の補足です。ここでは、Performance Collection Component がユーザー定義メトリック (ARM ではアプリケーション定義メトリックと呼ばれているもの) を処理する方法の例を示します。表 4 の例では、プログラムが次のようなデータ型を渡した場合に記録される内容を示します。

表 4　　記録される特定のプログラムのデータ型の例

プログラムが渡す内容	記録される内容
例 1 String8 Counter32 Gauge32 CntrDivr32	 Counter32 Gauge32 CntrDivr32
例 2 String32 NumericID32 NumericID64	 NumericID32 NumericID64

表 4　　記録される特定のプログラムのデータ型の例 (続き)

プログラムが渡す内容	記録される内容
例 3 NumericID32 String8 NumericID64 Gauge32 String32 Gauge64	NumericID32 NumericID64 Gauge32 Gauge64
例 4 String8 String32	(なし)
例 5 Counter32 Counter64 CntrDivr32 Gauge32 Gauge64 NumericID32 NumericID64	Counter32 Counter64 CntrDivr32 Gauge32 Gauge64 NumericID32

Performance Collection Component は文字列データを要約できないため、文字列データは記録されません。

例 1 では、カウンタ、ゲージ、カウンタ除数のみが記録されます。

例 2 では、数値のみが記録されます。

例 3 では、数値とゲージのみが記録されます。

例 4 では、何も記録されません。

例 5 では、最初の 6 つのユーザー定義メトリックのみが記録されるため、NumericID64 は記録されません。

scope の組み込み

ARM API コールには、scope データ コレクタが組み込まれています。Performance Collection Component が起動すると、scope は自動的に Scope_Get_Process_Metrics および Scope_Get_Global_Metrics の 2 つのトランザクションの記録を開始します。この 2 つのトランザクションは HP パフォーマンス ツールのアプリケーションにあります。

トランザクション データは 5 分ごとに記録されるため、各間隔で 5 つの Get Process トランザクション (1 分につき 1 つのトランザクション) が完了します。Scope_Get_Process_Metrics トランザクションはプロセス データの処理に組み込まれます。システムに 200 のプロセスがある場合、Scope_Get_Process_Metrics トランザクションには、システムに 30 のプロセスしかない場合より時間がかかります。

Scope_Get_Global_Metrics トランザクションは、グローバル データを含むすべての 5 分間隔のデータの収集に組み込まれます。これには、global、application、disk、transaction、およびその他のデータ型が含まれます。

プロセスとグローバル トランザクション データの記録を停止するには、ttd.conf ファイルにある scope トランザクションのエントリをコメント アウトするか削除します。

25 トランザクション ライブラリ

この付録では、次の項目について説明します。

- アプリケーション応答測定ライブラリ (libarm)
- プラットフォームごとの C コンパイラ オプションの例
- アプリケーション応答測定 NOP ライブラリ (libarmNOP)
- Java ラッパーの使用

ARM ライブラリ (libarm)

Performance Collection Component および GlancePlus を使用すると、ARM の機能のコンパイルおよび使用を簡単に行える環境を構築できます。開発に必要なライブラリは /opt/perf/lib/ に配置されています。コンパイルに関する特定の情報は、この付録の次のセクションを参照してください。

表 5 に示すライブラリ ファイルは HP-UX 11.11 以降の Performance Collection Component および GlancePlus のインストールに存在します。

表 5　HP-UX 11.11 以降の Performance Collection Component および GlancePlus のライブラリ ファイル

/opt/perf/lib/	libarm.0	ARM の HP-UX 10.X 互換共有ライブラリ (非スレッド セーフ) です。-larm を指定して HP-UX 10.20 にリンクされている HP-UX 11 でプログラムを実行した場合、11.0 ローダーは自動的にこのライブラリを参照します。
	libarm.1	HP-UX 11 の互換共有ライブラリ (スレッド セーフ) です。このライブラリは、-larm を指定して HP-UX リリースにリンクされているプログラムにより参照されます。10.20 にリンクするプログラムがこのライブラリを参照する場合 (たとえば、-L /opt/perf/lib にリンクしていない場合)、「/usr/lib/dld.sl: Unresolved symbol: _thread_once (code) from libtt.sl」というエラー メッセージが表示され、中断されます。
	libarm.sl	libarm.1 のシンボリック リンクです。
	libarmNOP.sl	ARM の「非操作用」共有ライブラリです (API コールは成功しますが何もしません。Performance Collection Component がインストールされていないシステム上でテストする場合に使用します)。
/opt/perf/examples/arm	libarmjava.sl	ARM の 32 ビット共有ライブラリです。
/opt/perf/examples/arm/arm64	libarmjava.sl	ARM の 64 ビット共有ライブラリです。
/opt/perf/lib/pa20_64/	これらのファイルは、HP-UX 11 上で +DD64 コンパイラ オプションでコンパイルされるプログラムにより自動的に参照されます。	
	libarm.sl	ARM の 64 ビット共有ライブラリです。
	libarmNOP.sl	ARM の 64 ビット「非操作用」共有ライブラリです (API コールは成功しますが何もしません。Performance Collection Component がインストールされていないシステム上でテストする場合に使用します)。

表 6 に示す追加のライブラリ ファイルは IA64 HP-UX インストールにあります。

表 6　　HP-UX IA64 ライブラリ ファイル

/opt/perf/lib/hpux32/	libarm.so.1	ARM の IA64/32 ビット共有ライブラリ
/opt/perf/lib/hpux64/	libarm.so.1	ARM の IA64/64 ビット共有ライブラリ
/opt/perf/examples/arm	libarmjava.so	ARM の 32 ビット共有ライブラリです。
/opt/perf/examples/arm/arm64	libarmjava.so	ARM の 64 ビット共有ライブラリです。

ARM ライブラリがコールする HP-UX のオペレーティング システムのバージョンは次のバージョンに変わることがあるため、プログラムは -larm を使って共有ライブラリ バージョンにリンクする必要があります。ARM API コールが組み込まれているアプリケーションのコンパイル、および ARM ライブラリのアーカイブ バージョンへのリンク (-Wl、-a archive) はサポートされていません (詳細は、第 19 章の 345 ページの「トランザクション追跡デーモン (ttd)」を参照してください)。

Performance Collection Component と GlancePlus がインストールされている AIX オペレーティング システムに存在するライブラリ ファイルは、次のとおりです。

表 7　　AIX ライブラリ ファイル

/usr/lpp/perf/lib/	libarm.a	32 ビット共有 ARM ライブラリです (スレッド セーフ)。このライブラリは -larm を使用してリンクされているプログラムによって参照されます。
/usr/lpp/perf/lib	libarmNOP.a	ARM の 32 ビット共有ライブラリです。このライブラリは、Performance Agent または Performance Collection Component がインストールされていないシステム上でのテストに使用されます。
/usr/lpp/perf/lib64/	libarm.a	64 ビット共有 ARM ライブラリです (スレッド セーフ)。このライブラリは -larm を使用してリンクされているプログラムによって参照されます。
/usr/lpp/perf/lib64	libarmNOP.a	ARM の 64 ビット共有ライブラリです。このライブラリは、Performance Agent または Performance Collection Component がインストールされていないシステム上でのテストに使用されます。

表 7　　AIX ライブラリ ファイル (続き)

/usr/lpp/perf/examples/arm	libarmjava.a	ARM の 32 ビット共有ライブラリ
/usr/lpp/perf/examples/arm/arm64	libarmjava.a	ARM の 64 ビット共有ライブラリです。
/usr/lpp/perf/lib/	libarmns.a	32 ビット アーカイブ ARM ライブラリです (スレッド セーフ)。機能的には、32 ビットの libarm.a と同じです。
/usr/lpp/perf/lib64/	libarmns.a	64 ビット アーカイブ ARM ライブラリです (スレッド セーフ)。機能的には、64 ビットの libarm.a と同じです。

Performance Collection Component と GlancePlus がインストールされている Solaris オペレーティング システムに存在するライブラリ ファイルは、次のとおりです。

表 8　　32 ビット プログラム用 Solaris ライブラリ ファイル

/opt/perf/lib/	libarm.so	32 ビット共有 ARM ライブラリです (スレッド セーフ)。このライブラリは -larm を使用してリンクされているプログラムによって参照されます。
	libarmNOP.so	ARM の 32 ビット共有ライブラリです。このライブラリは、Performance Collection Component がインストールされていないシステム上でのテストに使用されます。

表 9　　Sparc 64 ビット プログラム用 Solaris ライブラリ ファイル

/opt/perf/lib/sparc_64/	libarm.so	64 ビット共有 ARM ライブラリです (スレッド セーフ)。このライブラリは -larm を使用してリンクされているプログラムによって参照されます。
	libarmNOP.so	ARM の 64 ビット共有ライブラリ このライブラリは、Performance Agent または Performance Collection Component がインストールされていないシステム上でのテストに使用されます。
/opt/perf/examples/arm	libarmjava.so	ARM の 32 ビット共有ライブラリです。
/opt/perf/examples/arm/arm64	libarmjava.so	ARM の 64 ビット共有ライブラリです。

表 10　　x86 64 ビット プログラム用 Solaris ライブラリ ファイル

/opt/perf/lib/x86_64/	libarm.so	64 ビット共有 ARM ライブラリです (スレッド セーフ)。このライブラリは -larm を使用してリンクされているプログラムによって参照されます。
	libarmNOP.so	ARM の 64 ビット共有ライブラリ このライブラリは、Performance Agent がインストールされていないシステム上でのテストに使用されます。
/opt/perf/examples/arm	libarmjava.so	ARM の 32 ビット共有ライブラリです。
/opt/perf/examples/arm/arm64	libarmjava.so	ARM の 64 ビット共有ライブラリです。

32 ビット プログラム用に提供されているパラメータの他に、-xarch=generic64 コマンドライン パラメータを使用して 64 ビット プログラムをコンパイルする必要があります。

Performance Collection Component と GlancePlus がインストールされている Linux オペレーティング システムに存在するライブラリ ファイルは、次のとおりです。

表 11　　Linux ライブラリ ファイル

/opt/perf/lib/	libarm.so	32 ビット共有 ARM ライブラリです (スレッド セーフ)。このライブラリは -larm を使用してリンクされているプログラムによって参照されます。
	libarmNOP.so	ARM の 32 ビット共有ライブラリです。このライブラリは、Performance Collection Component がインストールされていないシステム上でのテストに使用されます。
/opt/perf/lib64/	libarm.so	64 ビット共有 ARM ライブラリです (スレッド セーフ)。このライブラリは -larm を使用してリンクされているプログラムによって参照されます。
	libarmNOP.so	ARM の 64 ビット共有ライブラリです。このライブラリは、Performance Collection Component がインストールされていないシステム上でのテストに使用されます。
/opt/perf/examples/arm	libarmjava.so	ARM の 32 ビット共有ライブラリです。
/opt/perf/examples/arm/arm64	libarmjava.so	ARM の 64 ビット共有ライブラリです。

Linux 2.6 IA 64 ビットでは、32 ビットの libarm.so と libarmjava.so は実装されていません。

プラットフォームごとの C コンパイル オプション例

arm.h インクルード ファイルは、/opt/perf/include/ にあります。このファイルは /usr/include/ からシンボリック リンク経由でアクセスできます。つまり、-I/opt/perf/include/ を使用する必要はありません (使用することもできます)。同じように、libarm は /opt/perf/lib/ に存在しますが、/usr/lib/ からリンクされます。ARM 組み込みアプリケーションを作成する場合、常に -L/opt/perf/lib/ を使用してください。

- Linux の場合 :

  次の例は、ARM API を使用する C プログラムのコンパイル コマンドを示します。

  ```
  cc myfile.c -o myfile -I /opt/perf/include -L
  -Xlinker -rpath -Xlinker /opt/perf/lib
  ```

- Linux の 64 ビット プログラムの場合 :

  ```
  cc  -m64 myfile.c -o myfile -I /opt/perf/include -L -Xlinker -rpath
  -Xlinker /opt/perf/lib64
  ```

- HP-UX の場合 :

  IA64 プラットフォーム上の HP-UX リリース 11.2x の場合、ARM 組み込み 32 ビット IA プログラムのコンパイルでは -L パラメータを -L/opt/perf/lib から -L/opt/perf/lib/hpux32 に変更し、ARM 組み込み 64 ビット IA プログラムのコンパイルでは -L/opt/perf/lib/hpux64 に変更します。

  次の例は、ARM API を使用する C プログラムのコンパイル コマンドを示します。

  ```
  cc myfile.c -o myfile -I /opt/perf/include -L /opt/perf/lib -larm
  ```

- Sun Solaris の場合 :

  次の例は、Sun Solaris 上の Performance Collection Component および GlancePlus で使用できます。

  ```
  cc myfile.c -o myfile -I /opt/perf/include -L /opt/perf/lib -larm -lnsl
  ```

- Sun Solaris の 64 ビット Sparc プログラムの場合 :

  次の例は、Sun Solaris 上の Performance Collection Component および 64 ビット プログラムで使用できます。

  ```
  cc -xarch=generic64 myfile.c -o myfile -I /opt/perf/include -L /opt/perf/
  lib/sparc_64 -larm -lnsl
  ```

- Sun Solaris の 64 ビット x86 プログラムの場合 :

  次の例は、Sun Solaris 上の Performance Agent および 64 ビット プログラムで使用できます。

  ```
  cc -xarch=generic64 myfile.c -o myfile -I /opt/perf/include -L /opt/perf/
  lib/x86_64 -larm -lnsl
  ```

- IBM AIX の場合 :

  IBM AIX 上のファイル配置は、他のプラットフォームと異なる (/opt/perf/ の代わりに /usr/lpp/perf/ が使用される) ため、IBM AIX の例は他のプラットフォームの例とは異なります。

  ```
  cc myfile.c -o myfile -I /usr/lpp/perf/include -L /usr/lpp/perf/lib -larm
  ```

- IBM AIX の 64 ビット プログラムの場合 :

  次の例は、IBM AIX 上の Performance Agent および 64 ビット プログラムで使用できます。

  ```
  cc -q64 myfile.c -o myfile -I /usr/lpp/perf/include -L /usr/lpp/perf/lib64
  -larm
  ```

C++ コンパイラでは、arm.h ファイルの適切な宣言を参照するために、コンパイル コマンドに -D_PROTOTYPES フラグを追加しなければならないことがあります。

ARM NOP ライブラリ

Performance Collection Component と Glance には、「非操作用」ライブラリ (libarmNOP.*:「*」は、OS プラットフォームに応じて sl、so、a のいずれか) が提供されています。この共有ライブラリは、各 ARM API コールに対して有効な状態を返す以外のことは行いません。これにより、Performance Collection Component および GlancePlus がインストールされていないシステム上で、ARM が組み込まれているアプリケーションを動作できます。

Performance Collection Component または GlancePlus がインストールされていないシステム上で ARM が組み込まれているアプリケーションを実行するには、NOP ライブラリを適切なディレクトリ (通常は /< インストール ディレクトリ >/lib/) にコピーして、libarm.sl (プラットフォームに応じて libarm.so または libarm.a) と名前を付けます。Performance Collection Component または GlancePlus がインストールされると、この NOP ライブラリは、正しく機能するライブラリによって上書きされます (他のファイルのように削除されません)。これにより、システムから Performance Collection Component または GlancePlus が削除されても、組み込まれたプログラムは停止しません。

Java ラッパーの使用

Java ネイティブ インターフェイス (JNI) ラッパーは、Java アプリケーションが HP ARM2.0 API をコールできるように作成された関数です。これらのラッパー (armapi.jar) は、ARM サンプル プログラムに含まれ、/< インストール ディレクトリ >/examples/arm/ ディレクトリに配置されています。InstallDir は、Performance Collection Component がインストールされているディレクトリです。

例

Java ラッパーの例は、/< インストール ディレクトリ >/examples/arm/ ディレクトリに配置されています。また、この場所には各ラッパーの機能を説明する README ファイルも保存されています。

アプリケーションの設定 (arm_init)

新規アプリケーションを設定するには、新しい ARMApplication のインスタンスを作成し、この API の名前と説明を渡します。各アプリケーションは、固有の名前によって識別される必要があります。ARMApplication クラスは C の関数 arm_init を使用します。

構文 :

```
ARMApplication myApplication =
new ARMApplication(“name”,”description”);
```

トランザクションの設定 (arm_getid)

新しいトランザクションを設定するには、ユーザー定義メトリック (UDM) を使用するかどうかを選択できます。Java ラッパーは、C の関数 arm_getid を使用します。

UDM を使用したトランザクションの設定

UDM を使用する場合は、まず新しい ARMTranDescription を定義する必要があります。ARMTranDescription は、arm_getid のデータ バッファを作成します (jprimeudm.java の例を参照)。

構文 :

```
ARMTranDescription myDescription =
new ARMTranDescription(“transactionName”,”details”);
```

詳細を使用しない場合は、別のコンストラクタを使用できます。

構文 :

```
ARMTranDescription myDescription =
new ARMTranDescription(“transactionName”);
```

メトリックの追加

メトリック 1-6:

構文 :

```
myDescription.addMetric(metricPosition, metricType,
metricDescription);
```

パラメータ :

metricPosition: 1-6

metricType: ARMConstants.ARM_Counter32
ARMConstants.ARM_Counter64 ARMConstants.ARM_CntrDivr32
ARMConstants.ARM_Gauge32 ARMConstants.ARM_Gauge64
ARMConstants.ARM_GaugeDivr32 ARMConstants.ARM_NumericID32
ARMConstants.ARM_NumericID64 ARMConstants.ARM_String8

メトリック 7:

構文 :

```
myDescription.addStringMetric(“description”);
```

これで、トランザクションを作成できます。

構文 :

```
myApplication.createTransaction(myDescription);
```

メトリック データの設定

メトリック 1-6:

構文 :

```
myTransaction.setMetricData(metricPosition, metric);
```

「メトリック」の例

```
ARMGauge32Metric metric = new ARMGauge32Metric(start);
ARMCounter32Metric metric = new ARMCounter32Metric(start);
ARMCntrDivr32Metric metric = new ARMCntrDivr32Metric(start, 1000);
```

メトリック 7:

構文 :

```
myTransaction.setStringMetricData(text);
```

UDM を使用しないトランザクションの設定

UDM を使用せずにトランザクションを設定する場合は、新しいトランザクションをすぐに作成できます。詳細を指定するかどうかを選択できます。

詳細を使用する場合

構文 :

```
ARMTransaction myTransaction =
myApplication.createTransaction("Transactionname","details";
```

詳細を使用しない場合

構文 :

```
ARMTransaction myTransaction =
myApplication.createTransaction("Transactionname");
```

トランザクション インスタンスの設定

新しいトランザクション インスタンスを設定するには、ARMTransaction の createTransactionInstance() メソッドを使って、新しい ARMTransactionInstance のインスタンスを作成します。

構文 :

```
ARMTransactionInstance myTranInstance =
myTransaction.createTransactionInstance();
```

トランザクション インスタンスの開始 (arm_start)

トランザクション インスタンスを開始するには、相関係数を使用するかどうかを選択できます。次のメソッドは、関連するパラメータを使って C の関数 arm_star をコールします。

相関係数を使用したトランザクション インスタンスの起動

相関係数を使用する場合は、相関係数の取得と送信を区別する必要があります。

相関係数の要求

トランザクション インスタンスが相関係数を要求する場合は、コールは次のようになります (jcorrelators.java の例も参照)。

構文 :

```
int status = myTranInstance.startTranWithCorrelator();
```

親相関係数の送信

以前のトランザクションの相関係数がすでに存在し、その相関係数をトランザクションに送信する場合は、構文は次のようになります。

構文 :

```
int status = startTran(parent);
```

パラメータ :

parent は送信された相関係数です。以前のトランザクションでは、getCorrelator() メソッドを使ってトランザクション インスタンス相関係数を取得できます。

親相関係数の要求と送信

以前のトランザクションの相関係数がすでに存在し、その相関係数をトランザクションに送信して相関係数を要求する場合は、構文は次のようになります。

構文 :

```
int status = myTranInstance.startTranWithCorrelator(parent);
```

パラメータ :

parent は送信された相関係数です。以前のトランザクションでは、getCorrelator() メソッドを使ってトランザクション インスタンス相関係数を取得できます。

相関係数情報の取得

getCorrelator() メソッドを使用して、トランザクション インスタンス相関係数を取得する場合は、次のようにします。

構文 :

```
ARMTranCorrelator parent = myTranInstance.getCorrelator();
```

相関係数を使用しないトランザクション インスタンスの起動

相関係数を使用しない場合は、トランザクション インスタンスを次のように開始できます。

構文：

```
int status = myTranInstance.startTran();
```

startTran は、一意のハンドルをステータスに返します。これは、更新および停止に使用されます。

トランザクション インスタンス データの更新

トランザクション インスタンスの UDM は、開始してから停止するまでの間に、何度でも更新できます。ラッパーのこの部分は、関連するパラメータを使って C の関数 arm_update をコールします。

UDM を使用したトランザクション インスタンス データの更新

UDM を使用してトランザクション インスタンスのデータを更新する場合は、まずメトリックの新しいデータを設定する必要があります。次に例を示します。

ARM_Counter32 ARM_Counter64、ARM_Gauge32、ARM_Gauge64、ARM_NumericID32、ARM_NumericID64 **の場合、**metric.setData(value)

ARM_CntrDivr32、ARM_GaugeDivr32 **の場合、**metric.setData(value,value)

ARM_String8、ARM_String32 **の場合、**metric.setData(string)

これで、メトリック データを新規に設定でき (「メトリック データの設定」セクションの例を参照)、更新をコールできます。

構文：

```
myTranInstance.updateTranInstance();
```

UDM を使用しないトランザクション インスタンス データの更新

UDM を使用しない場合は、更新をコールするだけでトランザクション インスタンスのデータを更新できます。これにより「ハートビート」が送信され、トランザクション インスタンスがまだ実行中であることが示されます。

構文：

```
myTranInstance.updateTranInstance();
```

大規模で非透過的なアプリケーション プライベート バッファの提供

2 番目のバッファ フォーマットを使用する場合は、バイト アレイを更新メソッドに渡す必要があります（『*アプリケーション応答測定 2.0 API ガイド*』を参照してください）。

構文：

```
myTranInstance.updateTranInstance(byteArray);
```

トランザクション インスタンスの停止 (arm_stop)

トランザクション インスタンスを停止するには、メトリック更新の有無を指定して停止をするかどうかを選択できます。

メトリックを更新するトランザクション インスタンスの停止

メトリックを更新してトランザクション インスタンスを停止するには、stopTranInstanceWithMetricUpdate メソッドをコールします。

構文：

```
myTranInstance.stopTranInstanceWithMetricUpdate
(transactionCompletionCode);
```

パラメータ：

トランザクション完了コードは、次のようになります。

ARMConstants.ARM_GOOD.	操作が正常に実行された場合に使用します。
ARMConstants.ARM_ABORT.	システムで致命的なエラーが発生した場合に使用します。
ARMConstants.ARM_FAILED.	トランザクションは正常に機能したが、結果が生成されなかったアプリケーションで使用します。

これらのメソッドは、C の関数 arm_stop に必要なパラメータを指定して使用します。

メトリックを更新しないトランザクション インスタンスの停止

メトリックを更新せずにトランザクション インスタンスを停止するには、メソッド stopTranInstance を使用できます。

構文：

```
myTranInstance.stopTranInstance(transactionCompletionCode);
```

完全なトランザクションの使用

Java ラッパーは arm_complete_transaction コールを使用できます。このコールを使用すると、指定された時間 (ナノ秒) だけ継続したトランザクションの終了をマーキングできます。これにより、ARM エージェント以外で測定されたトランザクション応答時間をリアルタイムに統合できます。

トランザクション インスタンスの終了を通知するだけでなく、トランザクション (UDM) に関する追加情報もオプションのデータ バッファで提供されます (jcomplete.java の例も参照してください)。

UDM を使用した完全なトランザクションの使用

構文 :

```
myTranInstance.completeTranWithUserData(status,responseTime;
```

パラメータ :

status	• ARMConstants.ARM_GOOD 操作が正常に実行された場合にこの値を使用します。 • ARMConstants.ARM_ABORT システムで致命的なエラーが発生した場合にこの値を使用します。 • ARMConstants.ARM_FAILED トランザクションは正常に機能したが、結果が生成されなかったアプリケーションでこの値を使用します。
responseTime	トランザクションの応答時間がナノ秒単位で表示されます。

UDM 使用しない完全なトランザクションの使用

構文 :

```
myTranInstance.completeTran(status,responseTime);
```

その他のドキュメント

Java クラスについての詳細は、/< インストール ディレクトリ >/examples/arm/ ディレクトリの doc フォルダを参照してください。ここには、すべての Java クラスが html 文書で含まれています。index.htm から始めます。

26 記録とトレース

記録とトレースの機構を使用すると、HP Operations エージェントの問題を診断してトラブルシューティングを実行できます。簡単に分析できるように、HP Operations エージェントではエラー、警告、一般メッセージをログ ファイルに保存します。

トレース機構は、エージェントのオペレーションで発生した特定の問題をトレースするときに役立ちます。また、トレース機構で生成されたトレース ファイルを HP サポートに送付して、さらに分析することもできます。

記録

HP Operations エージェントでは、警告メッセージ、エラー メッセージ、情報通知をノードの `System.txt` ファイルに書き込みます。`System.txt` ファイルの内容により、エージェントが正常に機能しているかどうかを判断できます。`System.txt` ファイルは、以下の場所にあります。

Windows

%ovdatadir%`log`

UNIX/Linux

`/var/opt/OV/log`

さらに、HP Operations エージェントは Performance Collection Component と coda のステータスの詳細を次のファイルに追加します。

Windows

- *%ovdatadir%*`\status.scope`
- *%ovdatadir%*`\status.perfalarm`
- *%ovdatadir%*`\status.ttd`
- *%ovdatadir%*`\status.mi`
- *%ovdatadir%*`\status.perfd`
- *%ovdatadir%*`\log\coda.txt`

UNIX/Linux

- `/var/opt/perf/status.scope`
- `/var/opt/perf/status.perfalarm`
- `/var/opt/perf/status.ttd`
- `/var/opt/perf/status.mi`
- `/var/opt/perf/status.perfd`
- *vMA のみ。*`/var/opt/perf/status.viserver`

- /var/opt/OV/log/coda.txt

記録ポリシーの設定

System.txt のファイル サイズが 1 MB になると、エージェントは新しい System.txt ファイルにメッセージを記録し始めます。System.txt ファイルのサイズを制限するように HP Operations エージェントのメッセージ記録ポリシーを設定できます。

<h2> デフォルトの記録ポリシーを変更するには :</h2>

1 ノードにログオンします。

2 以下の場所に移動します。

Windows

*%ovdatadir%*conf\xpl\log

UNIX/Linux

/var/opt/OV/conf/xpl/log

3 テキスト エディタで log.cfg ファイルを開きます。

4 System.txt ファイルのバイト サイズと文字数を制御しているパラメータは、BinSizeLimit および TextSizeLimit です。デフォルトでは、両方のパラメータは 1000000 (1 MB および 1000000 文字) に設定されています。目的の値にデフォルト値を変更します。

5 ファイルを保存します。

6 次のコマンドで、Operations Monitoring Component を再起動します。

a **ovc -kill**

b **ovc -start**

トレース

HP Operations エージェント アプリケーションのトレースを開始する前に、事前に必要な作業を実行する必要があります。これには、トレースする適切なアプリケーションの特定、トレースのタイプの設定、(必要に応じて) トレース設定ファイルの生成などがあります。

HP Operations エージェント プロセスのトレースを開始する前に、次の作業を実行します。

1 392 ページの「アプリケーションの特定」

2 393 ページの「トレース タイプの設定」

3 *オプション:* 396 ページの「設定ファイルの作成」

アプリケーションの特定

トレースする HP ソフトウェア アプリケーションを、管理対象システムで特定します。ovtrccfg -vc オプションを使用して、すべてのトレース対応アプリケーションの名前、および各トレース対応アプリケーションに定義されているコンポーネントとカテゴリを表示します。

または、ovtrcgui ユーティリティを使用して、トレース対応アプリケーションのリストを表示できます。ovtrcgui ユーティリティを使用して、トレース対応アプリケーションのリストを表示するには、次の手順を実行します。

1 ovtrcgui.exe ファイルを *%OvInstallDir%*\support ディレクトリから実行します。ovtrcgui ウィンドウが表示されます。

2 ovtrcgui ウィンドウで [**ファイル**] → [**新規**] → [**トレースの設定**] をクリックします。新しいトレース設定エディタが開きます。

3 ovtrcgui ウィンドウで [**編集**] → [**Add Application (アプリケーションを追加)**] をクリックします。または、エディタを右クリックし、[**Add Application (アプリケーションを追加)**] をクリックします。[Add Application (アプリケーションを追加)] ウィンドウが表示されます。

[Add Application (アプリケーションを追加)] ウィンドウに、利用できるトレース対応アプリケーションのリストが表示されます。

トレース タイプの設定

トレース機構を有効にする前に、アプリケーションに設定するトレースのタイプを決定して設定します。トレースのタイプを設定するには、次の手順を実行します。

設定するトレースのタイプ (静的または動的) を決定し、次の手順を実行します。

1 < ***データ ディレクトリ*** >/conf/xpl/trc/ の場所に移動します。

2 < ***アプリケーション名*** >.ini ファイルを見つけます。ファイルがあれば、393 ページの手順 3 に進みます。< ***アプリケーション名*** >.ini ファイルがなければ、次の手順を実行します。

- テキスト エディタで新しいファイルを作成します。
- ファイルに DoTrace、UpdateTemplate、DynamicTracing の各プロパティを、この順番で追加します。

  すべてのプロパティを 1 行に記述せず、各プロパティを新しい行に記述してください。次に例を示します。

  ```
  DoTrace=
  UpDateTemplate=
  DynamicTracing=
  ```

- ファイルを保存します。

3 < ***アプリケーション名*** >.ini ファイルをテキスト エディタで開きます。

4 **静的なトレース**を有効にするには、DoTrace プロパティを ON に設定し、DynamicTracing プロパティを OFF に設定します。

5 **動的なトレース**を有効にするには、DoTrace プロパティと DynamicTracing プロパティを ON に設定します。

6 UpdateTemplate プロパティを ON に設定します。

7 ファイルを保存します。

動的トレース設定の場合、アプリケーションを起動した後でもトレース機構を有効にできます。静的トレース設定の場合、アプリケーションを起動する前にトレース機構を有効にする必要があります。

トレース設定ファイルの概要

構文

TCF **バージョン** < *バージョン番号* >
APP: "< *アプリケーション名* >"
SINK: File "< *ファイル名* >" "maxfiles=[*1..100*];maxsize=[*0..1000*];"
TRACE: "< *コンポーネント名* >" "< *カテゴリ名* >" < *キーワードのリスト* >

構文の各行については、この後の章で説明します。

TCF Version

TCF バージョン行では、このファイルがトレース設定ファイルであることと、ファイルのバージョン番号を指定します。この行では大文字小文字が区別され、次に示すとおりに指定する必要があります。

構文 :

TCF Version < *バージョン番号* >

例 :

```
TCF Version 1.1
```

APP

アプリケーション行では、トレースするアプリケーションの名前を定義します。この行は APP で始まり、その後にコロン (:) とスペース、二重引用符 ("...") で囲まれたアプリケーション名が続きます。トレースには複数のアプリケーションを指定できます。このパターンを繰り返し記述し、トレースする各アプリケーションを指定します。

構文 :

APP: "< *アプリケーション名* >"

例 :

```
APP: "dbmanager"
APP: "opcmsg"
APP: "poller"
```

SINK

SINK 行ではトレース結果が出力されるターゲット ファイルを指定します。ターゲットは同じマシン上のファイルである必要があります。この行は SINK: FILE で開始する必要があります。この行の引数はスペースで区切ります。

SINK: FILE 行には 2 つの引数があります。

最初の引数はターゲット ファイルの名前です。これは二重引用符 ("...") で囲む必要があります。

2 つ目の引数は追加の SINK タイプ オプションです。このオプションは二重引用符 ("...") で囲み、各オプションの後ろにはセミコロン (;) を付ける必要があります。

SINK タイプ File のオプションは、次のとおりです。

- maxfiles=n
- maxsize=n

既存のトレース設定ファイルで、force オプションが使われていることがあります。このオプションは本バージョンのトレース ユーティリティではサポートされておらず、トレース機構への影響はありません。トレース機構を設定中、このオプションは無視してもかまいません。

maxfiles

maxfiles オプションには 1 ～ 100 の整数の値が続きます。このオプションを使用すると、保持する履歴トレース ログ ファイルの数を指定できます。アプリケーションがファイルにトレース記録を始めるたびに、以前のバックアップ ファイル (ある場合) の名前に「.001」が追加されたバックアップが作成されます。「.001」のファイルがすでにある場合、名前は「.002」になり、以後同様に名前が付けられます。現在のログ ファイルが最大サイズに到達すると、同じバックアップ スキームが適用されます。

maxsize

maxsize オプションは、0 ～ 1000 の整数または浮動小数値です。このオプションは、各ファイルに使用されるディスク領域の最大量をメガバイト (MB) で指定します。

トレース出力の最後のブロックがファイルに書き込まれることによって、指定したファイルの最大量が超過する場合、次の出力は現在の出力ファイルのバックアップを作成してからそのファイルを閉じ、新しい出力ファイルを作成します。値 0 は特別なケースであり、0 を指定するとファイルはディスク領域を使い果たすまで増大します。

トレースが出力されるターゲット ファイルの内容を表示するには、ovtrcmon または ovtrcgui のツールを使用します。標準のテキスト エディタを使用してファイルを開くと、フォーマットは正しく表示されません。

構文 :

```
SINK: File "< ファイル名 >" "maxfiles=[1..100];maxsize=[0..1000];"
```

例 :

```
SINK: File "C:\\TEMP\\Output.trc" "maxfiles=10;maxsize=100;"
```

TRACE

トレース行は、TRACE で始まり、コロン (:) とスペース ( ) が続く必要があります。行の引数はスペースで区切る必要があります。

最初の引数はトレース コンポーネント名です。これは二重引用符 ("...") で囲む必要があります。複数のコンポーネントを指定できます。

2 つ目の引数はトレース カテゴリ名です。これも二重引用符 ("...") で囲む必要があります。複数のコンポーネントを指定できます。コードに標準カテゴリの 1 つを使用している場合、ここで指定する文字列値にマッピングされます。標準カテゴリ定数を正確に文字列値にマッピングするには、適切な言語のドキュメント (C++、Java) を参照してください。

構文 :

```
TRACE: "< コンポーネント名 >" "< カテゴリ名 >" < キーワードのリスト >
```

例 :

```
TRACE: "database" "Parms" Error Info Warn
TRACE: "xpl.io" "Trace" Info
```

"*" は、コンポーネント名、カテゴリ名のいずれか、またはその両方に使用できます。"*" は、設定情報を直接ファイルから読み取るモードでアプリケーションを使用する場合に便利です。

アプリケーションが component A と category B の設定を決定しようとするとき、アプリケーションはこのペアの明示的なトレース定義が設定に含まれているかどうかを確認します。トレース定義がある場合、アプリケーションはそれらの設定を使用します。トレース定義がない場合、アプリケーションは component A および category * の設定があるかどうかを確認します。設定がある場合、アプリケーションはそれらの設定を使用します。設定がない場合、アプリケーションは component * および category * の設定があるかどうかを確認します。設定がある場合、アプリケーションはそれらの設定を使用します。設定がない場合、トレースは起動しません。

残りのパラメータはキーワード オプションの変数リストです。少なくとも Error、Info、Warn のうち 1 つのキーワードをリストに含める必要があります。サポート対象のキーワードには、次の種類があります。

表 12　　トレース キーワード

キーワード	説明
Error	エラーとしてマークされたトレースを有効にします。
Warn	警告としてマークされたトレースを有効にします。
Info	情報としてマークされたトレースを有効にします。
Support	通常のトレースを有効にします。トレースの出力には、情報、警告、エラーのメッセージが含まれます。問題のトラブルシューティングにはこのオプションをお勧めします。このオプションを指定するとトレース出力を記録するオーバーヘッドが最小になるため、長期のトレースを有効にできます。

サンプルのトレース設定ファイル

```
TCF Version 3.2
APP: "dbmanager"
SINK: File "C:\\TEMP\\Output.trc" "maxfiles=10;maxsize=100;"
TRACE: "DbManager" "Parms" Error Info Warn Developer
```

設定ファイルの作成

設定ファイルを使用せずにトレース機構を有効にする場合、この項を省略して 399 ページの「コマンド ライン ツールを使用したトレースの有効化とトレース メッセージの表示」に進みます。

トレース設定ファイルは、コマンド ライン ツールの ovtrccfg 、テキスト エディタ、ovtrcgui ユーティリティ (Windows ノードのみ) のいずれかを使用して作成できます。

コマンド ライン ツールの使用

次のコマンドを実行して、トレース設定ファイルを生成します。

ovtrccfg -app <*アプリケーション名*> **[-cm** <*コンポーネント名*>**]** **[-sink** <*ファイル名*>**]** **-gc** < ***設定ファイル名*** >

このコマンドにより、トレースするアプリケーションとコンポーネントの詳細が記された設定ファイルが作成されます。

テキスト エディタの使用

テキスト エディタを使用して手動で設定ファイルを作成する場合、次の手順を実行します。

1 テキスト エディタで新しいファイルを作成します。

2 設定ファイルの最初に、次の形式でバージョン番号を指定します。

TCF Version < ***バージョン番号*** >

次に例を示します。

```
TCF Version 1.0
```

3 トレースするアプリケーションを次の形式で指定します。

APP < *アプリケーション名* >

次に例を示します。

```
APP "coda"
```

4 トレース ファイルを格納する対象の場所を次の形式で指定します。

SINK: File "< *ファイル名* >" "maxfiles=< *最大ファイル数* >;maxsize=< *最大サイズ* >"

トレース ファイルの名前には、trc の拡張子を付ける必要があります。トレース出力ファイルの完全パスは、必ず次の形式で指定してください。

< *ドライブ名* >:\\< *ディレクトリ* >\\< *ファイル名* >

次に例を示します。

```
SINK: File "C:\\TEMP\\Output.trc" "maxfiles=10;maxsize=100;"
```

SINK パラメータに値を指定しないと、トレース機構はトレースするアプリケーションのホーム ディレクトリにトレース出力ファイルを配置します。

5 トレースするコンポーネントとカテゴリを次の形式で指定します。

TRACE: "< *コンポーネント名* >" "< *カテゴリ名* >" "< *キーワードのリスト* >"

複数のコンポーネントやカテゴリをトレースする場合、複数の TRACE 文を改行と共に追加します。

次に例を示します。

```
TRACE: "bbc.cb" "Parms" Error Info Warn

TRACE: "bbc.https.server" "Trace" Info
```

トレース キーワードの詳細は、396 ページの「トレース キーワード」を参照してください。

6 tcf の拡張子を付けてファイルを保存します。

トレース GUI の使用

Windows ノードでは、トレース GUI (ovtrcgui ユーティリティ) を使用して、トレース設定ファイルを作成できます。このユーティリティを使用してトレース設定ファイルを作成するには、次の手順を実行します。

1 ovtrcgui.exe ファイルを *%OvInstallDir%*\support ディレクトリから実行します。ovtrcgui ウィンドウが表示されます。

2 ovtrcgui ウィンドウで [**ファイル**] → [**新規**] → [**トレースの設定**] をクリックします。新しいトレース設定エディタが開きます。

3 ovtrcgui ウィンドウで [**編集**] → [**Add Application (アプリケーションを追加)**] をクリックします。または、エディタを右クリックし、[**Add Application (アプリケーションを追加)**] をクリックします。[Add Application (アプリケーションを追加)] ウィンドウが表示されます。

4 トレースするアプリケーションを選択し、[**OK**] をクリックします。[Configuration for *<application>* (アプリケーションの設定)] ウィンドウが表示されます。

[Configuration for *<application>* (アプリケーションの設定)] ウィンドウの [トレース] タブに、選択したアプリケーションのコンポーネントとカテゴリがすべて表示されます。デフォルトでは、すべてのコンポーネントとカテゴリのトレースは、`Off` に設定されています。

5 [トレース] タブで、コンポーネントとカテゴリのペアをクリックして、次のボタンを 1 つクリックします。

- [**Support**]: 情報の通知としてマークされたトレース メッセージを収集するには、これをクリックします。
- [**Developer**]: 情報の通知およびすべてのデベロッパー トレースとしてマークされたトレース メッセージを収集するには、これをクリックします。
- [**Max**]: トレースを最大レベルに設定するには、これをクリックします。
- [**Custom**]: [Custom] をクリックすると、[Modify Trace (トレースの変更)] ウィンドウが表示されます。

  [Modify Trace (トレースの変更)] ウィンドウで、カスタム オプションを選択し、目的のトレース レベルとオプションを選択し、[**OK**] をクリックします。

[Configuration for *<application>* (アプリケーションの設定)] ウィンドウで [**Off**] をクリックすると、コンポーネントとカテゴリのペアのトレースが無効になります。

6 [**OK**] をクリックします。

7 [SINK] タブに移動します。

8 [ファイル名] テキスト ボックスにトレース出力ファイルの名前を指定します。ファイルの拡張子は `.trc` である必要があります。

`.trc` ファイルの完全パスを指定します。

9 ドロップダウン リストから履歴ファイルの数を指定します (395 ページの「maxfiles」を参照)。

10 ドロップダウン リストから最大ファイル サイズを指定します (395 ページの「maxsize」を参照)。

11 [**適用**] をクリックします。

12 [**OK**] をクリックします。

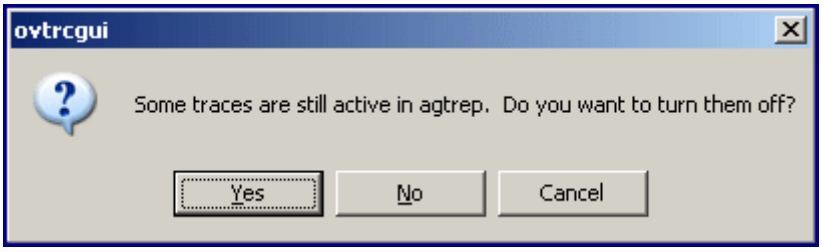
ovtrcgui ユーティリティで [**OK**] をクリックすると、トレース機構が有効になります。

13 [**ファイル**] → [**保存**] をクリックします。[名前を付けて保存] ダイアログ ボックスが表示されます。

14 [名前を付けて保存] ダイアログ ボックスで、適切な場所を参照して、.tcf 拡張子の付いたトレース設定ファイルの名前を [ファイル名] テキスト ボックスに指定し、[**保存**] をクリックします。

ovtrcgui ユーティリティは、指定した場所に指定した名前で新しいトレース設定ファイルを保存し、ファイルに指定されている構成に基づいてトレース機構を有効にします。ovtrcgui ユーティリティでトレース設定ファイルを開き、新しい設定の詳細を追加できます。

15 トレース設定エディタまたは ovtrcgui ウィンドウを閉じようとすると、次のメッセージが表示されます。



16 [**いいえ**] をクリックすると、トレース機構はシステムで設定したアプリケーションのトレースを続行します。[**はい**] をクリックすると、ovtrcgui ユーティリティは直ちにトレース機構を無効にします。

コマンド ライン ツールを使用したトレースの有効化とトレース メッセージの表示

次に説明する手順では、トレースを有効にするために必要な一般的な手順を取り上げます。トレース機構を有効にするには、次の手順を実行します。

1 ovtrccfg を使用して、トレース設定要求を行います。

ovtrccfg -cf < *設定ファイル名* >

ここで、< *設定ファイル名* > は、396 ページの「設定ファイルの作成」で作成された トレース設定ファイルの名前です。

トレース設定ファイルを使用しない場合、次のコマンドでトレースを有効にできます。

ovtrccfg -app < *アプリケーション* > **[-cm** < *コンポーネント* >**]**

2 静的トレース機構を設定する場合、トレースするアプリケーションを起動します。

3 トレースする問題を再現するために必要な、アプリケーション固有のコマンドを実行します。目的の動作が再現されたら、トレースを停止できます。

4 ovtrcmon を使用して、トレース監視要求を行います。

トレース メッセージを監視するには、次のコマンドの 1 つを実行するか、または追加の ovtrcmon コマンド オプションを使用して同様のコマンドを実行します。

- /opt/OV/bin/trace1.trc からトレース メッセージを監視し、トレース メッセージをテキスト形式でファイルに送信するには、次を実行します。

  ovtrcmon -fromfile /opt/OV/bin/trace1.trc -tofile /tmp/traceout.txt

- /opt/OV/bin/trace1.trc から、詳細形式でトレース メッセージを表示するには、次を実行します。

  ovtrcmon -fromfile /opt/OV/bin/trace1.trc -verbose

- /opt/OV/bin/trace1.trc から、詳細形式でトレース メッセージを表示し、トレース メッセージをファイルに出力するには、次を実行します。

  ovtrcmon -fromfile /opt/OV/bin/trace1.trc -short > /tmp/traces.trc

5 ovtrccfg を使用してトレースを停止または無効にするには、次のコマンドを実行します。

ovtrccfg -off

6 トレース設定ファイルとトレース出力ファイルを収集します。トレース メッセージを評価するか、またはファイルをパッケージ化して、評価のために HP ソフトウェア サポート オンラインに送付します。システムにはトレース出力ファイルの複数のバージョンがあります。Maxfiles オプションを使用すると、トレース機構で複数のトレース出力ファイルを生成できます。これらのファイルには、.trc の拡張子と、接尾辞 n が付いています (ここで n は 1 から 99999 の整数です)。

トレース GUI を使用したトレースの有効化とトレース メッセージの表示

Windows ノードでは、ovtrcgui ユーティリティを使用して、トレースを設定し、トレース メッセージを表示できます。

トレース機構の有効化

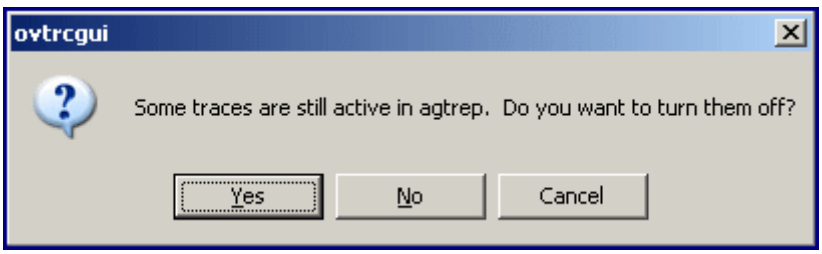
トレース設定ファイルを使用せずに、ovtrcgui ユーティリティを使用してトレース機構を有効にするには、次の手順を実行します。

1 397 ページの「トレース GUI の使用」の 397 ページの手順 1 から 398 ページの手順 6 を実行します。

2 トレース設定エディタを閉じます。

3 無題への変更を保存するかどうかを尋ねるプロンプトが表示されたら、[**いいえ**] をクリックします。

以下のメッセージが表示されます。



4 [**いいえ**] をクリックします。[**はい**] をクリックすると、ovtrcgui ユーティリティは直ちにトレース機構を無効にします。

ovtrcgui ユーティリティとトレース設定ファイルを使用してトレース機構を有効にするには、トレース設定ファイルがあるローカル システムの場所に移動し、トレース設定ファイルをダブルクリックします。または、ovtrcgui ユーティリティを開き、[**ファイル**] → [**開く**] をクリックして、トレース設定ファイルを選択し、[**開く**] をクリックします。

トレース メッセージの表示

ovtrcgui ユーティリティでトレース出力ファイルを表示するには、次の手順を実行します。

1. ovtrcgui.exe ファイルを *%OvInstallDir%*\support ディレクトリから実行します。ovtrcgui ウィンドウが表示されます。
2. [**ファイル**] → [**開く**] をクリックします。[開く] ダイアログ ボックスが表示されます。
3. トレース出力ファイルが配置されている場所に移動し、.trc ファイルを選択して、[**開く**] をクリックします。ovtrcgui ユーティリティに .trc ファイルの内容が表示されます。

   .trc ファイルの新しい行は、それぞれ新しいトレース メッセージを表します。
4. トレース メッセージをダブルクリックして、詳細を表示します。[Trace Properties (トレース プロパティ)] ウィンドウが表示されます。

   [Trace Properties (トレース プロパティ)] ウィンドウに、次の詳細が表示されます。

   - [Trace Info (トレース情報)]:
     - *[Severity (重要度)]:* トレース メッセージの重要度です。
     - *[Count (カウント)]:* メッセージのシリアル番号です。
     - *[Attributes (属性)]:* トレース メッセージの属性です。
     - *[構成要素]:* トレース メッセージを発行したコンポーネントの名前です。
     - *[カテゴリ]:* トレース対象アプリケーションによって割り当てられた任意の名前です。
   - [Process Info (プロセス情報)]:
     - *[Machine (マシン)]:* ノードのホスト名です。
     - *[アプリケーション]:* トレース対象アプリケーションの名前です。
     - *[PID]:* トレース対象アプリケーションのプロセス ID です。
     - *[TID]:* トレース対象アプリケーションのスレッド ID です。
   - [Time Info (時間情報)]:
     - *[Time (日時)]:* トレース メッセージの現地での日時です。
     - *[Tic count (Tic カウント)]:* 高分解能な経過時間です。
     - *[Tic difference (Tic 差)]:*
   - [Location (場所)]
     - *[ソース]:* トレースを生成するソースの行番号とファイル名です。
     - *[Stack (スタック)]:* トレース対象アプリケーションの呼び出しスタックの説明です。
5. [**次へ**] をクリックして、次のトレース メッセージを表示します。
6. すべてのトレース メッセージを表示したら、[**キャンセル**] をクリックします。

トレース リスト ビューの使用

デフォルトでは、ovtrcgui ユーティリティはトレース リスト ビューにあるトレース ファイルのトレース メッセージを表示します。トレース リスト ビューは、トレース メッセージを表形式で表示します。

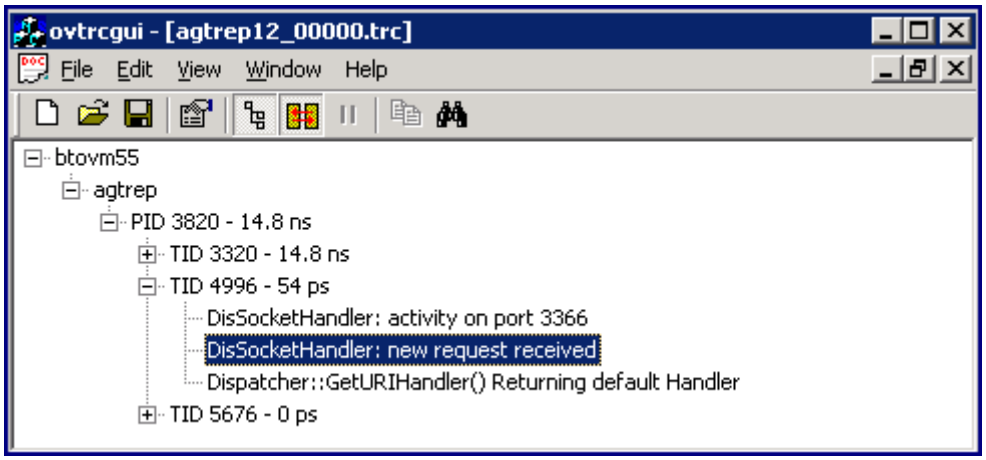
このトレース リスト ビューには、すべてのトレース メッセージが次のカラムと共に表示されます。

表 13　　トレース リスト ビュー

カラム	説明	
[Severity (重要度)]	トレース メッセージの重要度を示します。このビューでは次のアイコンを使用して、メッセージの重要度を表示します。	
	• 情報	
	• 警告	
	• エラー	
[アプリケーション]	トレース対象アプリケーションの名前を表示します。	
[構成要素]	トレース メッセージを生成したトレース対象アプリケーションのコンポーネントの名前を表示します。	
[カテゴリ]	トレース メッセージのカテゴリを表示します。	
[トレース]	トレース メッセージ テキストを表示します。	

手順のツリー ビューの使用

手順のツリー ビューでは、構造化された形式でトレース メッセージを表示できます。手順のツリー ビューでは、プロセス ID とスレッド ID に基づいてメッセージが並び替えられ、ツリー ビューの形式でデータが表示されます。



プロセス ID とスレッド ID を展開して、トレース メッセージを表示できます。トレース リスト ビューに戻るには、  をクリックします。

トレースのフィルタリング

ovtrcgui ユーティリティは、トレース設定ファイルの設定に基づいてトレース出力ファイルに記録されたすべてのトレース メッセージを表示します。利用可能なメッセージをフィルタリングすると、ovtrcgui コンソールに目的のメッセージのみを表示できます。利用可能なトレース メッセージをフィルタリングするには、次の手順を実行します。

1. ovtrcgui コンソールで [**表示**] → [**フィルタ**] をクリックします。[Filter (フィルタ)] ダイアログ ボックスが表示されます。
2. **[All Traces (すべてのトレース)]** を展開します。ダイアログ ボックスに、すべてのフィルタリング パラメータがツリー形式で表示されます。
3. パラメータを展開して、トレース メッセージをフィルタリングする項目を選択します。
4. [**OK**] をクリックします。ovtrcgui コンソールにはフィルタリングされたメッセージのみが表示されます。

27 トラブルシューティング

ここでは、HP Operations エージェント 11.00 を使用するときによく見られる問題の対策または回避策について説明します。この項で取り上げる領域は次のとおりです。

- Operations Monitoring Component
- Performance Collection Component
- RTMA

Operations Monitoring Component

- *問題:* Windows Server 2008 ノードで、opcmsga プロセスが機能せず、ovc コマンドで opcmsga のプロセスのステータスが aborted と表示される。

  対策:

  次のコマンドを実行して、OPC_RPC_ONLY 変数を TRUE に設定します。

  ovconfchg -ns eaagt -set OPC_RPC_ONLY TRUE

- *問題:* Windows ノードで、Perl スクリプトがポリシーから機能しない。

  原因: ポリシー内で Perl スクリプトを利用するには、PATH 設定変数に (HP Operations エージェントで提供されている) Perl があるディレクトリを含める必要がある。

  対策 :

  a 次のコマンドを実行して、PATH 設定変数を Perl ディレクトリに設定します。

  ovconfchg -ns ctrl.env -set PATH "%ovinstalldir%nonOV\perl\a\bin"

  b 次のコマンドを実行して、エージェントを再起動します。

  – **ovc -kill**

  – **ovc -start**

- *問題:* ovconfchg コマンドから変数の値を変更した後、その変更が有効にならない。

  原因 1:

  その変数はエージェントを再起動する必要がある。

  対策 1:

  次のコマンドを実行して、エージェントを再起動します。

  a **ovc -kill**

  b **ovc -start**

原因 2:

ノードに配布されている ConfigFile ポリシーにより、その変数が特定の値に設定される。

対策:

配布されている ConfigFile ポリシーに、設定変数を特定の値に設定するコマンドが含まれている場合、ovconfchg コマンドからの変更は有効になりません。ConfigFile ポリシーをノードから削除するか、または変数を目的の値に設定するコマンドを含めるようにポリシーを変更する必要があります。

原因 3:

ノードで利用可能なプロファイルまたはジョブ ファイルによって、変更が上書きされる。

対策:

プロファイルまたはジョブ ファイルをノードで開き、それらのファイルに競合する変数の設定が含まれていないことを確認します。

- *問題:* 設定変数 SNMP_SESSION_MODE の値を変更した後、ovc によって opctrapi プロセスのステータスが Aborted と表示される。

  原因:

  設定変数 SNMP_SESSION_MODE の値を変更した後、HP Operations エージェントは opctrapi を再起動しようとします。場合によっては、opctrapi の再起動プロセスが失敗することがあります。

  対策:

  次のコマンドを実行して、opctrapi を再起動します。

  ovc -start opctrapi

Performance Collection Component

- *問題:* HP-UX 11.11 システムで status.midaemon ファイルに次のエラーが表示される。

  mi_shared - MI initialization failed (status 28)

  原因: midaemon バイナリのページ サイズが大きい。

  対策: この問題を解決するには、次の手順を実行します。

  a. root ユーザーとしてシステムにログオンします。

  b. 次のコマンドを実行して、HP Operations エージェントを停止します。

     /opt/OV/bin/opcagt -stop

  c. 次のコマンドを実行して、midaemon のバックアップを作成します。

     cp /opt/perf/bin/midaemon /opt/perf/bin/midaemon.backup

  d. 次のコマンドを実行して、midaemon バイナリのページ サイズを 4K に減らします。

     chatr +pi 4K /opt/perf/bin/midaemon

  e. 次のコマンドを実行して、HP Operations エージェントを起動します。

     /opt/OV/bin/opcagt -start

- HP Operations エージェントのインストール後、トレース機構を有効にすると、System.txt ファイルに次のエラー メッセージが表示される。

  Scope data source initialization failed

  対策: このエラーは無視してください。

RTMA

- *問題:* vSphere Management Assistant (vMA) ノードで、rtmd プロセスが機能せず、ovc コマンドで rtmd のプロセスのステータスが aborted と表示される。

  原因: rtmd プロセスが IP アドレスに対してシステムのホスト名を解決できない。

  対策:

  a root 権限を使用して、ノードにログオンします。

  b テキスト エディタで、/etc ディレクトリから hosts ファイルを開きます。

  c localhost という文字が記されている行を見つけます。

  d # 文字を行の先頭から削除します。

  e ファイルを保存します。

  f 次のコマンドを実行し、すべてのプロセスを起動します。

  ovc -restart

- *問題:* HP Performance Manager の診断ビューでデータにアクセスできない。

  原因: rtmd プロセスが実行されていない。

  対策: rtmd プロセスが HP Operations エージェント ノードで実行されているかどうかを確認するには、**ovc -status rtmd** を実行します。rtmd プロセスを起動するには、**ovc -start rtmd** を実行します。

- *問題:* HP-UX 11.11 システムで status.perfd ファイルに次のエラーが表示される。

  mi_shared - MI initialization failed (status 28)

  原因: perfd バイナリのページ サイズが大きい。

  対策: この問題を解決するには、次の手順を実行します。

  a root ユーザーとしてシステムにログオンします。

  b 次のコマンドを実行して、HP Operations エージェントを停止します。

  /opt/OV/bin/opcagt -stop

  c 次のコマンドを実行して、perfd のバックアップを作成します。

  cp /opt/perf/bin/perfd /opt/perf/bin/perfd.backup

  d 次のコマンドを実行して、perfd バイナリのページ サイズを 4K に減らします。

  chatr +pi 4K /opt/perf/bin/perfd

  e 次のコマンドを実行して、HP Operations エージェントを起動します。

  /opt/OV/bin/opcagt -start

索引

数字

1 時間あたりのレコード , 267, 283

A

agdb, 152

agdbserver, 152

agdb データベース , 152

agsysdb, 152

alarmdef
  変更 , 282

alarmdef ファイル , 68, 69, 171, 178, 282

ALARM 文、アラーム構文 , 160

ALARM 文内での複合動作 , 163

ALERT 文、アラーム構文 , 165

ALIAS 文、アラーム構文 , 173

analyze コマンド、utility プログラム , 68

application コマンド、extract プログラム , 115

argv1 キーワード、parm ファイル , 33

arm.h インクルード ファイル , 381

ARM API
  scopeux の組み込み , 373
  エラー メッセージ , 363
  関数コール , 338
  共有ライブラリ , 355
  ステータス リターン , 345
  トランザクション追跡 , 341

ARM API コール
  arm_complete_transaction, 343
  arm_getid コール , 384
  arm_init コール , 383
  arm_start コール , 386

ARM 組み込みアプリケーションの例 , 343

ARM 使用のガイドライン , 351

ARM 相関係数 , 352

ASCII フォーマット、エクスポート ファイル , 100, 217

ASCII レコード フォーマット , 104

B

binary フォーマット、エクスポート ファイル , 100, 217

binary レコード フォーマット , 105

C

checkdef コマンド、utility プログラム , 69

class コマンド、extract プログラム , 116

cmd パラメータ、parm ファイル , 34

configuration コマンド、extract プログラム , 117

CPU オーバーヘッド , 352

cpu オプション , 25

cpu コマンド、extract プログラム , 117

C の関数
  arm_stop, 388

D

datafile フォーマット、エクスポート ファイル , 100, 217

datafile レコード フォーマット , 104

data type パラメータ、エクスポート テンプレート ファイル , 101

detail コマンド、utility プログラム , 70

disk オプション , 25

disk コマンド、extract プログラム , 118

dsilog
  構文 , 283
  スクリプトの記述 , 296
  入力 , 283
  ログ処理 , 283, 298

dsilog スクリプトの記述 , 296
  推奨する dsilog スクリプトの例 , 296
  問題となる dsilog スクリプトの例 , 296

dsilog プログラム , 255

dsilog へのデータのパイピング , 283
dsilog への入力 , 283
DSI、データ ソース統合を参照
DSI データの管理 , 293
DSI データへのアクセス , 292
DSI の使用例 , 295
  dsilog スクリプトの記述 , 296
  vmstat データの記録 , 297
  システム ユーザー数の記録 , 315
  単一ファイルの sar データの記録 , 299
  複数オプションの sar データの記録 , 309
  複数ファイルの sar データの記録 , 303
DSI ログ ファイル , 121, 125
DSI ログ ファイル データのエクスポート , 125

E

EXEC 文、アラーム構文 , 166
exit コマンド、extract プログラム , 119
exit コマンド、utility プログラム , 71
export コマンド、extract プログラム , 96, 119
export コマンドのデフォルト出力ファイル , 120
extract
  トランザクション データとの使用 , 358
extract コマンド、extract プログラム , 121
extract のコマンド
  application, 115
  class, 116
  configuration, 117
  cpu, 117
  disk, 118
  exit, 119
  export, 96, 119
  extract, 121
  filesystem, 123
  global, 124
  guide, 125
  help, 125
  list, 126
  lvolume, 128
  menu, 129
  monthly, 130
  output, 132
  process, 134
  quit, 134
  report, 135
  sh, 135
  shift, 136
  show, 137
  start, 138
  stop, 140
  weekdays, 142
  weekly, 143
  yearly, 145
extract プログラム , 89, 292
  コマンド , 111
  コマンド ライン インターフェイス , 91
  コマンド ラインの引数 , 92
  実行 , 90
  対話型モードとバッチ モードの比較 , 90

F

fifo, 283
filesystem コマンド、extract プログラム , 123
flush, 30
format パラメータ
  エクスポート テンプレート ファイル , 100

G

gapapp, 28
GlancePlus
  アプリケーション応答測定 2.0 のサポート , 342
  トランザクション データのアラーム , 360, 361
  トランザクション データの表示 , 338
  トランザクション データの分析 , 360
  トランザクション データの監視 , 360
  パフォーマンスのボトルネックの特定 , 338
global コマンド、extract プログラム , 124
group パラメータ、parm ファイル , 35
guide コマンド、extract プログラム , 125
guide コマンド、utility プログラム , 71

H

headings パラメータ、エクスポート テンプレート ファイル , 100, 217
help コマンド、extract コマンド , 125
help コマンド、utility プログラム , 72
HP Network Node Manager, 152

I

ID パラメータ
  parm ファイル , 23
IF 文、アラーム構文 , 168

INCLUDE 文、アラーム構文 , 170
items パラメータ、エクスポート テンプレート ファイル , 101

J

javaarg パラメータ、parm ファイル , 30
Java ラッパー , 383
  アプリケーションの設定 , 383
  完全なトランザクションの使用 , 389
  トランザクション インスタンス データの更新 , 387
  トランザクション インスタンスの開始 , 386
  トランザクション インスタンスの停止 , 388
  トランザクションの設定 , 384
  ドキュメント , 389
  例 , 383

L

layout パラメータ、エクスポート テンプレート ファイル , 101
libarm, 355, 375
libarmNOP, 383
list コマンド、extract プログラム , 126
list コマンド、utility プログラム , 72
logappl ファイル , 23
  PRM グループ , 23
logdev ファイル , 24
logfile コマンド、utility プログラム , 73
logglob ファイル , 23, 145
logproc ファイル , 23
log パラメータ、parm ファイル , 23
LOOP 文、アラーム構文 , 169
lvolume コマンド、extract プログラム , 128

M

maintenance 時間、parm ファイル , 29
mainttime パラメータ、parm ファイル , 29, 41
Measurement Interface daemon、midaemon を参照 , 341
memory オプション , 25
menu コマンド
  extract プログラム , 129
  utility プログラム , 75
MIB ID, 46
midaemon, 341, 357
  midaemon 共有メモリー セグメントのサイズの指定 , 358
  エラー , 346
  エラー メッセージ , 363
  共有メモリー セグメント , 346, 357
  メモリー オーバーヘッド , 353
midaemon 共有メモリー セグメントのサイズの指定 , 358
missing パラメータ、エクスポート テンプレート ファイル , 101, 217
monthly コマンド、extract プログラム , 130
mwa スクリプト , 39, 40

N

Netif Name Record, 110
Network Node Manager, 282
nokilled オプション , 26

O

OID, 46
Operations Manager, 152, 283
or パラメータ、parm ファイル , 35
output コマンド、extract プログラム , 132
output パラメータ、エクスポート テンプレート ファイル , 101
ovpa restart, 357
ovpa start, 357
ovpa stop scope, 356

P

parmfile コマンド、utility プログラム , 76
parm ファイル
  flush, 30
  gapapp, 28
  Performance Manager および Performance Agent の修正 , 357
  subprocinterval パラメータ , 27
  アプリケーション定義パラメータ , 31
  構成 , 232
  構文の確認 , 76
  データ記録間隔の設定 , 36
  パラメータ , 22
  変更 , 18, 233, 242

parm ファイル アプリケーション キーワード
argv1, 33
parm ファイル アプリケーション パラメータ
cmd, 34
parm ファイルの修正 , 357
parm ファイル パラメータ
group, 35
ID, 23
javaarg, 30
log, 23
mainttime, 29, 41
or, 35
priority, 35
scopetransactions, 27
size, 28
アプリケーション名 , 32
ファイル , 33
perfalarm, 152, 171
perflbd.mwc ファイル
フォーマット , 237
Performance Agent
extract プログラム , 89
parm ファイルの修正 , 357
utility プログラム , 55
アプリケーション応答測定 2.0 のサポート , 342
開始 , 357
再起動 , 357
ステータス確認 , 242
データ型 , 212
データ収集と記録 , 359
トランザクション データのエクスポート , 359
トランザクション データの表示 , 338
トランザクション データの抽出 , 359
要約レベル , 212
Performance Agent でのデータの収集 , 359
Performance Agent でのトランザクション データのエクスポート , 359
Performance Agent でのトランザクション データの走査 , 359
Performance Agent でのトランザクション データの抽出 , 359
Performance Agent のステータス確認 , 242
Performance Manager
DSI データの表示 , 292
トランザクション データのアラーム , 360, 361
トランザクション データの表示 , 338
トランザクション データの分析 , 360
Performance Manager でのデータの表示 , 292
perfstat コマンド , 17
PRINT 文、アラーム構文 , 167
priority パラメータ、parm ファイル , 35
PRM アプリケーション記録モード , 32
PRM グループ
APP_NAME_PRM_GROUPNAME, 23
proccmd, 31
process コマンド、extract プログラム , 134

Q

quit コマンド
extract プログラム , 134
utility プログラム , 76

R

range キーワード , 348
range または slo の変更、ttd.conf, 347, 359
report コマンド、extract プログラム , 135
report パラメータ、エクスポート テンプレート ファイル , 100
reptall ファイル , 97, 98
reptfile.mwr ファイル , 218
reptfile ファイル , 97, 135
repthist ファイル , 98
resize コマンド
utility プログラム , 56, 77
デフォルトのサイズ変更パラメータ , 79
レポート , 80

S

sar
単一ファイルの sar データの記録の例 , 299
複数オプションの sar データの記録の例 , 309
複数ファイルの sar データの記録 , 303
scan コマンド、utility プログラム , 82
scopetransactions パラメータ、parm ファイル , 27
scopeux, 247
ARM API コールへの組み込み , 373
終了 , 40
停止と再起動 , 345, 347
scopeux の停止と再起動 , 345, 347
SCOPE デフォルト データ ソース , 159, 171

SDL
クラスの仕様のエラー メッセージの接頭辞 , 278
sdlcomp, 297
コンパイラ , 297
sdlcomp コンパイラ , 252
sdlcomp のトラブルシューティング , 281
sdlgendata, 286
sdlutil, 293, 298
構文 , 293
separator パラメータ、エクスポート テンプレート ファイル , 100
shift コマンド、extract プログラム , 136
shortlived オプション , 26
show コマンド
extract プログラム , 137
utility プログラム , 84
sh コマンド
extract プログラム , 135
utility プログラム , 83
size パラメータ、parm ファイル , 28
SLO
サービスレベルの目標を参照 , 337
slo キーワード , 349
SLO の管理 , 337
SNMP
トラップ , 152
ノード , 152
SNMP_COMMUNITY, 46
SNMP_COMMUNITY_LIST, 46
SNMP トラップ , 282
SNMP トラップの送信 , 152
start コマンド
extract プログラム , 138
utility プログラム , 85
パラメータ , 85
status.mi, 363
status.scope ファイル , 17
stop コマンド
extract プログラム , 140
utility プログラム , 86
パラメータ , 86
summary パラメータ、エクスポート テンプレート ファイル , 100, 217
SYMPTOM 文、アラーム構文 , 174

T

threshold パラメータ、parm ファイル
cpu オプション , 25
disk オプション , 25
memory オプション , 25
nokilled オプション , 26
nonew オプション , 25
shortlived オプション , 26
Tip of the Day, 231
Transaction Tracker
アプリケーションの組み込み , 338
tran キーワード , 348
ttd, 341, 357
ttd.conf, 341, 347
range または slo の変更 , 347, 359
新しいトランザクションの追加 , 347
カスタマイズ , 358
キーワード , 348
デフォルト , 347, 348, 349, 358
トランザクションの追加 , 359
フォーマット , 349
例 , 369
ttd.conf ファイルのカスタマイズ , 358
ttdconf.mwc ファイル , 239

U

UNIX カーネル パラメータ , 283
UNIX タイムスタンプ , 270
USE 文、アラーム構文 , 171
utility 走査レポート
parm ファイルのアプリケーション追加通知および削除通知 , 62
parm ファイルのグローバル変更通知 , 62
scopeux のオフタイム通知 , 62
アプリケーション固有の要約レポート , 62
アプリケーションの全体的な要約 , 64
コレクタの適用範囲の要約 , 65
初期 parm ファイルのアプリケーション定義 , 61
初期 parm ファイルのグローバル情報 , 60
走査の開始および終了 , 64
プロセスの記録理由の要約 , 63
ログ ファイルの空きスペースの要約 , 66
ログ ファイルの内容の要約 , 65
utility のコマンド

analyze, 68
checkdef, 69
detail, 70
exit, 71
guide, 71
help, 72
list, 72
logfile, 73
menu, 75
parmfile, 76
quit, 76
resize, 56, 77
scan, 82
sh, 83
show, 84
start, 85
stop, 86

utility プログラム , 55, 67, 154
コマンド ライン インターフェイス , 55, 57
コマンド ラインの引数 , 58
シェル コマンドの入力 , 83
実行 , 55
対話型プログラムの例 , 56
対話型モード , 56
対話型モードとバッチ モードの比較 , 55
バッチ モード , 56
バッチ モードの例 , 56

V

VAR 文、アラーム構文 , 172

vmstat
vmstat データの記録の例 , 297

W

weekdays コマンド、extract プログラム , 142

weekly コマンド、extract プログラム , 143

who ワードカウントの例 , 315

WK1 (表計算シート) フォーマット、エクスポート ファイル , 217

WK1 フォーマット、エクスポート ファイル , 100

Y

yearly コマンド、extract プログラム , 145

Z

zone_app, 31

あ

アーカイブ、アーカイブ データ , 225

アーカイブ、データの追加 , 225

アーカイブ期間 , 225

アーカイブ処理、管理 , 43

アーカイブに関するヒント , 226

アクション , 282

新しいアプリケーションの追加 , 347

新しいトランザクションの追加、ttd.conf, 347, 359

アプリケーション
ttd.conf での定義 , 349
新規追加 , 347

アプリケーション応答測定
2.0 Software Developers Kit (SDK), 342
2.0 の機能 , 342
2.0 ロギング エージェント , 342
library (libarm), 375
アプリケーションの例 , 343
オーバーヘッドの考慮事項 , 351
共有ライブラリの取得 , 355
使用のガイドライン , 351
非操作用ライブラリ (libarmNOP), 383
利点 , 336

アプリケーション定義パラメータ、parm ファイル , 31

アプリケーションの LOOP 文、アラーム構文 , 170

アプリケーションの実行 , 357

アプリケーションの配置 , 357

アプリケーションの例 , 337

アプリケーション名パラメータ、parm ファイル , 32

アプリケーション名レコード , 109

アプリケーション メトリック、アラーム定義 , 159

アラート , 282

アラーム
Operations Manager へのメッセージの送信 , 152
構成 , 283
処理 , 282
ジェネレータ , 282
定義 , 282
ローカル アクション , 153

アラーム構文 , 157
ALARM 文 , 160
ALERT 文 , 165
ALIAS 文 , 173

EXEC 文 , 166
IF 文 , 168
INCLUDE 文 , 170
LOOP 文 , 169
PRINT 文 , 167
SYMPTOM 文 , 174
USE 文 , 171
VAR 文 , 172
共通要素 , 157
コメント , 158
式 , 159
条件 , 158, 162, 168
定数 , 159
表記 , 157
複合文 , 158
変数 , 172
メッセージ , 160
メトリック名 , 159
リファレンス , 157
アラーム構文内の複合文 , 158
アラーム構文内のメッセージ , 160
アラーム構文内のメトリック名 , 159, 173
アラーム処理 , 282
アラーム処理のエラー , 153
アラーム ジェネレータ , 152
アラーム情報の送信 , 282
アラーム定義
アプリケーション メトリック , 159
カスタマイズ , 178
構成 , 235
構文の確認 , 69
コンポーネント , 156
ファイル , 68, 226, 235
変更 , 236
メトリック名 , 159
例 , 176
アラーム定義内のメトリック , 282
アラーム定義での DSI メトリック , 282
アラームの構成 , 283
アラームの定義
DSI メトリック名 , 282
アラーム メッセージの送信 , 152, 165

い

インデックス インターバル、クラス , 260

え

エクスポート関数
process, 96
エクスポート テンプレート ファイル , 97
エクスポート テンプレート ファイルの構文 , 99
概要 , 96
使用方法 , 102
タスクのサンプル , 97
データ ファイル , 98
エクスポート テンプレート ファイル
data type, 101
format, 100
headings, 100
items, 101
layout, 101
missing, 101
output, 101
report, 100
separator, 100
summary, 100
エクスポート ファイルのタイトル , 102
クイック テンプレートの作成 , 221
欠落値 , 217
構成 , 222, 223
構文 , 99
パラメータ , 100
ファイル フォーマット , 217
フィールド区切り記号 , 217
複数レイアウト , 218
分単位の要約 , 217
ヘッダー , 217
エクスポート ファイル
ASCII フォーマット , 217
binary フォーマット , 217
datafile フォーマット , 217
WK1 (表計算シート) フォーマット , 217
属性 , 216, 218
タイトル , 102
デフォルトのファイル名 , 219
エクスポート ファイルのタイトル , 218
エクスポート ファイルのメトリックの区切り , 217
エクスポートまたは抽出するデータの範囲 , 213
エスケープ文字 , 260, 261, 270
エラー、アラーム処理 , 153
エラーの対応の考慮事項 , 357
エラー メッセージ , 317
ARM API から , 363
midaemon から , 363

お

オーバーフローの状態 , 358

オーバーヘッド
- ARM 使用での考慮事項 , 351
- CPU, 352
- ディスク I/O, 352
- メモリー , 353

オプションによる要約 , 271

か

カーネル パラメータ , 283

開始
- Performance Agent, 357
- ttd, 345

拡張収集ビルダ / マネージャ , 244
- 使用のヒント , 244

カスタマイズしたエクスポート テンプレート ファイル , 98, 222

カスタム グラフまたはレポートの作成 , 102

可変データ ロギング , 36

カラム ヘッダー、エクスポート ファイルでの指定 , 217

ガイド モード
- extract, 125
- utility, 71

概要
- データ ソース統合 , 247

き

キーワード
- range, 348, 358
- slo, 349, 358
- tran, 348, 358

共有メモリー セグメント、midaemon, 346, 357

共有ライブラリ , 383

記録されたデータ、エクスポート , 292

記録されたデータのエクスポート , 292, 299

く

クイック エクスポート テンプレートの作成 , 221

区切り記号 , 272, 286

クラス
- 1 時間あたりのレコード , 267
- ID の要件 , 259
- sdlutil によるリスト , 293
- インデックス インターバル , 260
- 記述のデフォルト , 259
- 構文 , 259
- 説明 , 252
- 定義 , 257
- 名前の要件 , 259
- 文 , 259
- 容量 , 265, 268
- ラベル , 260
- ロール インターバル , 260

クラスの仕様
- sdlutil による再作成 , 293
- コンパイル , 297, 301, 307
- 作成 , 297, 300, 303
- テスト , 286
- 変更 , 291
- メトリックの定義 , 269

クラスの仕様のコンパイル , 297, 301, 307

クラスの仕様ファイルの変更 , 291

け

傾向分析 , 336

こ

構成
- parm ファイル , 232
- ttdconf.mwc ファイル , 239
- アラーム定義ファイル , 235
- エクスポート テンプレート ファイル , 222, 223
- 収集パラメータ , 232
- トランザクション , 239

構成ユーザー オプション , 231

構文
- dsilog, 283
- sdlutil, 293
- エクスポート , 292

コマンド
- extract プログラム , 111
- perfstat, 17
- utility プログラム , 67

コマンドの省略
- extract, 111
- utility, 67

コマンド ライン インターフェイス
- extract プログラム , 90, 91
- utility プログラム , 55, 57

コマンド ラインの引数

extract プログラム , 92
utility プログラム , 57
コメント、アラーム構文内の使用方法 , 158
固有のトランザクションの制限 , 358
コンパイラ出力、サンプル , 279
コンパイラ出力のサンプル , 279

さ

サービスレベルの目標
管理 , 337
定義 , 356
サイズ変更
タスク , 42
ログ ファイル , 77
作成
クラスの仕様 , 248
ログ ファイル , 252
サポート
アプリケーション応答測定 2.0, 342

し

式、アラーム構文内 , 159
指定済みのパイプ , 283
収集パラメータ
構成 , 232
変更 , 233, 242
終了
extract プログラム , 119, 134
scopeux, 40
ttd, 345
utility のコマンド , 76
utility プログラム , 71
アプリケーション , 345
データ収集 , 39
小数点以下の桁数、メトリック , 271
仕様への受信データのマッピング , 289
実行
extract プログラム , 90
ttd, 345
utility プログラム , 55
アプリケーション , 357
条件
アラーム構文 , 158, 162, 168
状態の永続化 , 47

す

数値フォーマット オプション , 289
数値メトリック、フォーマット ファイル , 289
ステータス バー、表示 , 231

せ

精度 , 271
メトリック , 271
設定
監視エージェント , 45
コミュニティ文字列 , 45

そ

相関係数データ収集 , 357
測定、範囲の定義 , 358

た

対象プロセス , 23, 42
タイムスタンプ , 270
抑制 , 283
対話型モード
extract プログラム , 91
utility プログラム , 56

ち

長期的な傾向の分析 , 336

つ

ツールのヒント、表示 , 231
ツールバー、表示 , 231

て

定義
サービスレベルの目標 , 356, 358
測定範囲 , 358
定数、アラーム構文内 , 159
テキスト メトリック
指定 , 272
フォーマット ファイル , 289
テキスト メトリックの長さ , 272
テスト
クラスの仕様 , 286
ログ処理 , 286
ディスク I/O、オーバーヘッド , 352

ディスク デバイス名レコード , 109
データ
  アクセス , 292
  エクスポート , 292
  管理 , 293
  記録 , 248
  収集 , 248
データ型 , 97, 212, 371
データ型のエクスポート , 97
データ記録間隔の設定 , 36
データ収集
  管理 , 41
  終了 , 39
データ ソース , 171
データ ソース統合
  DSI の使用例 , 295
  エラー メッセージ , 317
  概要 , 247
  テスト , 286
  動作 , 247
データの記録
  dsilog プログラムの実行 , 255
データの記録の拒否 , 289
データの分析
  GlancePlus の使用 , 360
  Performance Manager の使用 , 360
デフォルト
  1 時間あたりのレコード , 267
  区切り記号 , 272, 286
  クラスの仕様 , 259
  クラス ラベル , 260
  メトリック , 270
  要約レベル , 267, 283
デフォルト log パラメータ、parm ファイル , 357
デフォルトの ttd.conf ファイル , 347, 348, 349
デフォルトのエクスポート ファイル名 , 219

と

統計、sdlutil によるリスト , 293
トランザクション
  ttd.conf への新規追加 , 347
  データ , 336
  名前 , 358
  メトリック , 365
トランザクション構成 , 239
トランザクション構成ファイル , 347
トランザクション構成ファイル、ttd.conf を参照 , 341
トランザクション追跡
  アプリケーションの設定 , 355
  失われたデータ , 358
  エラーの対応 , 357
  開始 , 345
  概要 , 337
  固有のトランザクションの制限 , 358
  コンポーネント , 341
  テクニカルリファレンス , 341
  データの表示 , 338
  利点 , 336
  例 , 367
トランザクション追跡登録デーモン、ttd を参照 , 341
トランザクション追跡の概要 , 341
トランザクション追跡のコンポーネント , 341
トランザクション データのアラーム
  GlancePlus の使用 , 360, 361
  Performance Agent の使用 , 361
  Performance Manager の使用 , 360
トランザクション データの記録 , 338
トランザクション データの表示
  GlancePlus の使用 , 338
  Performance Agent の使用 , 338
  Performance Manager の使用 , 338
  概要 , 338
トランザクション パフォーマンス データの監視 , 338
トランザクション名 , 348
トランザクション名の指定 , 348, 358
トランザクション名レコード , 109
トランザクション メトリック , 365
トレース
  設定 , 392
  設定ファイル
    SINK, 396
    アプリケーション , 395
    構文バージョン , 395
    トレース , 396
  動的
    有効化 , 403
トレースの設定 , 392

な

生ログ ファイル

スペースの管理 , 77
名前 , 73

は

バージョン情報、表示 , 293
バイナリ ヘッダー レコード レイアウト , 105
パフォーマンス カウンタ、収集の構築 , 244
パフォーマンス カウンタの収集の構築 , 244
パフォーマンスのボトルネックの特定 , 338
パラメータ
subprocinterval, 27

ひ

非操作用ライブラリ , 383
表記、アラーム構文 , 157

ふ

ファイル
alarmdef, 68, 69, 178, 282
logappl, 23
logdev, 24
logglob, 23
logproc, 23
parm, 232
reptall, 97, 98
reptfile, 97, 135
reptfile.mwr, 218
repthist, 98
status.scope, 17
ttdconf.mwc, 239
アラーム定義 , 68, 226, 235
エクスポート テンプレート , 97
収集パラメータ , 232
ファイル属性、エクスポート , 216
ファイル パラメータ、parm ファイル , 33
ファイル フォーマット パラメータ、エクスポート テンプレート ファイル , 217
ファイル名、デフォルトのエクスポート ファイル , 219
フィールド区切り記号パラメータ、エクスポート テンプレート ファイル , 217
フォーマット ファイル , 283, 289
複数レイアウト、エクスポート ファイルでの指定 , 218
複数レイアウト パラメータ、エクスポート テンプレート ファイル , 218
分単位の要約、エクスポート ファイルでの指定 , 217
分析
履歴ログ ファイル データ , 68, 154
ログ ファイル , 68, 154
プラットフォームごとの C コンパイラ オプションの例 , 381

へ

ヘルプへのアクセス
extract プログラム , 125
utility プログラム , 72
変更
alarmdef ファイル , 282
parm ファイル , 18, 233, 242
アラーム定義 , 236
クラス定義 , 291
収集パラメータ , 18, 233, 242
変数、アラーム構文 , 172

め

メトリック , 365
ID の要件 , 269
sdlutil によるリスト , 293
キーワード , 269
順序 , 270
精度 , 271
説明 , 252
定義 , 269
テキスト , 272
デフォルト , 270
名前の再使用 , 270
名前の要件 , 269
要約方法 , 271
ラベル , 270
ラベルの要件 , 270
メトリック、エクスポート対象の選択 , 222, 223
メトリックの順序、変更 , 289
メトリック名の再使用 , 270
メモリー オーバーヘッド , 353

ゆ

ユーザー オプション、構成 , 231
ユーザー定義メトリック , 372
ユーティリティ、sdlutil, 293

よ

要約方法 , 271

要約レベル , 212, 283
　デフォルト , 267

容量 , 265
　文 , 268

ら

ライブラリ
　libarm の使用 , 355

ラベル
　クラス , 260
　メトリック , 270

り

リソース データ収集 , 357

履歴ログ ファイル データ内のアラーム状態 , 68, 154, 226, 227

履歴ログ ファイル データのアラーム条件のレポート , 227

履歴ログ ファイル データの分析 , 226

れ

例
　ttd.conf, 369
　トランザクション追跡 , 367

レコード フォーマット
　ASCII, 104
　binary, 105
　datafile, 104

ろ

ローカル アクション
　アラーム , 167
　実行 , 153

ローカル アクションの実行 , 153

ロール
　アクション , 261
　アクションの例 , 262
　インターバル , 260

ロギング エージェント、アプリケーション応答測定 2.0, 342

ログ処理 , 283, 298
　dsilog, 298
　テスト , 286

ログ処理の開始 , 283

ログ ファイル
　DSI, 121, 247
　サイズ変更 , 77
　最大サイズの設定 , 28, 41
　走査 , 82
　ディスク スペースの制御 , 41
　データのアーカイブ , 43
　編成 , 252
　ロール バック , 41, 42

ログファイル
　サイズ、制御 , 265

ログ ファイル セット
　sdlutil によるリスト , 293
　定義 , 252
　名前 , 257
　ロール , 265

ログ ファイル データ
　アーカイブ , 130, 143, 145, 225, 226
　アラーム状態の分析 , 154, 226, 227
　エクスポート , 119, 216, 220
　サイズ変更 , 229, 230
　走査 , 228
　抽出 , 121, 214

ログ ファイル データのアーカイブ, 43, 130, 143, 145, 225, 226

ログ ファイル データのエクスポート , 119, 216, 220
　日時に基づいた , 213

ログ ファイル データの抽出 , 121, 214
　日時に基づいた , 213

ログ ファイルにより使用されるディスク スペース、制御 , 41

ログ ファイルにより使用されるディスク スペースの制御 , 41

ログ ファイルのサイズ変更 , 229, 230

ログ ファイルの最大サイズの設定 , 41

ログ ファイルの走査 , 82

ログ ファイルの内容のレポート , 228

ログ ファイルの分析 , 226, 227

ログ ファイルのロール バック , 42

論理ボリューム名レコード , 110