HP Operations Agent

ソフトウェア バージョン: 12.00
Windows®、HP-UX、Linux、Solaris、AIX オペレーティング システム

ユーザー ガイド

ドキュメント リリース日 : 2015 年 9 月 (英語版)
ソフトウェア リリース日 : 2015 年 9 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

© Copyright HP Confidential Hewlett-Packard Development Company, L.P.

商標について

Adobe ® は、Adobe Systems Incorporated (アドビシステムズ社) の商標です。

Microsoft ® および Windows ® は、米国におけるMicrosoft Corporationの登録商標です。

UNIX ® は The Open Group の登録商標です。

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

謝辞

この製品には、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。

この製品には、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/)

この製品には、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。

本製品には 'zlib' 汎用圧縮ライブラリのインタフェースが使用されています。'zlib': Copyright © 1995-2002 Jean-loup Gailly and Mark Adler.

ドキュメントの更新情報

本書のタイトル ページには、以下の識別情報が含まれています。

- ソフトウェア バージョン番号。ソフトウェアのバージョンを示します。
- ドキュメント リリース日付。ドキュメントがアップデートされるたびに変更されます。
- ソフトウェア リリース日付。ソフトウェアのこのバージョンのリリース日付を示します。

最新のアップデートを確認したり、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したりするには、以下の URL にアクセスしてください。**https://softwaresupport.hp.com**

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。**https://hpp12.passport.hp.com/hppcf/createuser.do**

または、HPソフトウェアサポートページ上部の**登録**リンクをクリックしてください。

また、適切な製品サポート サービスをお申し込みいただいた場合は、アップデート版または新版をお送りします。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

次の HP ソフトウェア サポート オンラインの Web サイトにアクセスしてください。
https://softwaresupport.hp.com

この Web サイトには HP ソフトウェアの製品、サービス、サポートに関する詳細情報とコンタクト先が示してあります。

HP ソフトウェア サポート オンラインでは、セルフソルブ技術情報を提供しています。ビジネスの管理に必要なインタラクティブな技術サポート ツールに、すばやく効率的にアクセスすることができます。サポート Web サイトを使用することにより、重要なサポート カスタマとして以下のメリットを受けることができます。

- 興味のある知識に関するドキュメントを検索する
- サポート ケースおよび機能強化要求を提出し追跡する
- ソフトウェアのパッチをダウンロードする
- サポート契約を管理する
- HP サポートの連絡先を調べる
- 利用可能なサービスに関する情報を確認する
- 他のソフトウェア カスタマと議論を始める
- ソフトウェア トレーニングに関して調査し登録する

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 ID を登録するには、以下の Web サイトにアクセスしてください。

https://hpp12.passport.hp.com/hppcf/createuser.do

アクセス レベルの詳細については、次の URL にアクセスしてください。

https://softwaresupport.hp.com/web/softwaresupport/access-levels

HP Software Solutions Now は、HPSWのソリューションと統合に関するポータル Web サイトです。このサイトでは、お客様のビジネスニーズを満たすHP製品ソリューションを検索したり、HP 製品間の統合に関する詳細なリストやITILプロセスのリストを閲覧することができます。このサイトの URL は **http://h20230.www2.hp.com/sc/solutions/index.jsp** です。

目次

第I部: はじめに 15
このドキュメントで使用する名称 16

第II部: 設定 17
第1章: HP Operations Agent の操作 18
監視エージェントの設定 18
MIB オブジェクトを監視するエージェントの設定 18
監視対象オブジェクトの永続化 19
SNMPv3 GET を実行するためのセキュリティ パラメータの機能拡張 20
SNMPv3Get を実行するための opcmona の有効化 21
SourceEX API を使用したセキュリティ パラメータのポリシーへの追加 23
SNMP トラップ インターセプタの設定 25
SNMPv3 トラップ用の SNMP トラップ インターセプタの設定 27
SNMPv3 トラップを傍受するための opctrapi の設定 28
opcpwcrpt ユーティリティによるパスワードの暗号化 30
Varbind のオブジェクト ID に基づきトラップを傍受するための、SNMP トラップ インターセプタの機能拡張 30
HP Operations Agent と HP NNMi の統合 32
メッセージの重要度を関連付けるための NNMi Northbound インターフェイスとの統合 33
メッセージの重要度を高めるための SNMP トラップ インターセプタの設定 33
HPOM コンソール上に CMA を追加するための NNMi Norhbound インターフェイスとの統合 34
HPOM コンソール上での NNMi CIA から CMA を作成するための SNMP トラップ インターセプタの設定 35
SNMP トラップのソースを割り当てるための NNMi トラップ転送インターフェイスとの統合 37
ソース ノード名を引き出すための SNMP トラップ インターセプタの設定 37
メッセージ ストームの抑制の設定 37
メッセージ ストームの検出と抑制 の設定 39
メッセージ レートの確認 42
バックアップ サーバーの設定 42
RTMA コンポーネントの設定 44
Perfd プロセスのためのライセンスの確認 44
設定の変更 44
IPv6 接続上での HP Operations Agent の監視 47
アクセスの制限 47

エージェント ユーザーの設定 48
デフォルトではないユーザーを使用するための要件 49
デフォルトではないユーザーを使用するときの制限事項 49
インストール中のエージェント ユーザーの設定 50
インストール後のエージェント ユーザーの設定 53
Windows でのデフォルト ユーザーの変更 54
もう一方の方法: ovswitchuser コマンドの使用 54
UNIX/Linux でのデフォルト ユーザーの変更 56
プロファイル ファイルの使用 56
もう一方の方法: ovswitchuser コマンドの使用 58
コマンドのデフォルト ユーザーの変更 59
セキュリティ コンポーネント変数の設定 60
対象キーのセキュリティ コンポーネントの設定 62
ハッシュ アルゴリズムを使ったセキュリティ コンポーネントの設定 64
Windows イベント ログの監視 64
HPOM for Windows からのアプリケーション ログおよびサービス イベント ログの監視 66
UNIX/Linux 9.xx 上の HPOM からのアプリケーション ログおよびサービス イベント ログの監視 67
第2章: RTMA コンポーネントのアドバイザ 69
アラームと兆候 69
adviser スクリプトでの作業 69
アドバイザの使用 70
複数のシステム上での adviser スクリプトの実行 71
アドバイザ構文 71
構文表記 71
コメント 72
条件 72
定数 72
式 73
アドバイザ構文内のメトリック名 73
出力リスト 74
変数 75
ALARM 文 75
ALERT 文 76
ALIAS 文 76
ASSIGNMENT 文 77
COMPOUND 文 77
EXEC 文 77
IF 文 77
LOOP 文 78
PRINT 文 78

SYMPTOM 文 79
第3章: パフォーマンス アラーム 80
アラームの処理 80
アラーム ジェネレータ 80
Network Node Manager への SNMP トラップの送信 82
HPOM へのメッセージの送信 83
ローカル アクションの実行 83
アラーム処理のエラー 84
アラームの履歴データの分析 84
履歴データのアラーム情報の例 84
アラーム定義の構成要素 85
アラーム構文のリファレンス 86
アラーム構文 86
構文表記 87
共通要素 87
メトリック名 87
メッセージ 88
ALARM 文 89
構文 89
使い方 91
例 91
ALERT 文 92
構文 93
使い方 93
例 93
EXEC 文 93
構文 93
使い方 94
例 94
PRINT 文 95
IF 文 95
構文 95
使い方 96
例 96
LOOP 文 97
構文 97
使い方 97
例 97
INCLUDE 文 98
構文 98
使い方 98
例 98

USE 文 98
構文 99
使い方 99
VAR 文 100
構文 100
使い方 101
例 101
ALIAS 文 101
構文 101
使い方 101
例 101
SYMPTOM 文 102
アラーム定義の例 102
CPU の問題の例 102
スワップ利用率の例 103
時間に起因するアラームの例 103
ディスク インスタンスのアラームの例 103
アラーム定義のカスタマイズ 104
第4章: 安全な環境における HP Operations Agent 106
ポリシー 106
HTTPS モード通信 107
HTTPS 通信の利点 107
通信ブローカ 109
ファイアウォールのシナリオ 109
HTTPS ベースのセキュリティ コンポーネント 110
証明書 112
HP Operations Agent 証明書サーバー 112
認証局 113
証明書クライアント 113
ルート証明書の更新と配布 114

第III部: HP Operations Agent のパフォーマンス コンポーネントの使用 115
第5章: データ収集の管理 116
メトリック データストアの使用 116
データの収集 117
oacore プロセスのステータスの検証 118
oacore プロセスの起動 119
データ ログ機能の検証 119
データベース ファイルにより使用されるディスク スペースの制御 120
デフォルトのパフォーマンス メトリック クラスを記録するデータベース ファイルにより使用されるディスク スペースの制御 120

カスタム データを記録するデータベース ファイルが使用するディスク スペースの制御 122
データ収集の終了と再開始 122
データ収集の終了 123
データ収集の再開始 123
夏時間 124
システムの時刻の手動変更 124
parm ファイルの使用 124
HP Operations Agent 12.00 のインストール 124
HP Operations Agent 12.00 へのアップグレード 125
parm ファイルの変更 126
parm ファイルのパラメータ 127
パラメータの説明 131
ID 131
Log 132
Thresholds 133
procthreshold 134
appthreshold 136
diskthreshold 136
bynetifthreshold 136
fsthreshold 136
lvthreshold 136
bycputhreshold 137
subprocinterval 137
gapapp 137
fstypes 138
wait 139
size 139
javaarg 139
Flush 140
project_app 140
proclist 140
appproc 140
proccmd 141
ignore_mt 141
cachemem 145
アプリケーション定義パラメータ 146
アプリケーション名 146
File 147
argv1 148
cmd 149
user 149

group 150
or 150
priority 151
アプリケーションの定義の例 151
データ記録間隔の設定 152
フレームに対するデータ収集の設定 152
タスク 1: パスワードなしの SSH アクセスを設定する 153
パスワードなしの SSH アクセスの設定 153
パスワードなしの SSH アクセスの確認 153
タスク 2: HMC システムでフレーム使用率の監視を有効にする 154
タスク 3: HP Operations Agent を設定する 154
Linux 上での GlancePlus に対するグローバルおよびプロセスのシステム コール メトリックの有効化 155
init_ftrace.sh を使用したメトリック コレクションの設定 155
手動でのメトリック コレクションの設定 156
ハイパースレッディングまたは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 158
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 159
第6章: utility プログラムの使い方 161
utility プログラムの実行 161
utility 走査レポート 163
utility のコマンド 163
analyze 164
checkdef 165
detail 165
help 165
filename 166
parmfile 166
scan 167
第7章: Extract プログラムの使い方 168
コマンド ライン インターフェイスを使用した Extract プログラムの実行 168
Export 関数の使用 170
データをエクスポートする方法 170
デフォルトのパフォーマンス メトリック クラス 171
出力ファイル 172
エクスポート テンプレート ファイルの構文 174
パラメータ 176
エクスポート ファイルの出力 177
カスタマイズした出力ファイルの作成 178
ASCII フォーマットに関する注意 178
extract のコマンド 179
第8章: cpsh プログラムの使用 185

インタラクティブ モードの使用 185
リアルタイム メトリックの表示 186
メトリック クラスの変更 186
すべての使用可能メトリックの表示 186
メトリック クラスの整理 187
メトリックのヘルプの表示 187
要約メトリック データの表示 187
しきい値とフィルタ条件の有効化 188
第9章: ベースライン計算の概要 190
HP Operations Agent ノードでのベースラインの設定 190
ベースライン計算が動作していないときのトラブルシューティングの方法 194
第10章: ノード解決の概要 196
第11章: 記録とトレース 201
記録 201
記録ポリシーの設定 202
トレース 202
アプリケーションの特定 203
トレース タイプの設定 205
トレース設定ファイルの概要 206
構文 206
設定ファイルの作成 206
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 206
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 208
トレース機構の有効化 208
トレース メッセージの表示 208
トレース リスト ビューの使用 211
手順のツリー ビューの使用 212
トレースのフィルタリング 212
トレース GUI の使用 214
第IV部: カスタム データの記録 220
第12章: Perl アプリケーション プログラミング インターフェイスによるカスタム データの送信 221
カスタム データの送信 222
カスタム データをメトリック データストアに送信する方法 223
API を使用してデータをデータストアに送信するための使用例 225
第13章: データ ソース統合の概要 228
DSI の動作 229
第14章: DSI メトリックをメトリック データストアに記録するためのモデルの作成 231
クラス仕様のファイルの作成 231
クラスの仕様構文 232
クラスの記述 232

構文 233
CLASS 233
LABEL 233
Records per Hour 234
デフォルトの設定 235
クラス仕様のサンプル 235
メトリックの記述 237
LABEL 237
要約方法 238
DSI コンパイラによるクラスの仕様ファイルのコンパイル 239
sdlcomp コンパイラ 239
クラス仕様の変更 239
第15章: DSI メトリックのメトリック データストアへの記録 241
構文 241
dsilog によるログ処理 242
dsilog によるデータ処理方法 242
sdlutil によるデータの管理 242
構文 243
第16章: メトリック データストアに記録される DSI データの使用 244
DSI メトリックのアラーム定義 244
アラーム処理 245
DSI データのエクスポート 245
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 245
Performance Manager でのデータの表示 245
第17章: データ ソースの統合例 246
dsilog スクリプトの記述 246
例 1 - 問題となる dsilog スクリプト 246
例 2 - 推奨する dsilog スクリプト 246
vmstat データの記録 247
クラス仕様のファイルの作成 247
クラス仕様ファイルのコンパイル 248
dsilog によるログ処理の開始 248
データへのアクセス 249
システム ユーザー数の記録 249

第V部: トランザクション追跡 251
第18章: トランザクション追跡の紹介 252
パフォーマンス管理の改善 252
トランザクション追跡の利点 252
ユーザーの視点から見たトランザクション時間 253
トランザクション データ 253
サービス レベル目標 253

リアルタイムでの注文処理のシナリオ 254
リアルタイム注文処理の必要事項 254
注文処理アプリケーションの準備 254
トランザクション データの監視 255
ARM の使用のガイドライン 255
第19章: トランザクション追跡の動作 257
ARM 2.0 のサポート 257
ARM API コールのサポート 257
arm_complete_transaction コール 258
ARM 組み込みアプリケーションの例 258
アプリケーション名とトランザクション名の指定 259
トランザクション追跡デーモン (ttd) 260
ARM API コール ステータス リターン 260
測定インターフェイス デーモン (midaemon) 261
トランザクション構成ファイル (ttd.conf) 262
新しいアプリケーションの追加 262
新しいトランザクションの追加 262
range 値または slo 値の変更 263
構成ファイルのキーワード 263
tran 263
range 264
slo 264
構成ファイルのフォーマット 264
構成ファイル例 266
ARM を使用する際のオーバーヘッドの考慮事項 267
ガイドライン 267
ディスク I/O オーバーヘッド 268
CPU オーバーヘッド 268
メモリ オーバーヘッド 268
第20章: トランザクションについて 270
開始する前に 270
トランザクション追跡のセットアップ 270
サービス レベル目標 (SLO) の定義 271
parm ファイルの変更 271
トランザクション データの収集 272
エラーの対応 272
固有のトランザクションの制限 273
構成ファイルのカスタマイズ (オプション) 273
パフォーマンス データの監視 274
アラーム 275
第21章: トランザクション追跡のメッセージ 276
第22章: トランザクション メトリック 277

第23章: トランザクション追跡の例 278
リアルタイム注文処理の擬似コード 278
構成ファイル例 281
例 1 (注文処理の擬似コード例) 281
例 2 281
例 3 281
例 4 282
第24章: 高度な機能 283
データ型の使用方法 283
ユーザー定義メトリック 284
第25章: トランザクション ライブラリ 286
ARM ライブラリ (libarm) 286
プラットフォームごとの C コンパイラ オプションの例 290
ARM NOP ライブラリ 292
Java ラッパーの使用 292
例 292
アプリケーションの設定 (arm_init) 292
構文: 292
トランザクションの設定 (arm_getid) 293
UDM を使用したトランザクションの設定 293
メトリックの追加 293
メトリック データの設定 294
UDM を使用しないトランザクションの設定 294
トランザクション インスタンスの設定 294
トランザクション インスタンスの開始 (arm_start) 295
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 295
相関係数の要求 295
親相関係数の送信 295
親相関係数の要求と送信 295
相関係数情報の取得 296
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 296
トランザクション インスタンス データの更新 296
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 296
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 297
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 297
トランザクション インスタンスの停止 (arm_stop) 297
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 297
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 298
完全なトランザクションの使用 298
UDM を使用した完全なトランザクションの使用 298
UDM を使用しない完全なトランザクションの使用 299
その他のドキュメント 299

第VI部: トラブルシューティング 300
運用監視コンポーネント 300
Performance Collection Component 304
RTMA 305
HP GlancePlus 306
hpsensor 306
その他 306

第VII部: HP Operations Agent の以前のバージョンから 12.00 へのアップグレード 309
HP Operations Agent 12.00 と以前のバージョンの比較 310
Performance Collection Component 313
parm ファイル 314
Utility プログラム 318
utility 走査レポート 319
初期 parm ファイルのグローバル情報 320
初期 parm ファイルのアプリケーション定義 321
parm ファイルのグローバル変更通知 322
parm ファイルのアプリケーション変更内容 322
scope のオフタイム通知 323
アプリケーション固有の要約レポート 323
プロセス要約レポート 324
走査の開始および終了レポート 325
アプリケーションの全体的な要約 325
コレクタの適用範囲の要約 326
クラス要約およびログ ファイルの内容の要約 326
ログ ファイルの空きスペースの要約 327
Extract 328
メトリック 330
SNMP トラップ インターセプタ 333
データ ソース統合 334
よくある質問 (FAQ) 337

ドキュメントのフィードバックを送信 340

第I部: はじめに

HP Operations Agentはシステムの重要な要素のヘルス、パフォーマンス、可用性、リソース使用率を示すメトリックを収集することにより、システムの監視を可能にします。

HP Operations Agent は組み込まれたデータ コレクタの **oacore** を使用し、システム全体のパフォーマンスおよび状況データを継続的に収集し、収集したデータをメトリック データストアに記録します。

HP Operations Agent を HPOM、Smart Plug-in (SPI) および Operations Manager i (OMi) と共に使用すると、監視システムで稼働するビジネス アプリケーション、インフラストラクチャ (システム リソース)、およびアプリケーション負荷を監視できます。

次の機能は、HP Operations Agent のデータ収集機能と監視機能を強化します。

機能	説明
parmfile	**oacore** データ コレクタのデータ収集機構は、**parm** ファイルの設定により制御します。**parm** ファイルに定義されているクラスに基づいて、**oacore** データ コレクタはさまざまな一連のデータを収集し、システムのヘルスとパフォーマンスに関して幅広い見解を表します。
Utility および Extract	Utility や Extract などのツールを使用すると、メトリック データストアに保存されている特定の情報を表示できます。
カスタム データ ログ	カスタム データ ログ記録用の API または DSI のいずれかを使用すると、カスタム データをメトリック データストアに記録できます。
ベースライン計算	ベースライン計算のプロセスを使用すると、基準値を計算して返し、パフォーマンスの傾向の分析、および最適なしきい値の動的な設定を行うことができます。
リアルタイム メトリック アクセス	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、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。
alarmdef ファイル	**alarmdef** ファイルにアラームを定義できます。データが **oacore** または他のコレクタにより記録されると、そのデータは `alarmdef` ファイル内のアラーム定義と比較されす。監視対象メトリックが定義された条件を満たす場合や超える場合に、アラートまたはアクションがトリガされます。

トランザクション追跡	トランザクションのアプリケーションにおける進行状況は、追跡できます。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、サービス レベル アグリーメント (SLA: Service Level Agreement) の管理を支援します。さらに、サービス レベル目標 (SLO: Service Level Objective) が **alarmdef** ファイルに定義されている条件を超えると、アラームを発生させます。

このドキュメントで使用する名称

このドキュメントでは、以下の名称を使用します。

名称	説明
`<OvInstallDir>` HP Operations Agent のインストール ディレクトリ。	**<OvInstallDir>** は、このドキュメントで以下の場所を示すのに使用されています。 • Windows の場合: `%ovinstalldir%` • HP-UX/Linux/Solaris の場合: `/opt/OV/` • AIX の場合: `/usr/lpp/OV/`
`<OvDataDir>` HP Operations Agent の設定データ ファイルと実行時データ ファイルのディレクトリ。	**<OvDataDir>** は、このドキュメントで以下の場所を示すのに使用されています。 • Windows の場合: `%ovdatadir%` • HP-UX/Linux/Solaris の場合: `/var/opt/OV/` • AIX の場合: `/var/opt/OV/`
<OvInstallBinDir> bin ディレクトリには、HP Operations Agent のすべてのバイナリ (実行可能ファイル) があります。	**<OvInstallBinDir>** は、このドキュメントで以下の場所を示すのに使用されています。 • Windows x64 の場合: `%ovinstalldir%\bin\win64\` • Windows x32 の場合: `%ovinstalldir%\bin\win32\` • HP-UX/Linux/Solaris の場合: `/opt/OV/bin` • AIX の場合: `/usr/lpp/OV/bin`

第II部: 設定

HP Operations Agent が HPOM ベースの管理環境にインストールされている場合、中央コンソールからネットワーク環境に配布されているシステムとアプリケーションを監視できます。HPOM のポリシーをノードに配布した後に、運用監視コンポーネントの各種コンポーネントを使用できます。設定変数を使用すると、HP Operations Agent のデフォルトの動作を設定することができます。これらの変数には、ovconfchg コマンドを使って目的の値を設定できます。設定できるのは、監視エージェント、SNMP トラップ インターセプタ、メッセージ ストームの抑制、バックアップ サーバー、RTMA コンポーネント、およびセキュリティ コンポーネントです。

第1章: HP Operations Agent 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 運用監視コンポーネント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 コンポーネントは監視を開始します。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。

- Windows の HPOM の場合: HPOM for Windows のオンライン ヘルプの「Event Policy Editors」を参照してください。
- UNIX/Linux の HPOM の場合:『HPOM for UNIX 9.10 Concepts Guide』の「Implementing Message Policies」を参照してください。

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構成する手順は以下のとおりです。

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 の場合:

%ovinstalldir%bin

HP-UX/Linux/Solaris の場合:

/opt/OV/bin

AIX の場合:

/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 <community_string>

  この <community_string> 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 <list_of_community_strings>

  この <list_of_community_strings> 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 Agent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。

  たとえば、次のようになります。

  ovconfchg -ns eaagt SNMP_COMMUNITY_LIST “C1,C2,C3”

  HP Operations Agent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 Agent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。

> **注:** 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 Agent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。

監視対象オブジェクトの永続化

監視対象オブジェクトとセッション変数の値を定期的に保存するように HP Operations Agent を設定することができます。 監視対象オブジェクトとセッション変数の値を保存すると、中断や障害の発生時に保存した値を利用できます。

OPC_MON_SAVE_STATE 変数を使うと、監視対象オブジェクトとセッション変数の値を保存するようにエージェントを設定できます。

監視対象オブジェクトとセッション変数の値を定期的に保存するようにエージェントを設定するには、以下の手順に従います。

1. root または管理者としてノードにログオンします。
2. 次のコマンドを実行します。

   ovconfchg -ns eaagt -set OPC_MON_SAVE_STATE TRUE

   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開始します。

HP Operations Agent 12.00 をインストールすると、OPC_MON_SAVE_STATE 変数に次のような影響を与えます。

- HP Operations Agent 12.00 をインストールする前に変数を任意の値に設定しなかった場合、OPC_MON_SAVE_STATE 変数は値を FALSE と見なします。
- HP Operations Agent 12.00 をインストールする前に ovconfchg コマンドを使用して値 (TRUE または FALSE) を設定した場合、インストール プロセスの終了後も設定した値は有効のままになります。

たとえば、バージョン 12.00 をインストールする前に `ovconfchg -ns eaagt -set OPC_MON_SAVE_STATE TRUE` コマンドを使用した場合、HP Operations Agent 12.00 のインストール後にも同じ値 (TRUE) が維持されます。

SNMPv3 GET を実行するためのセキュリティ パラメータの機能拡張

HP Operations Agent ノード上では、監視エージェント コンポーネント (**opcmona**) が、HPOM 管理サーバーからの測定しきい値ポリシー配布後に、MIB オブジェクトを監視します。

MIB データを照会するために、**opcmona** は、指定したオブジェクト ID (OID) 上で **SNMPv3 GET** を実行します。**opcmona** が **SNMPv3Get** を実行できるようにするには、追加のセキュリティ パラメータが必要になります。これらのパラメータを、次の表に示します。

パラメータ	説明
SNMPV3_ENGINEID	SNMP エンティティを一意に識別するために使用される SNMP エンジン ID を指定します。
SNMPV3_USER	管理者が作成した SNMPv3 ユーザー名を指定します。
SNMPV3_AUTHTYPE	パスワードの暗号化に使用するプロトコルを指定します。 パスワードの暗号化には、Message Digest Algorithm 5 (MD5) または Secure Hash Algorithm (SHA) のいずれかのプロトコルを使用できます。
SNMPV3_AUTHPASSPHRASE	**opcpwcrpt** ユーティリティを使用して暗号化したパスワードを指定します。 **注:** 詳細については、「SNMPv3 GET を実行するためのセキュリティ パラメータの機能拡張」(20ページ)を参照してください。

SNMPV3_ENCRYPTTYPE	プロトコル データ ユニット (PDU) を暗号化するためのプロトコルを指定します。 PDU の暗号化には、Data Encryption Standard (DES) または Advanced Encryption Standard (AES) のいずれかのプロトコルを使用できます。
SNMPV3_ENCRYPTPASSPHRASE	PDU を暗号化するために、DES が使用するキーを指定します。 **注:** 管理者によって作成された暗号化キーは、**opcpwcrpt** ユーティリティで暗号化されます。 詳細については、「SNMPv3 GET を実行するためのセキュリティ パラメータの機能拡張」(20ページ)を参照してください。

SNMPv3Get を実行するための opcmona の有効化

次のいずれかのシナリオで、**opcmona** で **SNMPv3Get** を実行するようにできます。

- opctrapi がノード情報ポリシーを使用するように設定されている場合
- opctrapi がノード情報ポリシーを使用するように設定されていない場合

opctrapi がノード情報ポリシーを使用するように設定されている場合

opctrapi がノード情報ポリシーを使用するように設定されている場合、SNMP エンティティのエンジン ID をノード上で使用可能です。詳細については、「ノード情報ポリシーを使用するための opctrapi の設定」を参照してください。測定しきい値ポリシーにエンジン ID を追加して、**opcmona** が **SNMPv3Get** を実行できるようにします。

注: `ovconfget eaagt` コマンドを実行すると、エンジン ID を取得できます。次の出力が表示されます。

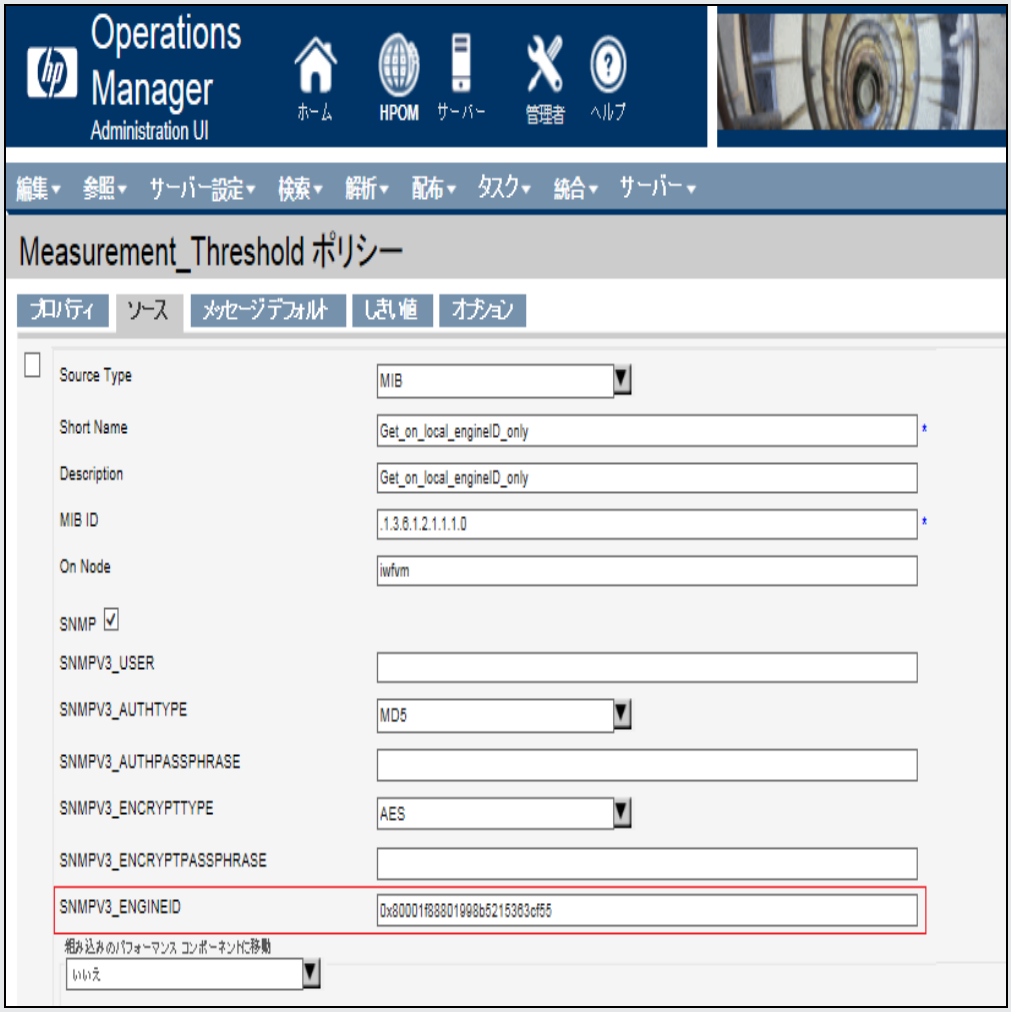
```
OPC_INSTALLATION_TIME=Wed Jan 7 22:42:20 IST 2015
OPC_NODENAME=iwfvm05835.hpswlabs.hp.com
SNMP_V3_
USERSDATA=demosnmpv3user:MD5:7*=*C61Ntd=@*Eb#@*E0##D:DES:7*=*C61Ntd=@*Eb#@*E0
##D:0x80001f88801998b5215363cf55
```

この場合、エンジン ID は 0x80001f88801998b5215363cf55 です。

例

opctrapi を設定するために使用したエンジン ID は、0x80001f88801998b5215363cf55 であるとします。

次に示すように、測定しきい値ポリシーにエンジン ID を追加します。



SNMP エンティティのエンジン ID を測定しきい値ポリシーに追加すると、**opcmona** は、その SNMP エンティティ上に存在する OID 上で **SNMPv3Get** を実行できます。

opctrapi がノード情報ポリシーを使用するように設定されていない場合

opctrapi がノード情報ポリシーを使用するように設定されていない場合、次の行を測定しきい値ポリシーに追加する必要があります。

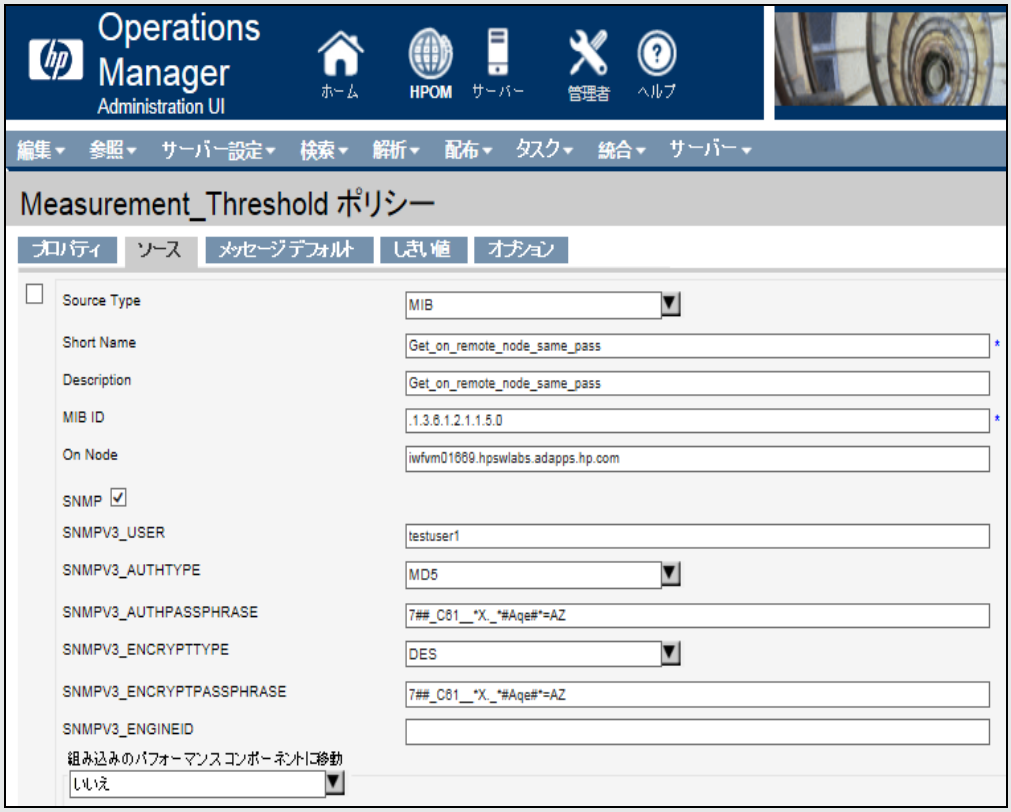
SNMPV3_USER `<user name>`
SNMPV3_AUTHTYPE `<authentication method>`
SNMPV3_AUTHPASSPHRASE `<authentication password>`
SNMPV3_ENCRYPTTYPE `<encryption method>`
SNMPV3_ENCRYPTPASSPHRASE `<encryption key>`

注:

SNMPV3_AUTHPASSPHRASE および SNMPV3_ENCRYPTPASSPHRASE は、自動的に暗号化されます。

暗号化と認証のパスワードが同じ場合、SNMPV3_ENCRYPTPASSPHRASE 変数の値を空白のままにしておくことができます。

例:



SNMP エンティティのパラメータを測定しきい値ポリシーに追加すると、**opcmona** は、その SNMP エンティティ上に存在する OID 上で SNMPv3Get を実行できます。

SourceEX API を使用したセキュリティ パラメータのポリシーへの追加

SNMPv3 MIB の監視に使用される複数のセキュリティ パラメータをサポートするために、新しいインターフェイスの **SourceEx_SNMPV3** が導入されました。SourceEx_SNMPV3 API では、Perl スクリプトを使用して、セキュリティ パラメータを測定しきい値ポリシーに追加できます。

注: 測定しきい値ポリシーの VBScript で、SourceEx_SNMPV3 API を使用することはできません。

次のシナリオでセキュリティ パラメータを測定しきい値ポリシーに追加するには、次の構文を使用します。

1. **opctrapi** がノード情報ポリシーを使用するように設定されている場合:

```
Policy-> SourceEx_SNMPV3 ("SNMP\\<object id>[\\<hostname>]","<Engine ID>");
```

注: エンジン ID のみが指定されている場合、**opcmona** は、**opctrapi** 用に配布されているノード情報ポリシーからその他のセキュリティ ポリシーを取得し、SNMPv3 トラップを受信します。詳細については、「ノード情報ポリシーを使用するための opctrapi の設定」を参照してください。

例:

```
ADVMONITOR "TestMIBviaPerl_V3"
DESCRIPTION "<initial policy version>"
SCRIPTTYPE "Perl"
INSTANCEMODE  SAME
MAXTHRESHOLD
SEPARATORS "                "
SEVERITY Unknown
EXTERNAL "SNMPPolicy"
DESCRIPTION ""
MSGCONDITIONS
DESCRIPTION "New threshold level"
CONDITION_ID "c5e0a2e4-a401-4186-876f-8b20b1ffcd8b"
CONDITION
THRESHOLD
SCRIPT "
use warnings;
my $MetricEx = $Policy->SourceEx_SNMPV3
(\"SNMP\\\\\\\\.1.3.6.1.2.1.1.1.0\\\\\\\\1x.18x.28.7x
\",'0x8000000001020304');
```

ここで、

パラメータ	値
<object id>	\\.1.3.6.1.2.1.1.1.0
<hostname>	\\1x.18x.28.7x
<Engine ID>	0x8000000001020304

2. **opctrapi** がノード情報ポリシーを使用するように設定されていない場合:

```
$Policy-> SourceEx_SNMPV3 ("SNMP\\<object id>[\\<hostname>]", "<user
name>","<authentication method>","<authentication password>","<encryption
method>","<encryption key>");
```

例:

```
ADVMONITOR "TestMIBviaPerl_V3"
DESCRIPTION "<initial policy version>"
SCRIPTTYPE "Perl"
INSTANCEMODE  SAME
MAXTHRESHOLD
```

```
SEPARATORS "                "
SEVERITY Unknown
EXTERNAL "SNMPPolicy"
DESCRIPTION ""
MSGCONDITIONS
DESCRIPTION "New threshold level"
CONDITION_ID "c5e0a2e4-a401-4186-876f-8b20b1ffcd8b"
CONDITION
THRESHOLD
SCRIPT "
use warnings;
my $MetricEx = $Policy->SourceEx_SNMPV3
(\"SNMP\\\\\\\\.1.3.6.1.2.1.1.1.0\\\\\\\\1x.18x.28.7x\",
\"TestUser\",\"MD5\",'7*=*C61Ntd=@*Eb#@*E0##D',\"DES\",'7*=*C61Ntd=@*Eb#@*E0
##D');
```

ここで、

パラメータ	値
<object id>	\\.1.3.6.1.2.1.1.1.0
<hostname>	\\1x.18x.28.7x
<user name>	TestUser
<authentication method>	MD5
<authentication password>	'7*=*C61Ntd=@*Eb#@*E0##D'
<encryption method>	DES
<encryption key>	'7*=*C61Ntd=@*Eb#@*E0##D'

メモ: パスワードが単一引用符で囲まれていることを確認します。

SNMP トラップ インターセプタの設定

デフォルトでは、SNMP トラップ インターセプタは、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、構成に基づいて適切なイベントを生成できます。

注: SNMP トラップ インターセプタ (`opctrapi`) は、MIB のテキスト部分をフォーマットしません。たとえば、メッセージ テキストでは、MIB が `.iso.identified-organization.dod.internet.private` ではなく、`.1.3.6.1.4` として表示されます。

SNMP トラップ 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構成によって変更できます。

- SNMP_TRAP_PORT: デフォルトのポートは **162** です。この値は、HP Operations Agentノードの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と、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、SNMP トラップ インターセプタは、HP Operations Agent 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と、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と、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、SNMP トラップ 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。SNMP トラップ 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。

  注: 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。

SNMP トラップ インターセプタのデフォルト動作を変更する手順は以下のとおりです。

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 <port_number>
     ```

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

   - SNMP トラップ 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 setSNMP_TRAP_FORWARD_ENABLE TRUE
     ```

   - SNMP トラップ 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 ”<machinename>:<port>”
     ```

     <machinename> 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、<port> 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 Filter>”

  <OID Filter> には、OID とワイルドカード文字を指定します。SNMP トラップ 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。

SNMPv3 トラップ用の SNMP トラップ インターセプタの設定

ネットワーク環境では、トラップ送信デバイスと HP Operations Agent との間の安全な通信を確保することが重要になります。Simple Network Management Protocol バージョン 3 (SNMPv3) では、ユーザーを認証し、ネットワークを介して送信されるデータ パケットを暗号化することにより、トラップ送信デバイスに安全にアクセスできます。

以前のバージョンの HP Operations Agent では、**opctrapi** プロセスは、SNMPv1 および SNMPv2 トラップを傍受するように設定されていました。現在のバージョンの HP Operations Agent では、**opctrapi** は SNMPv3 トラップと通知メッセージのどちらも傍受できます。

opctrapi で SNMPv3 トラップを傍受できるようにするには、次の変数を設定します。

注: opctrapi プロセスは、NETSNMP モードでのみ SNMPv3 トラップを傍受します。

- SNMP_V3_ENABLE - この変数は必須で、eaagt 名前空間で設定する必要があります。デフォルトでは、この変数は TRUE に設定されます。

  opctrapi が SNMPV3 トラップを傍受しないようにするには、SNMP_V3_ENABLE を FALSE に設定します。この変数を FALSE に設定した場合、**opctrapi** は SNMPv1 と SNMPv2 トラップのみを傍受します。

- SNMP_V3_USERSDATA - この変数は必須で、eaagt 名前空間で設定する必要があります。この変数を使用して、ユーザーを設定します。

  SNMP_V3_USERSDATA = <ユーザー 1 のパラメータ>;<ユーザー 2 のパラメータ>;<ユーザー n のパラメータ>

  <ユーザーのパラメータ> には以下のものが含まれています。

  <ユーザー名 >:<認証方法 >:<認証パスワード >:<暗号化方式 >:<暗号化キー>:<エンジン ID>

パラメータ	説明
<ユーザー名 >	管理者が作成した SNMPv3 ユーザー名を指定します。
<認証方法 >	パスワードの暗号化に使用するプロトコルを指定します。 パスワードの暗号化には、Message Digest Algorithm 5 (MD5) または Secure Hash Algorithm (SHA) のいずれかのプロトコルを使用できます。

<認証パスワード>	**opcpwcrpt** ユーティリティを使用して暗号化したパスワードを指定します。 **注:** 詳細については、「opcpwcrpt ユーティリティによるパスワードの暗号化」を参照してください。
<暗号化方式>	プロトコル データ ユニット (PDU) を暗号化するためのプロトコルを指定します。 PDU の暗号化には、Data Encryption Standard (DES) または Advanced Encryption Standard (AES) のいずれかのプロトコルを使用できます。
<暗号化キー>	PDU を暗号化するために、DES が使用するキーを指定します。 **注:** 管理者によって作成された暗号化キーは、**opcpwcrpt** ユーティリティで暗号化されます。 詳細については、「opcpwcrpt ユーティリティによるパスワードの暗号化」を参照してください。
<エンジン ID>	SNMP エンティティを一意に識別するために使用される SNMP エンジン ID を指定します。

SNMPv3 トラップを傍受するための opctrapi の設定

SNMPv3 トラップを傍受するように opctrapi を設定するには、次のいずれかの方法を使用できます。

- ノード情報ポリシーの使用
- XPL 変数の使用

ノード情報ポリシーの使用

以下の手順に従ってください。

1. HPOM サーバーにログオンします。
2. 次の変数をノード情報ポリシーに追加します。
   a. SNMPv3 を有効にするために、SNMP_V3_ENABLE 変数を True に設定します。

      `SNMP_V3_ENABLE = TRUE`

   b. ユーザーを設定するために、次のコマンドを実行します。

      `SNMP_V3_USERSDATA` = <Parameters for User 1>;<Parameters for User 2>;<Parameters for User n>
3. ノード情報ポリシーをノード上に配布します。`opctrapi` が自動的に再起動されます。

以下に、SNMPv3 トラップを傍受するために、opctrapi を設定するノード情報ポリシーのスニペットの例を示します。

```
;XPL config
[eaagt]
SNMP_V3_ENABLE=TRUE
SNMP_V3_USERSDATA=<Parameters for User 1>;<Parameters for User 2>;<Parameters for User n>
```

注: ノード情報ポリシー タイプを使用すると、HP Operations Agent が実行中の管理ノード上で設定情報を変更することができます。

XPL 変数の使用

以下の手順に従ってください。

1. HP Operations Agent ノードにログオンします。
2. SNMPv3 トラップの傍受を有効にするには、以下のコマンドを実行します。

   `ovconfchg -ns eaagt -set SNMP_V3_ENABLE TRUE`
3. ユーザーを設定するために、次のコマンドを実行します。

   `ovconfchg -ns eaagt -set SNMP_V3_USERSDATA` <Parameters for User 1>; <Parameters for User 2>;<Parameters for User n>

   `opctrapi` が自動的に再起動されます。

例:

```
ovconfchg -ns eaagt -set SNMP_V3_ENABLE TRUE
ovconfchg -ns eaagt -set SNMP_V3_USERSDATA
"snmpv3User1:SHA:7*=*C61Ntd=@*Eb#@*E0##D:DES:7*=*C61Ntd=@*Eb#@*E0##D:0x8000000001020304;snmpv3User2:SHA:8*=*D61Ntd=@*Eb#@*E0##D:DES:8*=*D61Ntd=@*Eb#@*E0##D:0x8000000001030404;"
```

ここで、

パラメータ	値	値
<user name>	snmpv3User1	snmpv3User2
<authentication method>	SHA	SHA
<authentication password>	7*=*C61Ntd=@*Eb#@*E0##D	8*=*D61Ntd=@*Eb#@*E0##D
<encryption method>	DES	DES
<encryption key>	7*=*C61Ntd=@*Eb#@*E0##D	8*=*D61Ntd=@*Eb#@*E0##D
<Engine ID>	0x8000000001020304	0x8000000001030404

メモ: 各パラメータは、コロン (:) で区切り、各ユーザーは、セミコロン (;) で区切ります。

opcpwcrpt ユーティリティによるパスワードの暗号化

1. 管理者権限で HPOM サーバーにログオンします。
2. コマンド プロンプトを開きます。
3. 以下の場所に移動します。

   Windows の場合:

   "%OvBinDir%\OpC\install"

   Linux の場合:

   /opt/OV/bin/OpC/install
4. 以下のコマンドを実行して、パスワードを暗号化します。

   # opcpwcrpt <your password>

出力文字列が暗号化後のパスワードです。<authentication password> または <encryption key> の代わりに、暗号化されたパスワードを適切に使用します。

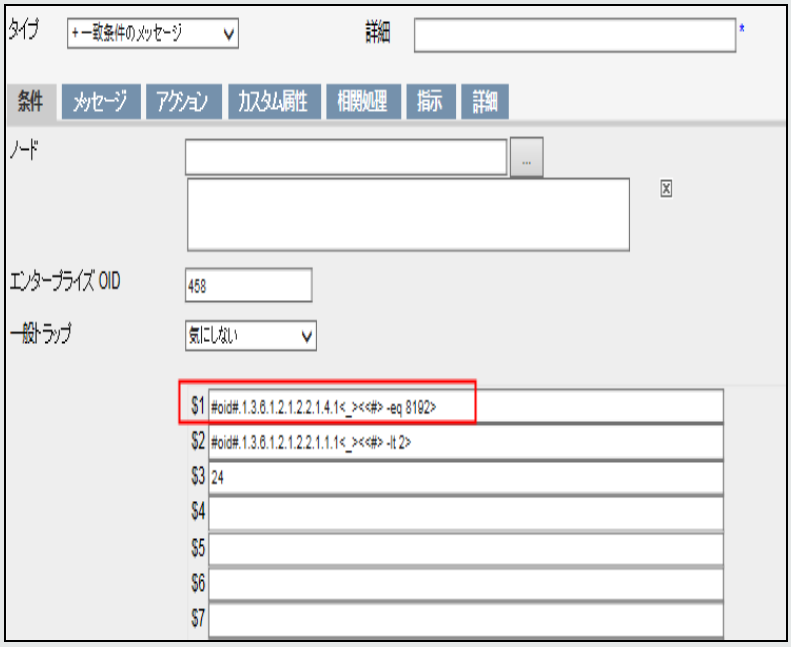
Varbind のオブジェクト ID に基づきトラップを傍受するための、SNMP トラップ インターセプタの機能拡張

SNMP トラップ インターセプタ **opctrapi** は、varbind のオブジェクト ID (OID) と位置に基づいて、SNMP トラップを傍受するように機能拡張されています。

opctrapi が varbind のオブジェクト ID に基づいて SNMP トラップを傍受できるようにするには、SNMP トラップ インターセプタ ポリシーを作成または変更し、以下の例に示すコードを追加 (または変更) できます。

<keyword><object id><separator><pattern string>

例:



スクリーンショットでマークしたインスタンスの場合:

<keyword> は常に “#oid#”

<object id> は .1.3.6.1.2.1.2.2.1.4.1

<separator> は <_>

<pattern string> は <<#> -eq 8192>

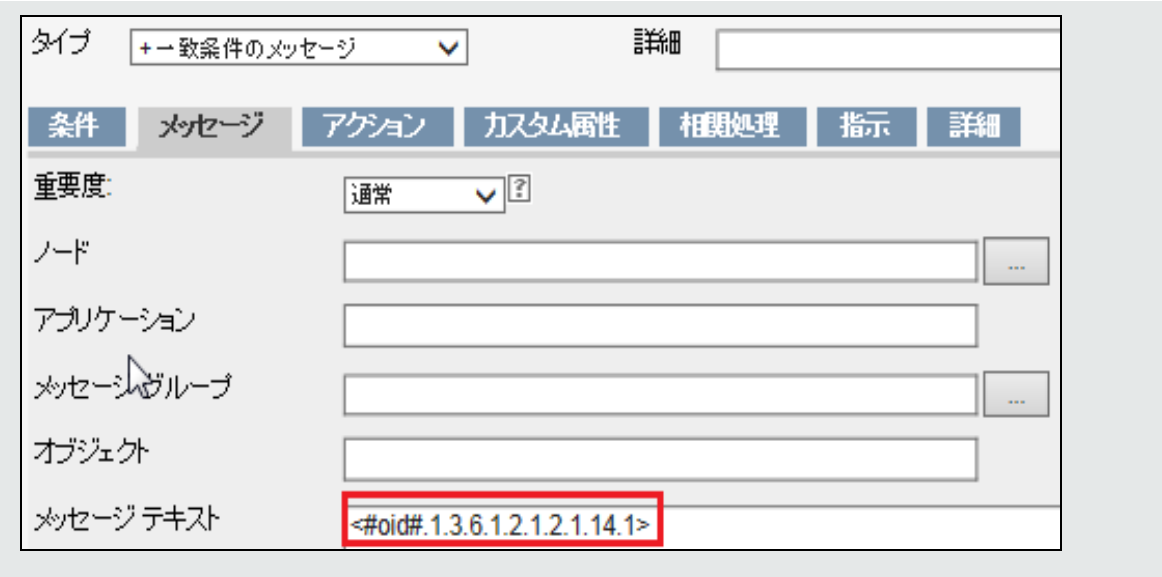
値は 8192

varbind とのマッチング時に、**opctrapi** はフィールドがキーワード **#oid#** で始まっているかどうかを確認します。フィールドがキーワード **#oid#** で始まっている場合、トラップ内の varbind OID が、フィールドに記載されているパターン文字列と比較され、条件が検証されます。

注: SNMP トラップ インターセプタ ポリシーでは、以下の項目で OID を使用することもできます。

- メッセージ属性 (メッセージ テキスト、メッセージ グループ、サービス名、メッセージ オブジェクト、メッセージ キー、オブジェクト、アプリケーション)
- カスタム メッセージ属性 (CMA)

次の例では、OID がメッセージ テキストで使用されています。



HP Operations Agent と HP NNMi の統合

HP Network Node Manager i ソフトウェア (HP NNMi) は、ネットワーク要素の検出とステータスに対して、既存の SNMP エージェントを使用するネットワーク管理ソフトウェアです。HP Operations Agent と統合すると、NNMi によって転送されたトラップの監視、および HPOM コンソール上への豊富な SNMP トラップの表示を行うのに役立ちます。

HP Operations Agent と HP NNMi の統合は、以下により実現されます。

- Northbound インターフェイス - NNMi Northbound インターフェイスは、SNMPv2c トラップを受信可能なすべてのアプリケーションに、NNMi インシデントを転送するのに使用されます。SNMP イベントは、Northbound インターフェイスを使用する NNMi から SNMP トラップ インターセプタ (opctrapi) に送信されます。トラップ送信側デバイスと受信側アプリケーションでのイベントとの間に正しい関連付けを行うには、これらのトラップに対するルールを varbind によってカスタマイズする必要があります。統合は、opctrapi で NNMi Northbound インターフェイスによって転送されるトラップを傍受するためのルールが含まれているポリシーを使用することで、実行されます。また、opctrapi は、ポリシーに定義された重要度やノード名などのさまざまな設定に基づいて、発信メッセージの各種パラメータを設定します。Northbound インターフェイスの使用の詳細については、『HP Network Node Manager i Software デプロイメント リファレンス、バージョン 9.22』を参照してください。
- イベント転送 - NNMi SNMP トラップ転送メカニズムは各 SNMP トラップを強化してから、トラップ転送先に転送します。イベント転送メカニズムには以下の 2 つがあります。
  - 元のトラップ転送 - varbind はこれ以上追加されません。
  - デフォルトのトラップ転送 - NNMi は元のソース オブジェクトを識別するための varbind を追加します。opctrapi はこのメカニズムを使用して SNMP トラップを強化します。

デフォルトのトラップ転送の詳細については、『NNMi オンライン ヘルプ』のトラップ転送の設定に関するトピックを参照してください。

NNMi は NNMi イベント転送または Northbound インターフェイスのいずれかを使用してインシデントを転送し、元のイベントに varbind を追加します。これらの varbind には追加のトラップ情報が含まれています。HP Operations Agent でこの情報を処理して使用するには、以下を行うために SNMP インターセプタ (opctrapi) を設定する必要があります。

- HPOM メッセージに対する SNMP トラップで使用可能な重要度の関連付け。この強化機能は、Northbound インターフェイスでのみ動作します。詳細については、「メッセージの重要度を関連付けるための NNMi Norhbound インターフェイスとの統合」を参照してください。
- NNMi インシデントからの HPOM カスタム メッセージ属性 (CMA) の作成。この強化機能は、Northbound インターフェイスでのみ動作します。詳細については、「HPOM コンソール上に CMA を追加するための NNMi Northbound インターフェイスとの統合」を参照してください。
- SNMP トラップの転送元のソース ノード名の派生。この機能強化は、デフォルトのトラップ転送のみで動作します。詳細については、「SNMP トラップのソースを割り当てるための NNMi トラップ転送インターフェイスとの統合」を参照してください。

メッセージの重要度を関連付けるための NNMi Northbound インターフェイスとの統合

HP Operations Agent では NNMi ツール (nnmopcexport.ovpl) で生成されたポリシーが使用されます。このツール (nnmopcexport.ovpl) は SNMP トラップの重要度を設定します。NNMi ツールによって生成される重要度が Normal (正常域)、Critical (危険域)、Warning (注意域)、Major (メジャー)、Minor (マイナー) である場合、メッセージの重要度は、それぞれ Normal (正常域)、Critical (危険域)、Warning (注意域)、Major (メジャー)、Minor (マイナー) として表示されます。

NNMi ツールが HPOM ポリシーによって生成されるメッセージの重要度を**不明**として示す場合にのみ、トラップ内の使用可能な varbind を使用して重要度の値を引き出すように、opctrapi を設定します。この設定を使用すると、SNMP トラップで使用可能な重要度を、HPOM メッセージの重要度に関連付けることができます。

メッセージの重要度を高めるための SNMP トラップ インターセプタの設定

SNMP トラップで使用可能な重要度レベルに基づいてメッセージの重要度を設定するために、SNMP トラップ インターセプタ (opctrapi) を設定できます。SNMP トラップの特定の OID から重要度を読み込み、同じ重要度レベルを HPOM メッセージに関連付けます。この設定手順は、opctrapi で SNMP トラップで使用可能な重要度レベルを使用するのに役立ちます。

> **注:** この設定は、HPOM で設定したメッセージの重要度が「不明」であるルールに適用されます。

opctrapi のデフォルト動作は、以下のプロパティを設定することで変更できます。

- OPC_SNMP_SET_SEVERITY: 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** に設定することで、SNMP トラップ インターセプタが、varbind OID (**.1.3.6.1.4.1.11.2.17.19.2.2.12**) を持つ SNMP トラップを読み込み、メッセージの重要度を設定するようにすることができます。このデフォルトの OID 値が SNMP トラップで使用できない場合は、メッセージの重要度は「不明」のままになります。
- OPC_SNMP_OVERRIDE_SEVERITY_OID: 新しい OID の値を設定できます。新しい値がメッセージの重要度を指定します。

SNMP インターセプタを設定するには、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。

- SNMP インターセプタが、SNMP トラップに定義した重要度レベルを読み込めるようにするには、以下のコマンドを実行します。

  ```
  ovconfchg -ns eaagt.integration.nnm -set OPC_SNMP_SET_SEVERITY TRUE
  ```

- 新しい OID の値を設定するには、以下のコマンドを実行します。

  ```
  ovconfchg -ns eaagt.integration.nnm -set OPC_SNMP_OVERRIDE_SEVERITY_OID<OID >
  ```

  <OID > はオブジェクト ID です。ここで、OID は SNMP トラップの重要度レベルを引き出すのに使用されます。

デフォルトでは、重要度レベルはデフォルトの OID **“.1.3.6.1.4.1.11.2.17.19.2.2.12”** から引き出されます。以下のコマンドを実行することで、デフォルトの OID を変更できます。

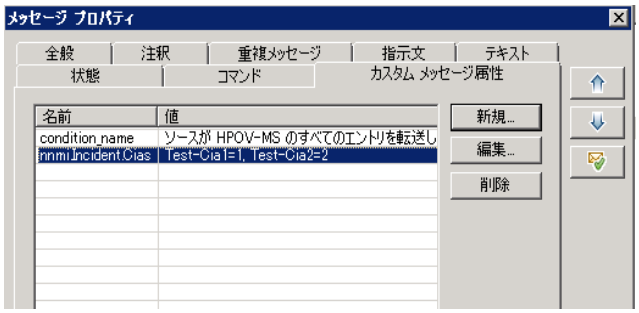
```
ovconfchg -ns eaagt.integration.nnm -set OPC_SNMP_OVERRIDE_SEVERITY_
OID<.1.3.6.1.4.1.11.2.17.19.2.2.22>
```

これ以降、メッセージの重要度は新しい OID 値 <.1.3.6.1.4.1.11.2.17.19.2.2.22> に基づくことになります。

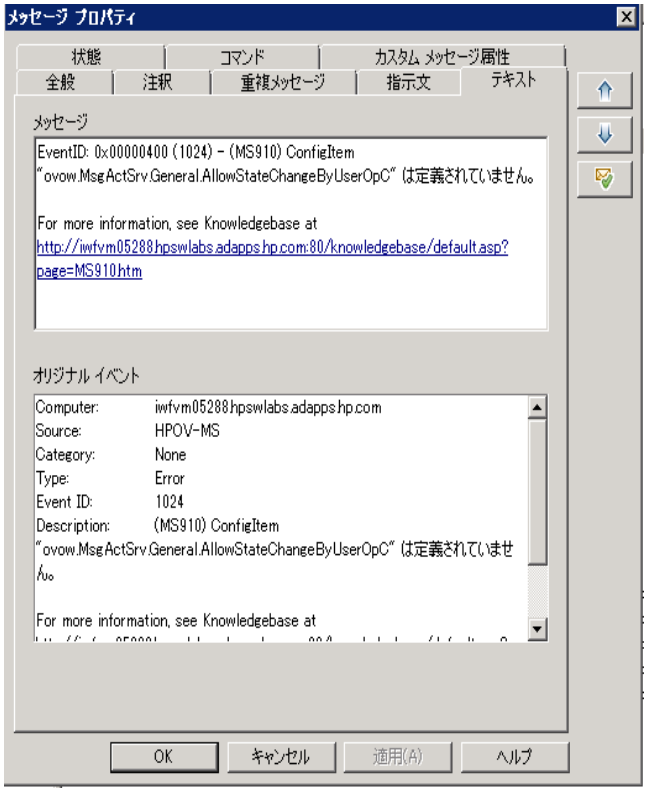
HPOM コンソール上に CMA を追加するための NNMi Norhbound インターフェイスとの統合

NNMi Northbound 統合で SNMP トラップを転送する場合、トラップにはカスタム インシデント属性 (CIA) を示す varbind が含まれます。HPOM ポリシー内にカスタム メッセージ属性 (CMA)[1] を設定した場合、HPOM コンソール上の 1 つの CMA に NNMi の CIA が表示されます。リストには CMA 名と値のペアが表示されます。これはデフォルトの動作です。[CMA] タブのメッセージは、次のように表示されます。

[1]カスタム メッセージ属性 (CMA) には、意味のある情報を指定できます。1 つのメッセージには複数の CMA をアタッチできます。



HPOM ポリシー内に CMA を設定していない場合、NNMi CIA メッセージはテキスト メッセージとして表示されます。



opctrapi を設定すると、SNMP トラップ インターセプタは CIA にアクセスし、OM メッセージ内で各 CIA をカスタム メッセージ属性 (CMA) として表示します。

HPOM コンソール上での NNMi CIA から CMA を作成するための SNMP トラップ インターセプタの設定

以下を設定することで、opctrapi を設定して、[CMA プロパティ] タブに CMA 名と値ペアとして NNMi CIA を表示することができます。

- OPC_SPLIT_NNM_CUSTOM_ATTR: 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** に設定することで、varbind **(.1.3.6.1.4.1.11.2.17.19.2.2.20)** に存在するすべての NNMi CIA 値は個別の CMA として表示されます。

- OPC_SPLIT_NNM_CUSTOM_ATTR_MAX: この変数は**オプション**です。このプロパティは、OPC_SPLIT_NNM_CUSTOM_ATTR 変数が TRUE に設定されている場合のみ有効です。この変数は、HPOM メッセージが読み込み、解釈可能な NNMi カスタム属性数を定義します。デフォルトでは、この値は 20 に設定されます。したがって、20 個の CMA のみがそれぞれの値とは別に表示され、残りの NNMi CIA は 1 つの CMA 内に表示されます。 この値は必要に応じて指定できます。

NNMi CIA から CMA を作成するために opctrapi 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。

- opctrapi で varbind (**.1.3.6.1.4.1.11.2.17.19.2.2.20**) に表示されるリストから個別に CMA を読み込み、作成するには、以下のコマンドを実行します。

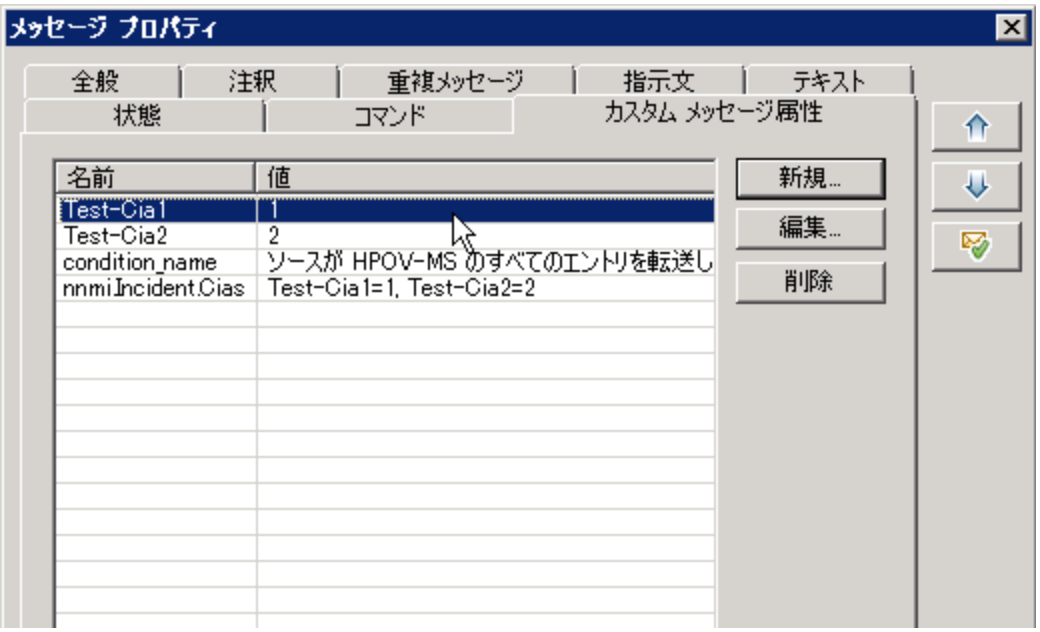
  ```
  ovconfchg -ns eaagt.integration.nnm -set OPC_SPLIT_NNM_CUSTOM_ATTR TRUE
  ```

- メッセージを読み込み、CMA 名と値ペアを作成するために変数の値を設定するには、以下のコマンドを実行します。

  ```
  ovconfchg -ns eaagt.integration.nnm -set OPC_SPLIT_NNM_CUSTOM_ATTR_MAX <Value>
  ```

  <Value> は整数です。デフォルトでは、この値は 20 です。

設定後、[CMA] タブ内のメッセージは、名前と値の各属性とともに表示されます。さらに、NNMi CIA のメッセージ全体 (`nnmi.Incident.Cias`) がリストに表示されます。



SNMP トラップのソースを割り当てるための NNMi トラップ転送インターフェイスとの統合

以前のバージョンでは、SNMP v2c トラップをノード (ノード上では Operations Agent が利用可能) に転送するために NNMi を使用すると、必ず HPOM メッセージ内のソース ノード名は、NNMi ノード名として表示されます。

今回のバージョンでは、HP Operations Agent は NNMi 転送インターフェイスと統合されています。NNMi 転送インターフェイスは、トラップの転送元のソースを識別するために varbind を追加します。

トラップを生成したソース ノードを引き出すように opctrapi を設定できます。HPOM メッセージが生成されると、SNMP トラップ インターセプタ (opctrapi) は、トラップを生成したノードとして、NNMi ノード名ではなくソース ノード名を設定します。

ソース ノード名を引き出すための SNMP トラップ インターセプタの設定

以下のプロパティを設定できます。

OPC_NODENAME_FROM_NNM_FRWD_TRAP: 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** に設定することで、すべてのトラップが varbind “.1.3.6.1.4.1.11.2.17.2.19.1.1.2.0” および .1.3.6.1.4.1.11.2.17.2.19.1.1.3.0 について検索されます。opctrapi は次の varbind を使用します。

- (.1.3.6.1.4.1.11.2.17.2.19.1.1.2.0) - この varbind は IP アドレス タイプを特定するのに使用されます。
- (.1.3.6.1.4.1.11.2.17.2.19.1.1.3.0) - この varbind はソース ノードとしてノードを設定するために IP アドレスを引き出すのに使用されます。

NNMi が SNMPv2 イベントを転送する際にソース ノード名を割り当てるように opctrapi を設定するには、以下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て、opctrapi で varbind を読み込み、トラップの転送元のノードを割り当てるようにします。

   ```
   ovconfchg -ns eaagt.integration.nnm -set OPC_NODENAME_FROM_NNM_FRWD_TRAP TRUE
   ```

   注: 変数を設定しないか、変数を False に設定すると、opctrapi はトラップを生成した元のソース (ノード名) を引き出しません。

メッセージ ストームの抑制の設定

短い時間間隔内で膨大な量の新着メッセージを管理サーバー上で受信し、有効なメッセージ ブラウザにこれらのメッセージが殺到すると、メッセージ ストームが発生します。このような現象が発生

すると、管理サーバーが停止することがあります。

HP Operations Agent では、管理ノードのメッセージ ストームを検出、抑制できます。以下のプロパティを設定します。

- OPC_MSG_STORM_DETECTION: この変数は**必須**です。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** に設定することで、メッセージ ストームの検出を有効にできます。
- OPC_MSG_STORM_DETECTION_CATEGORY: この変数は**必須**です。このプロパティは、OPC_MSG_STORM_DETECTION 変数が TRUE に設定している場合のみ有効です。POLICY、MSGGROUP、APPLICATION、OBJECT、SEVERITY などのメッセージ属性のいずれかにこの変数を設定できます。
- OPC_MSG_STORM_RATE: この変数は**必須**です。この変数は以下のパラメータを定義します。
  - Threshold: 受信メッセージの制限値を定義します。受信メッセージ数が定義した制限値を超えると、メッセージ ストーム状態が検出されます。
  - Time: メッセージ ストーム状態を検出するために受信メッセージをカウントする間隔
  - Reset: メッセージ数が定義した値を下回っているときの制限値を定義します。このパラメータは、ストーム状態が解決されたときを検出するのに使用されます。
- OPC_SEND_INTERNAL_MSG_ON_MSGSTORM: この変数は**オプション**です。この変数は、内部メッセージを送信するか、または停止するかを定義します。デフォルトでは、この値は TRUE に設定されます。
- OPC_SUPPRESS_MSG_ON_MSG_STORM: この変数は**オプション**です。この変数は、メッセージを送信するか、または抑制するかを定義します。デフォルトの動作では、しきい値条件が満たされ、メッセージ ストームが検出された場合、しきい値を超えるすべてのメッセージが抑制されます。デフォルトでは、この値は TRUE です。
- OPC_MSG_STORM_TRACE_SUPPRESSED_MSGS: この変数は**オプション**です。この変数は、<OPC_SUPPRESS_MSG_ON_MSG_STORM> を TRUE に設定したときのみにメッセージをログ ファイルに記録するかどうかを定義します。 デフォルトでは、この値は FALSE です。

メッセージ エージェントがメッセージ ストームを検出したとき、またはメッセージ ストームを解決したとき、メッセージはログ ファイル (System.txt) に記録されます。

Windows の場合:

`%OvDataDir%\log`

HP-UX/Linux/Solaris の場合:

`/var/opt/OV/log`

これらのパラメータは `eaagt.msgstorm` 名前空間で使用できます。

管理ノード上でメッセージ ストームを検出することの利点を、以下に示します。

- ECS サーキットが不要になる
- メッセージ ストームをソース ノードで識別する
- 設定手順が容易になり、各種のメッセージ属性に対して設定できる

注: 管理ノード上でメッセージ ストームを検出、抑制する例です。

OPC_MSG_STORM_DETECTION パラメータを TRUE に設定することで、ストーム状態を検出できます。

メッセージ ストーム状態を有効にしてから、以下のパラメータを定義します。

ポリシーとして OPC_MSG_STORM_DETECTION_CATEGORY。配布されるポリシーは、Opclepolicy と Opcmsgipolicy です。

OPC_MSG_STORM_RATE パラメータを設定するには、パラメータを設定するための受信メッセージ レートを計算します。「メッセージ ストームの抑制の設定」(37ページ)を参照してください。メッセージ レートごとに値を設定できます。

OPC_MSG_STORM_RATE パラメータの値を、Threshold=100、Time=20、Reset Value=50 と設定します。

Opclepolicy は 50 個のメッセージを送信し、Opcmsgipolicy は 101 個のメッセージを送信します。Opclepolicy からの 101 個のメッセージは、設定したしきい値 (100) を超えるため、ストームが検出されます。

メッセージ ストームが検出され、OPC_SEND_INTERNAL_MSG_ON_MSGSTORM パラメータによって解決されると、通知を取得できます。デフォルトでは、この値は TRUE に設定されます。

デフォルトでは、100 を超えるメッセージ数は抑制されるか無視されます。パラメータ (OPC_SUPPRESS_MSG_ON_MSG_STORM) を FALSE に設定すると、抑制されているメッセージを取得できます。

パラメータ (OPC_MSG_STORM_TRACE_SUPPRESSED_MSGS) を設定することで、抑制されたメッセージをログ ファイルに記録できます。

メッセージ ストームの検出と抑制 の設定

注: この設定に関するパラメータを変更したときには、必ずメッセージ エージェントを再起動してください。メッセージ エージェントを再起動すると、メッセージ ストームの検出に関連する内容はリセットされます。

メッセージ ストームの検出または抑制を設定するには、以下の手順を実行します。

1. 適切な権限でノードにログオンするか、リモートで設定します。
2. メッセージ ストームの検出を有効にするには、以下のコマンドを実行します。

   ```
   ovconfchg -ns eaagt.msgstorm -set OPC_MSG_STORM_DETECTION TRUE
   ```

   デフォルト値は FALSE に設定されます。
3. メッセージのカテゴリを設定するには、以下のコマンドを実行します。

   ```
   ovconfchg -ns eaagt.msgstorm -set OPC_MSG_STORM_DETECTION_CATEGORY<CATEGORY>
   ```

   注: これは必須の変数です。

カテゴリは、以下のいずれか **1 つ**を定義する必要があります。

- POLICY
- GROUP
- APPLICATION
- OBJECT
- SEVERITY

注: 定義できるのは、使用可能なオプションのいずれか 1 つのみです。各値を組み合せて使用することはできません。

4. メッセージ レートの変数を設定するには、以下のコマンドを実行します。

```
ovconfchg -ns eaagt.msgstorm -set OPC_MSG_STORM_
RATE<ThresholdValue:Time:ResetValue>
```

注: これらの変数は必須です。

変数は以下のように定義されます。

- しきい値 - メトリック値を設定します。設定した時間間隔におけるメッセージ数がしきい値を超えた場合、この状態をメッセージ ストーム状態といいます。このような状態になると、メッセージ エージェントは、リセット条件が満たされるまでメッセージを抑制します。しきい値は、設定したカテゴリで使用可能なすべてのグループに対して、メッセージ ストーム状態であるかどうかを確認します。このパラメータを設定するためにメッセージ レートを確認できます。「メッセージ ストームの検出と抑制 の設定」(39ページ)を参照してください。

例:カテゴリを「SEVERITY (重要度)」に設定します。このメッセージの重要度カテゴリでは、Critical (危険域)、Major (重要警戒域)、Minor (警戒域)、Warning (注意域)、Normal (正常域) など、5 つのグループ タイプがあります。メッセージは重要度に基づいてグループ分けされ、設定したしきい値について確認されます。

しきい値は 100 に設定されています。Critical (危険域)、Major (重要警戒域)、Minor (警戒域)、Warning (注意域)、Normal (正常域) の重要度で受信するメッセージは、それぞれ 80、60、40、50、110 に設定されています。ここで、メッセージ数 (110) は設定したしきい値 (100) を超えているため、メッセージ ストームは「正常域」状態でのみ検出されます。正常域の重要度を持つ残りの 10 個のメッセージは抑制されます。

- 時刻 - 時間間隔を秒単位で設定します。着信メッセージのカウントが記録される時間間隔です。推奨値は 900 秒未満です。
- リセット値 - しきい値と同じまたはそれ以下にする必要がある数値を設定します。メッセージ ストームの検出は、次の条件が満たされる場合にリセットされます。

- メッセージ レートが設定した時間間隔で確認される場合。
- メッセージ レートがリセット値を下回る必要がある場合。

メッセージ ストーム状態に対するパラメータを設定する例。

set OPC_MSG_STORM_DETECTION=TRUE

set OPC_MSG_STORM_DETECTION_CATEGORY=SEVERITY

set OPC_MSG_STORM_RATE=100:60:45

ここで、Threshold 値は 100、Time は 60、Reset 値は 45 です。

メッセージ ストームの検出後、着信メッセージ数が設定した定期的な間隔 (60 秒間隔) で確認されます。設定した時間間隔で、メッセージがリセット値 (45) を下回ると、メッセージ エージェント (opcmsga) はメッセージを無視する処理を停止し、これによりメッセージ ストーム状態が終了します。

注: 手順 5、6、7 はオプションの手順です。

5. 内部メッセージを受信および停止するには、以下のコマンドを実行します。

   `ovconfchg -ns eaagt.msgstorm -set OPC_SEND_INTERNAL_MSG_ON_MSGSTORM<Value>`

   <Value> には、以下のいずれか 1 つを定義する必要があります。

   - TRUE: メッセージ ストーム状態が検出されたとき、およびその状態が解決されたときには必ず内部メッセージが生成されます。デフォルト値は TRUE です。
   - FALSE: メッセージ ストーム状態に対しても、その状態が解決されてもどちらに対しても、内部メッセージは生成されません。

6. メッセージ ストーム状態になった場合に、メッセージを抑制または受信するには、以下のコマンドを実行します。

   `ovconfchg -ns eaagt.msgstorm -set OPC_SUPPRESS_MSG_ON_MSG_STORM<Value>`

   値は、以下のいずれか 1 つを定義する必要があります。

   - TRUE: メッセージ ストーム状態になると、メッセージ エージェントはメッセージを抑制します。デフォルト値は TRUE です。
   - FALSE: メッセージ ストーム状態になっても、メッセージ エージェントはメッセージを抑制しません。

7. OPC_SUPPRESS_MSG_ON_MSG_STORM 変数を FALSE に設定した場合は、**この手順を省略します**。

   メッセージ ストーム状態になった場合、抑制されたメッセージを記録するには、以下のコマンドを実行します。

   `ovconfchg -ns eaagt.msgstorm -set OPC_MSG_STORM_TRACE_SUPPRESSED_MSGS <Value>`

   値は、以下のいずれか 1 つを定義する必要があります。

- TRUE: メッセージ エージェントは抑制したすべてのメッセージをログ ファイルに保存します。
- FALSE: メッセージ エージェントは抑制したメッセージを保存しません。デフォルトでは、この値は FALSE です。

抑制したメッセージは、次のディレクトリの (msgsuppress.log <プロセスのプロセス ID>) にあります。

Windows の場合:

```
%OvDataDir%\tmp\OpC
```

HP-UX/Linux/Solaris の場合:

```
/var/opt/OV/tmp/OpC
```

8. 以下のコマンドを実行して、メッセージ エージェントを再起動します。

```
ovc -restart opcmsga
```

メッセージ レートの確認

メッセージ レートは、システム全体に及ぶメッセージの平均レートを測定するのに使用します。平均メッセージ レートを確認すると、理想のしきい値制限とリセット制限を設定して、メッセージ ストーム状態を検出し抑止することができます。メッセージ レートの確認は、メッセージ ストーム検出を設定した後にのみ実行できます。

特定の時点でメッセージ レートを確認するには、以下のコマンドを実行します。

Windows の場合:

```
%OVInstallDir%/lbin/eaagt/opcmsga -message_rate
```

HP-UX/Linux/Solaris の場合:

```
/opt/OV/lbin/eaagt/opcmsga -message_rate
```

AIX の場合:

```
/usr/lpp/OV/lbin/eaagt/opcmsga -message_rate
```

バックアップ サーバーの設定

メッセージ エージェントは、HP Operations Manager (HPOM) として使用しているプライマリ サーバーにメッセージを送信します。環境内に複数の HPOM サーバーが存在する場合、環境内のバックアップ サーバーとして別のサーバーを設定することもできます。ご使用の環境にバックアップ サーバーを設定している場合、プライマリ サーバーへの通信リンクがダウンしてプライマリ サーバーにメッセージを送信できなくなると、メッセージ エージェントはバックアップ サーバーにメッセージを送信します。

たとえば、Manager-of-Manager (MoM) シナリオのように、プライマリ サーバーとバックアップ サーバーの両方の間のメッセージ同期における負荷分散用に、バックアップ サーバーを設定することも

できます。MoM シナリオの詳細については、『HP Operations Manager コンセプト ガイド UNIX オペレーティング システム向け』または『HP Operations Manager オンライン ヘルプ』を参照してください。環境内の別の HPOM サーバーをバックアップ サーバーにすることができます。複数のサーバーをバックアップ サーバーとして設定することができます。

以下の変数に値を設定することで、HP Operations Agent がバックアップ サーバーにメッセージを送信できるようにすることができます。

- OPC_BACKUP_MGRS: バックアップ サーバーとして設定する 1 つのサーバーまたはサーバー リストを指定できます。リスト内の値はカンマまたはセミコロンで区切りします。
- OPC_BACKUP_MGRS_FAILOVER_ONLY: この変数の値を **TRUE** に設定した場合、プライマリ サーバーがダウンすると、メッセージ エージェントはメッセージをバックアップ サーバーにのみ転送します。値を **FALSE** に設定した場合、プライマリ サーバーの状態に関係なく、メッセージはバックアップ サーバーに送信されます。この変数のデフォルト値は **FALSE** です。

バックアップ サーバーのリストを設定した場合、初期化時にメッセージ エージェントはバックアップ サーバーのリストを作成し、OPC_BACKUP_MGRS_FAILOVER_ONLY 変数に設定した値を確認します。 メッセージが msgagtdf ファイルに到着すると、OPC_BACKUP_MGRS_FAILOVER_ONLY 変数の値は次のようになります。

- **FALSE**: メッセージ エージェントはプライマリ サーバー、次にバックアップ サーバーの順にメッセージを送信します。メッセージが正常に両方のサーバーに送信されると、メッセージのエントリは msgagtdf ファイルから削除されます。
- **TRUE**: メッセージ エージェントはプライマリ サーバーにメッセージを送信します。メッセージの送信が失敗すると、メッセージ エージェントはメッセージをバックアップ サーバーに転送することを試みます。一覧されているバックアップ サーバーが複数存在し、メッセージが少なくとも 1 つのバックアップ サーバーに配信されると、メッセージ エントリは `msgagtdf` ファイルから削除されます。メッセージ エージェントは他のバックアップ サーバーにメッセージを送信しなおすことはなく、またプライマリ サーバーが復帰したときにプライマリ サーバーにメッセージを送信することもしません。

メモ: OPC_BACKUP_MGRS_FAILOVER_ONLY を **TRUE** に設定した場合、ローカル自動アクションを起動したメッセージがプライマリ サーバーに送信され、その後にプライマリ サーバーがダウンすると、メッセージ エージェントはアクション応答をバックアップ マネージャに送信します。しかし、このアクション応答はバックアップ サーバーにはアクションを起動した元のメッセージが存在しないため、バックアップ サーバーによって破棄されます。

前提条件:

- HP Operations Agent ノード上に、バックアップ サーバーの信頼された証明書をインストールする必要があります。
- バックアップ サーバー上に、プライマリ サーバーの信頼された証明書をインストールする必要があります。
- HP Operations Agent ノードをバックアップ サーバーに追加する必要があります。

変数の値を設定するには、次の手順を実行します。

1. root または管理者としてノードにログオンします。
2. 以下のコマンドを実行します。
   - バックアップ サーバーを設定するには、以下のコマンドを実行します。

     `ovconfchg -ns eaagt -set OPC_BACKUP_MGRS` <servername>

     <servername> はバックアップ サーバーの名前です。たとえば、「abc.ind.hp.com」などです。

     バックアップ サーバーのリストを設定するには、以下のいずれかのコマンドを実行します。

     - `ovconfchg -ns eaagt -set OPC_BACKUP_MGRS` <servername1>,<servername2>,...,<servernameN>
     - `ovconfchg -ns eaagt -set OPC_BACKUP_MGRS` <servername1>;<servername2>;...;<servernameN>

       バックアップ サーバーの順番は、サーバー名の指定順序に依存します。前述した例では、<servername1> は最初のバックアップ サーバーとして動作し、<servername2> は 2 番目のバックアップ サーバーとして動作します。

   - OPC_BACKUP_MGRS_FAILOVER_ONLY 変数の値を変更するには、以下のコマンドを実行します。

     `ovconfchg -ns eaagt -set OPC_BACKUP_MGRS_FAILOVER_ONLY TRUE`

     デフォルト値は **FALSE** です。

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 Agent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、デフォルトの構成でノードで動作を開始します。

Perfd プロセスのためのライセンスの確認

このコンポーネントは HP Ops OS Inst to Realtime Inst LTU、Glance Pak Software LTU、または Glance Software LTU を使用する場合にのみ使用できます。詳細については、『HP Operations Agent ライセンス ガイド』を参照してください。

perfd プロセスによるデータ収集は、ライセンスが有効であることを確認した後にのみ開始されます。ライセンスが有効でないと、プロセスはアイドル状態になります。すべてのアクティビティはログ ファイルに記録されます。

設定の変更

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%

HP-UX/Linux/Solaris の場合:

/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 の場合:** %ovdatadir% **HP-UX/Linux/Solaris の場合:** /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
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

(続き)

パラメータ	説明	デフォルト値
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 IPv6 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。 **注:** perfd が明示的に IPv4 接続のみを受け付けるようにする場合、**perfd.ini** ファイルで IPv4 値を true に設定します。	False
logsize	このパラメータは、ファイル サイズを指定します。値がこのファイル サイズを超過すると、perfd がそのログ ファイルまたはトレース ファイルをロールオーバーします。4096 バイト未満のサイズを指定した場合は、無視されます。	1048576 バイト
add	データを収集する必要があるメトリック クラスのカンマ区切りリストを指定するには、このパラメータを使用します。このパラメータは、lightweight パラメータが True に設定されている場合に、個々のメトリック クラスを追加するときに役立ちます。	
exclude	データを収集しないメトリック クラスのカンマ区切りリストを指定するには、このパラメータを使用します。このパラメータは、lightweight パラメータが False に設定されている場合に、個々のメトリック クラスを除外するときに役立ちます。	

デフォルト設定を変更する手順は以下のとおりです。

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 Agent を再起動して設定変更を有効にします。

IPv6 接続上での HP Operations Agent の監視

HP Operations Agent 12.00 は、インストールが完了すると、サポートされている IP 設定を特定し、対応する IP アドレスとバインドします。IPv6 接続の場合、追加の設定は不要です。デュアル スタック ノードの場合、ノードがサポートされているサーバーの IP アドレスを特定します。

サーバー	デュアル スタック ノード
IPv4	IPv4 を使用
IPv6	IPv6 を使用
デュアル スタック	IPv6 を使用

注: サーバーとノードの両方がデュアル スタックの場合、IPv6 通信が優先されます。

アクセスの制限

RTMA コンポーネントを使用して、ローカル システムのリアルタイム パフォーマンス データに他のシステムがアクセスしないように HP Operations Agent を設定できます。

cpsh、padv、mpadv ユーティリティを使うと、エージェント ノードのリアルタイム パフォーマンス データにリモートでアクセスできます。cpsh ユーティリティを使用して、perfd プロセスが実行されている任意のリモート システムにあるリアルタイム パフォーマンス データを表示できます。

padv または mpadv ユーティリティを使用して、perfd プロセスが実行されている任意のリモート システムで adviser を実行できます。

これらのユーティリティはリアルタイム パフォーマンス データにアクセスするために perfd プロセスに依存します。

他のシステムが perfd プロセスを起動することによってローカル システムのリアルタイム パフォーマンス データにアクセスしないようにするには、以下の手順に従います。

注: この手順では、HP Performance Manager の Diagnostic View から、システムのリアルタイム パフォーマンス データへのアクセスを禁止するようには設定されません。

1. 管理者または root としてシステムにログオンします。以下のディレクトリに移動します。

   Windows の場合:

   `%OvDataDir%\`

   HP-UX/Linux/Solaris の場合:

   `/var/opt/perf/`

2. 環境にある (HP Operations Agent が利用可能な) 他のすべてのシステムからは、このシステムのリアルタイム パフォーマンス データにアクセスできなくなりました。

3. この場所に空のファイルを作成して、そのファイルを **authip** として保存します。

4. このシステムのリアルタイム パフォーマンス データに別のシステムからアクセスできるようにするには、テキスト エディタで **authip** ファイルを開いて、システムの FQDN または IP アドレスを追加し、ファイルを保存します。複数の FQDN または IP アドレスを指定できます (エントリは改行で区切る必要があります)。

5. **perfd** を再起動して、変更内容を有効にします。

authip ファイルで指定したシステムは、このシステムのリアルタイム パフォーマンス データにアクセスできるようになりました。

6. すべてのシステムからアクセスできるようにするには、**authip** ファイルを削除します。

エージェント ユーザーの設定

インストール後、HP Operations Agent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 アカウントで稼働し始めます。ただし、root やローカル システム ユーザーよりも少ない権限を持つデフォルト以外のユーザーで実行するように HP Operations Agent を設定できます。

運用監視コンポーネント は root 以外のユーザーやローカル システム ユーザー以外のユーザーだけで実行し、残りのコンポーネントは、デフォルトの root またはローカル システム ユーザーで実行できます。

使用するユーザー アカウントに応じて、次のエージェント操作モードを設定できます。

- **権限のないモード:** HP Operations Agent のすべてのコンポーネントは、root やローカル システム ユーザーよりも少ない権限を持つデフォルト以外のユーザー アカウントで実行されます。

  注: HP-UX では権限のないモードで HP Operations Agent を実行することはできません。

  システム上に HP Operations Agent と NNMi の両方が共存している場合、HP Operations Agent を権限のないユーザー モードで実行してはなりません。

- **root モード:** HP Operations Agent のすべてのコンポーネントは、デフォルト ユーザー (root またはローカル システム)で実行されます。これは、エージェントの操作モードのデフォルトです。
- **混合モード:** Performance Collection Component はデフォルト ユーザーで実行され、運用監視コンポーネント はローカル ユーザー アカウントで実行されます。

  注: 混合モードで実行する場合、root または権限のあるユーザーがエージェント プロセスを開始することを推奨します。

HP Performance Manager がインストールされているシステムで、デフォルト以外のユーザーで実行されるようにエージェントが設定されている場合、HP Performance Manager の `OvTomCatB` サービスは、デフォルト以外のエージェント ユーザーで実行が開始されます。

さらに、HPOM 管理対象環境では、エージェントを実行するように設定されているユーザーとは異なるユーザーが自動コマンドまたはオペレータによるコマンドを実行するようにエージェントを設定することもできます。

デフォルトではないユーザーを使用するための要件

デフォルトではないエージェント ユーザーを使用するには、以下の要件を満たす必要があります。

- **Windows のみの要件:**
  - ユーザーには `HKEY_LOCAL_MACHINE/Software/Hewlett-Packard/OpenView` レジストリ キーのフル コントロールが必要です。
  - ユーザーには `HKEY_LOCAL_MACHINE/Software/Microsoft/WindowsNT/CurrentVersion/Perflib` レジストリ キーへの読み取りアクセス権が必要です。
  - ユーザーには次の権限が必要です。
    - サービスとしてログオン - Performance Collection Component サービスの起動と停止を行います。
    - 監査およびセキュリティ ログの管理 - ユーザーはイベント ビューアのセキュリティ ログで監査済みのイベントを表示できます。この権限を持つユーザーは、セキュリティ ログを表示およびクリアできます。
    - オペレーティング システムの一部として動作 - エージェント プロセスを制御します。
    - プロセスレベル トークンの置き換え - エージェント プロセスを root 以外のユーザーとして作成します。
- ポリシーを使用してログ ファイルを監視するには、エージェント ユーザーにログ ファイルの読み取り権限が必要です。

  注: Windows の場合、エージェント ユーザーには権限がないため、ログ ファイルを読み取ることができません。
- 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するには、エージェント ユーザーに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。

デフォルトではないユーザーを使用するときの制限事項

デフォルトではない (root 以外や権限がない) エージェント ユーザーを使用するには、以下の制限事項があります。

- 権限のないユーザー モードは HP-UX プラットフォームではサポートされていません。
- 権限のないユーザー モードは、HPOM 管理サーバーではサポートされていませんが、混合ユーザー モードは、HP Operations Manager for Unix、Linux、および Solaris のバージョン 9.20 からサポートされています。
- 権限のないユーザー モードおよび混合モードは AIX WPAR ではサポートされていません。
- Xen、KVM、VMware vSphere、Hyper-V、および他の仮想化ドメインの BYLS メトリック データは収集できません。
- デフォルトでは、権限のないユーザー モードおよび混合モードのエージェント ユーザーは、監視対象のログ ファイルの読み取り権限がありません。
- デフォルトでは、権限のないユーザー モードおよび混合モードのエージェント ユーザーは、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する権限がありません。
- 権限のないユーザー モードおよび混合モードのエージェント ユーザーに管理者権限がない場合、HP Operations Smart Plug-in 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。
- 権限のないユーザーが所有するプロセスのすべてのインスタンスで、HP Operations Agent は、`PROC_REGION_*` または `PROC_FILE_*` で始まるメトリックを収集できません。また、オペレーティング システムによっては、`ovbbccb`、`sshd` などのより高い権限で実行されているプロセスを、権限のないモードで利用できる場合やできない場合があります。
- Windows の場合、エージェント ユーザー以外のユーザーに所有されているプロセスで、`PROC_USER_NAME` メトリックは `Unknown` と表示されます。
- AIX の場合、デフォルトではないユーザーを使用するようにエージェントを設定すると、コマンド ライン コンソール (または `/var/opt/OV/log` ディレクトリの `oainstall.log` ファイル) に次のエラー メッセージが表示されることがあります。

  ```
  Product activation failure.Refer to the log file for more details.
  ```

  このエラーは無視してください。
- Solaris の場合、HP Operations Agent が取得できるプロセスの詳細は最大 80 文字で、これは Solaris の制限事項です。`opcmona` は `/proc/pid/psinfo` ファイルを読み取り、結果を構造内に保存します。拡張情報が必要な場合、システムは `/proc/pid/as` ファイルを読み取ります。エージェントが root 以外のユーザーとして実行中で、`/proc/pid/as` ファイルを開く権限がない場合は、プロセスの詳細が `psinfo` で利用可能な限定された情報と比較されます。

インストール中のエージェント ユーザーの設定

インストール時に、システムでデフォルト以外のユーザー (root またはローカル システム以外) で HP Operations Agentを実行するように設定できます。このように設定するには、手動でのインストールに使用するプロファイル ファイルを使うか、HPOM によるリモート インストールに使用するデフォルト ファイルを使ってエージェントをインストールする必要があります。インストール時にこのように設定できない場合は、インストール後の設定手順を実行して、デフォルトのエージェント ユーザーを変更します (「インストール後のエージェント ユーザーの設定」(53ページ) を参照)。

注: HPOM コンソールから Windows ノードにエージェントをリモートでインストールする場合、この手順は使用できません。HPOM コンソールから Windows ノードにエージェントをインストールしているときに、非アクティブ モードでエージェントをインストールし、インストール後の設定手順を使用して、デフォルト以外のユーザーでエージェントが実行されるように設定します。詳細については、『HP Operations Agent and HP Operations Smart Plug-ins for Infrastructure インストール ガイド』を参照してください。

デフォルト以外のユーザーで実行されるようにインストール中にエージェントを設定するには、以下の手順に従います。

1. ユーザーがシステムで作成されていることおよびとユーザーがすべての要件を満たしていることを確認します。
2. HP Operations Agentを手動でノードにインストールする場合は、**プロファイル** ファイルを作成します。
   a. エージェントをインストールするシステムで新しいファイルを作成して、そのファイルをテキスト エディタで開きます。
   b. 次の文を指定して、エージェントの操作モードを指定します。

      root 以外またはローカル システム以外のアカウントでエージェントを実行するには、次のように入力します。

      set eaagt:MODE= NPU

      root 以外またはローカル システム以外のアカウントで 運用監視コンポーネント のみを実行するには、次のように入力します。

      set eaagt:MODE= MIXED
   c. 次の文を入力します。

      set eaagt:OPC_RPC_ONLY=TRUE
   d. 混合モード (手順 b で `set eaagt:MODE=MIXED` と入力した場合)、次の文を入力します。

      set eaagt:NPU_TASK_SET=EVENT_ACTION
   e. Windows のみ。次の文を入力します。

      set eaagt:OPC_PROC_ALWAYS_INTERACTIVE=NEVER

      注: この手順は、エージェントが NPU または MIXED モードで実行中に、HPOM のノード上で自動アクションとオペレータ起動アクションを正常に実行するために必要です。
   f. 次の文を入力します。

      注: これは、UNIX/Linux ノードに必須の手順です。Windows ノードではこの手順を省略できますが、Windows ノードでもこれらの手順を設定することをお勧めします。

      set bbc.cb:SERVER_PORT=<Comm_Port>

      set eaagt:SNMP_TRAP_PORT=<SNMP_Port>

> **注:** エージェントのデフォルトの通信ポートは 383 であり、UNIX/Linux の root 以外のユーザーは 1024 未満のポートにアクセスできないため、この手順を実行して、デフォルト以外の通信ポートをエージェントに割り当てる必要があります。

上記の説明

<Comm_Port> は選択した通信ポート番号です。

<SNMP_Port> は HP Operations Agent で SNMP トラップを受け取るポートです。

これらのポートは 1024 より大きい番号である必要があります。UNIX/Linux の root 以外のユーザーは 1024 未満のポートにアクセスできないためです。

g. 次の文を入力します。

set ctrl.sudo:OV_SUDO_USER=<User_Name>

set ctrl.sudo:OV_SUDO_GROUP=<User_Group>

ここで、<User_Name> はデフォルト以外のユーザーの名前であり、<User_Group> はデフォルト以外のユーザーが属するグループです。

HPOM コンソールからエージェントをリモートでインストールする場合、インストールのデフォルトを次のように設定します。

> **注:** Windows ノードにエージェントをインストールする場合、この手順は使用できません。Windows ノードでは、エージェントをノードに手動でインストールするか、インストール後の設定を実行して、エージェント ユーザーを設定してください。

a. 以下のディレクトリに移動します。

HPOM for Windows の場合:

<share_dir>`\conf\PMAD`

HPOM for HP-UX/Linux/Solaris の場合:

`/etc/opt/OV/share/conf/OpC/mgmt_sv`

b. 次のファイルを保存します。

HPOM for Windows の場合:

`agent_install_defaults.cfg.sample` ファイルを `agent_install_defaults.cfg` として保存します。

HPOM for HP-UX/Linux/Solaris の場合:

`bbc_inst_defaults.sample` ファイルを `bbc_inst_defaults` ファイルとして保存します。

c. テキスト エディタでファイルを開きます。

d. 次の内容を追加します。

```
[eaagt]
<node_details> :MODE=<MODE>
<node_details> :OPC_RPC_ONLY=TRUE
```

```
<node_details>:NPU_TASK_SET=EVENT_ACTION
<node_details>:SNMP_TRAP_PORT=<SNMP_Port>
[ctrl.sudo]
<node_details> :OV_SUDO_USER=<User_Name>
<node_details> :OV_SUDO_GROUP=<User_Group>
[bbc.cb]
<node_details>:SERVER_PORT=<Comm_Port>
```

ここで、

<node_details> は 1 つまたは複数のノード名または IP アドレスに一致するパターンです。標準の HPOM パターン構文を使用します。たとえば、次のようになります。

- node1.example.com は、node1.example.com という文字列が含まれる名前を持つ任意のノードに一致します
- example.com$ は、example.com で終わる名前を持つ任意のノードに一致します
- ^192.168.<<#> -lt 10> は、IP アドレスが 192.168.0.0 から 192.168.9.255 の範囲にある任意のノードに一致します

<MODE> はエージェントの操作モード (NPU または混合) です

<User_Name> はデフォルト以外のユーザーの名前です

<User_Group> はデフォルト以外のユーザーが属するグループです

<Comm_Port> は選択した通信ポート番号です。これは 1024 より大きい番号である必要があります。管理サーバーもこのノードと通信するときにポート <Comm_Port> に接続するように設定する必要があります。詳細については、『HP Operations Agent and HP Operations Smart Plug-ins for Infrastructure インストール ガイド』の「通信ブローカ ポートの設定」を参照してください。

<SNMP_Port> は HP Operations Agent で SNMP トラップを受け取るポートです。このポートは、さまざまなソースの SNMP トラップを <SNMP_Port> に送信できるように、1024 より大きい番号である必要があります。

e. ファイルを保存します。

3. HP Operations Agent をインストールします。HP Operations Agent のインストールの詳細は、『HP Operations Agent and HP Operations Smart Plug-ins for Infrastructure インストール ガイド』を参照してください。

インストール後のエージェント ユーザーの設定

HP Operations Agent のインストール時に、デフォルトではないユーザーを使用するように設定できない場合、インストール後に `oainstall` スクリプト (エージェント ノード上で利用可能) の `-configure` オプションまたは `ovswitchuser` コマンドを使用して、この設定を完了できます。

Windows でのデフォルト ユーザーの変更

インストール時に、デフォルト以外のユーザーでエージェントを実行するように設定できない場合は (「インストール中のエージェント ユーザーの設定」(50ページ) を参照)、HP Operations Agent を非アクティブモードでインストールすることをお勧めします。詳細については、『HP Operations Agent インストール ガイド』を参照してください。

デフォルト以外のユーザーを使用してエージェントを設定するには、次の方法を使用できます。

- プロファイル ファイルの使用
- ovswitchuser コマンドの使用

もう一方の方法: ovswitchuser コマンドの使用

注: `ovswitchuser.vbs` コマンドを使用する前に、すべての Operations Agent プロセスが停止していることを確認してください。

プロファイル ファイルを使用しない場合は、以下の手順に従います。

注: 設定手順としては、プロファイル ファイルでの設定の方をお勧めします。

1. エージェントを停止します。

   Windows 64 ビット ノードの場合:

   ```
   %ovinstalldir%bin\win64\opcagt -kill
   ```

   その他の Windows ノードの場合:

   ```
   %ovinstalldir%bin\opcagt -kill
   ```

2. 次のコマンドを実行して、デフォルト以外のユーザーで実行されるようにエージェントを設定します。

   ```
   cscript "%ovinstalldir%bin\ovswitchuser.vbs" -existinguser <DOMAIN\USER> -existinggroup <GROUP> -passwd <PASSWORD>
   ```

   ここで、

   <DOMAIN\USER> はドメインとユーザー名です。

   <GROUP> はユーザーが属するグループの名前です (AgentGroup など)。

   <PASSWORD> はユーザーのパスワードです。

   注: 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。

3. 次のコマンドを実行して、必要なアクセス権限をデフォルト以外のユーザーに設定します。

   ```
   cscript %ovinstalldir%lbin\xpl\ovsetscmpermissions.vbs -user<User_Name> -f
   ```

ここで、<User_Name> はデフォルト以外のユーザーの名前です。

4. 次のいずれかのコマンドを実行します。

   ローカル システム以外のアカウントでエージェントのすべてのコンポーネントを実行するには

   Windows 64 ビット ノードの場合:

   ```
   %OvInstallDir%bin\win64\ovconfchg -ns eaagt -set MODE NPU
   ```

   それ以外のすべての Windows ノードの場合:

   ```
   %OvInstallDir%bin\ovconfchg -ns eaagt -set MODE NPU
   ```

   ローカル システム以外のアカウントで 運用監視コンポーネント のみを実行するには

   Windows 64 ビット ノードの場合:

   ```
   %OvInstallDir%bin\win64\ovconfchg -ns eaagt -set MODE MIXED
   ```

   それ以外のすべての Windows ノードの場合:

   ```
   %OvInstallDir%bin\ovconfchg -ns eaagt -set MODE MIXED
   ```

5. 以下のコマンドを実行します。

   Windows 64 ビット ノードの場合:

   ```
   %OvInstallDir%bin\win64\ovconfchg -ns ctrl.sudo -set OV_SUDO_USER<User_Name>
   %OvInstallDir%bin\win64\ovconfchg -ns ctrl.sudo -set OV_SUDO_GROUP<Group_Name>
   %OvInstallDir%bin\win64\ovconfchg -ns eaagt -set OPC_PROC_ALWAYS_INTERACTIVE
   NEVER
   ```

   それ以外のすべての Windows ノードの場合:

   ```
   %OvInstallDir%bin\ovconfchg -ns ctrl.sudo -set OV_SUDO_USER<User_Name>
   %OvInstallDir%bin\ovconfchg -ns ctrl.sudo -set OV_SUDO_GROUP<Group_Name>
   %OvInstallDir%bin\ovconfchg -ns eaagt -set OPC_PROC_ALWAYS_INTERACTIVE NEVER
   ```

   <User_Name> はデフォルト以外のユーザーの名前、<Group_Name> はデフォルト以外のユーザーが属するグループです。

6. 権限のない操作モードを選択した場合、次のコマンドを実行します。

   Windows 64 ビット ノードの場合:

   ```
   %OvInstallDir%bin\win64\ovconfchg -ns eaagt -set OPC_RPC_ONLY TRUE
   ```

   それ以外のすべての Windows ノードの場合:

   ```
   %OvInstallDir%bin\ovconfchg -ns eaagt -set OPC_RPC_ONLY TRUE
   ```

7. 混合操作モードを選択した場合、次のコマンドを実行します。

   Windows 64 ビット ノードの場合:

   ```
   %OvInstallDir%bin\win64\ovconfchg -ns eaagt -set NPU_TASK_SET EVENT_ACTION
   ```

   それ以外のすべての Windows ノードの場合:

   ```
   %OvInstallDir%bin\ovconfchg -ns eaagt -set NPU_TASK_SET EVENT_ACTION
   ```

8. 次のコマンドを実行します。

> **注:** この手順は UNIX/Linux ノードに必要であり、Windows ノードでは必須ではありません。ただし、Windows ノードでもこの手順を完了することをお勧めします。

Windows 64 ビット ノードの場合:

```
%OvInstallDir%bin\win64\ovconfchg -ns eaagt -set SNMP_TRAP_PORT <SNMP_Port>
%OvInstallDir%bin\win64\ovconfchg -ns bbc.cb -set SERVER_PORT<Comm_Port>
```

それ以外のすべての Windows ノードの場合:

```
%OvInstallDir%bin\ovconfchg -ns eaagt -set SNMP_TRAP_PORT <SNMP_Port>
%OvInstallDir%bin\ovconfchg -ns bbc.cb -set SERVER_PORT<Comm_Port>
```

上記の説明

<Comm_Port> は選択した通信ポート番号です。SERVER_PORT を 383 に設定した場合、これ以上の設定は必要ありません。ポートの値を 383 に設定しない場合、HPOM 管理サーバーのポートを設定します。

<SNMP_Port> は HP Operations Agent で SNMP トラップを受け取るポートです。

これらのポートは 1024 より大きい番号である必要があります。

9. エージェントを再起動します。

   Windows 64 ビット ノードの場合:

   ```
   %ovinstalldir%bin\win64\opcagt -start
   ```

   それ以外のすべての Windows ノードの場合:

   ```
   %ovinstalldir%bin\opcagt -start
   ```

UNIX/Linux でのデフォルト ユーザーの変更

インストール時に、デフォルト以外のユーザーでエージェントを実行するように設定できない場合は (「インストール中のエージェント ユーザーの設定」(50ページ) を参照)、HP Operations Agent を非アクティブモードでインストールすることをお勧めします。詳細については、『HP Operations Agent インストール ガイド』を参照してください。

デフォルト以外のユーザーを使用してエージェントを設定するには、次の方法を使用できます。

- プロファイル ファイルの使用
- ovswitchuser コマンドの使用

プロファイル ファイルの使用

プロファイル ファイルを使ってデフォルト エージェント ユーザーを変更するには、次の手順を実行します。

1. エージェントをインストールするシステムで新しいファイルを作成して、そのファイルをテキスト エディタで開きます。
2. 次の文を指定して、エージェントの操作モードを指定します。

ローカル システム以外のアカウントでエージェントを実行するには、次のように入力します。

set eaagt:MODE=NPU

ローカル システム以外のアカウントで 運用監視コンポーネント のみを実行するには、次のように入力します。

set eaagt:MODE=MIXED

3. 権限のないモードを選択した場合 (前の手順で `set eaagt:MODE=NPU` と入力した場合)、次の文を入力します。

   set eaagt:OPC_RPC_ONLY=TRUE

4. 権限のないモードを選択した場合 (前の手順で `set eaagt:MODE=NPU` と入力した場合)、次の文を入力します。

   set eaagt:SNMP_TRAP_PORT=<SNMP_port_number>

   set bbc.cb:SERVER_PORT=<Comm_port_number>

   > **注:** これは UNIX/Linux ノードには必須です。UNIX/Linux では root 以外のユーザーには 1024 未満のポートにアクセスできる権限がないためです。

   上記形式の説明

   <Comm_Port> は選択した通信ポート番号です。

   <SNMP_Port> は HP Operations Agent で SNMP トラップを受け取るポートです。

   これらのポートは 1024 より大きい番号である必要があります。

5. 混合モード (手順 2 で `set eaagt:MODE=MIXED` と入力した場合)、次の文を入力します。

   set eaagt:NPU_TASK_SET=EVENT_ACTION

6. 次の文を入力します。

   set ctrl.sudo:OV_SUDO_USER=<User_Name>

   set ctrl.sudo:OV_SUDO_GROUP=<Group_Name>

   <User_Name> はデフォルト以外のユーザーの名前、<Group_Name> はデフォルト以外のユーザーが属するグループです。

7. システムのローカル ディレクトリにファイルを保存します。
8. プロファイル ファイルで指定したユーザーで実行されるようにエージェントを再設定します。
   a. ノードの以下の場所に移動します。

      `/opt/OV/bin/OpC/install`

   b. 次のコマンドを実行します。

      `./oainstall.sh -a -configure -agent_profile`<path>/<profile_file>

      ここで、<profile_file> はプロファイル ファイルの名前であり、<path> はプロファイル ファイルへの完全パスです。

もう一方の方法: ovswitchuser コマンドの使用

注: ovswitchuser.sh コマンドを使用する前に、すべての Operations Agent プロセスが停止していることを確認してください。

プロファイル ファイルを使用しない場合は、以下の手順に従います。

注: 設定手順としては、プロファイル ファイルでの設定の方をお勧めします。

1. 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`

2. 以下のコマンドを実行して、エージェントを停止します。

   `./opcagt -kill`

3. 次のコマンドを実行して、デフォルト以外のユーザーで実行されるようにエージェントを設定します。

   `./ovswitchuser.sh -existinguser` <User_Name> `-existinggroup` <Group_Name>

   ここで、

   <User_Name> はエージェントを実行するユーザーの名前です。

   <Group_Name> はユーザーが属するグループの名前です (AgentGroup など)。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。

   注: 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. 次のいずれかのコマンドを実行します。

   root 以外のユーザーでエージェントのすべてのコンポーネントを実行するには

   `./ovconfchg -ns eaagt -set MODE NPU`

   root 以外のユーザーで 運用監視コンポーネント のみを実行するには

   `./ovconfchg -ns eaagt -set MODE MIXED`

5. 次のコマンドを実行します。

   注: これは UNIX/Linux ノードには必須です。UNIX/Linux では root 以外のユーザーには 1024 未満のポートにアクセスできる権限がないためです。

   ```
   ./ovconfchg -ns eaagt -set SNMP_TRAP_PORT <SNMP_Port>
   ./ovconfchg -ns bbc.cb -set SERVER_PORT <Comm_Port>
   ```

   上記形式の説明

   <Comm_Port> は選択した通信ポート番号です。SERVER_PORT を 383 に設定した場合、これ以上の設定は必要ありません。ポートの値を 383 に設定しない場合、HPOM 管理サーバーのポートを設定します。

   <SNMP_Port> は HP Operations Agent で SNMP トラップを受け取るポートです。

   これらのポートは 1024 より大きい番号である必要があります。

6. 以下のコマンドを実行して、エージェントを起動します。

   ```
   ./opcagt -start
   ```

エージェントが root 以外のユーザーで実行されるように設定すると、**System.txt** ファイルに以下のエラー メッセージが表示されることがあります。

```
ovbbccb (22461/1):(bbc-188) 現行プロセスのルート ディレクトリを変更できません。
```

このエラーは無視してください。

コマンドのデフォルト ユーザーの変更

デフォルトでは、エージェントは、現在エージェント自体を実行しているユーザー 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、別のユーザー アカウントでコマンドを開始するように HP Operations Agent を設定できます。そのためには、ノードの eaagt 名前空間の OVO_STD_USER 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。

- HP Operations Agent のインストール時にデフォルトの値を設定します。この方法は、多数のノードでユーザー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 を使用します。
- <ユーザー/|<暗号化されたパスワード> の形式で OVO_STD_USER の値を指定します。

  <ユーザー> をユーザー名に置き換えます。ドメイン ユーザーの場合、ドメイン名とユーザー名を指定します (EXAMPLE\AgentUser など)。ローカル ユーザーの場合、名前のみを指定します (AgentUser など)。

  <暗号化されたパスワード> を **opcpwcrpt** <パスワード> の出力結果に置き換えます。このコマンドは、管理サーバーのコマンド プロンプトから実行できます。

  Windows の場合:

  $OVO_STD_USER として実行します。

HP-UX/Linux/Solaris の場合:

$OVO_STD_USER の後に action が続きます。

> **注:** OVO_STD_USER 変数は、現在 Windows プラットフォームでのみ利用可能です。

ツールを設定または起動する場合は、OVO_STD_USER を使用することもできます。ユーザー名を $OVO_STD_USER とし、パスワードは空白のままとします。

コマンドやツールを正しく実行するための適切な権限がユーザー アカウントに割り当てられているかどうかをテストする必要があります。

> **注:** OVO_STD_USER でエージェントがコマンドまたはツールの開始に失敗する場合は、エージェントを現在実行しているユーザー アカウントでコマンドまたはツールが実行されている可能性があります。誤ったユーザー名またはパスワードを指定した場合などに、この現象が発生します。

セキュリティ コンポーネント変数の設定

RSA 証明書と非対称鍵暗号化方式は、ノード間、およびノードと管理サーバー間で SSL ハンドシェイク中の安全な通信を確立するために使用されます。HP Operations Agent をインストールすると、証明書管理コンポーネント OvSecCm は、設定した非対称キーの長さの値に基づいて、安全な通信を確立するために使用する RSA キー ペアを生成します。キーの長さを長くすると、暗号化を強化できます。

設定可能な非暗号キーの長さ機能は、RSA キーの長さを必要な値に設定するのに有用です。デフォルトでは、この値は 2048 に設定されており、OvSecCm は通信用に 2048 ビットの RSA キー ペアを作成します。CA 証明書はエイリアス <CA_ovcoreid_keylength> を使用して作成されます。たとえば、CA_8cd78962-ab51-755c-1279-85f5ba286e97_2048 となります。ここで、2048 はデフォルトのキーの長さです。RSA キーの長さを変更するには、設定変数 **ASYMMETRIC_KEY_LENGTH** を使用します。

HP Operations Agent をインストールすると、証明書クライアント コンポーネント OvSecCc は設定変数 ASYMMETRIC_KEY_LENGTH を 2048 に設定します。

RSA キーの長さを変更するには、変数 ASYMMETRIC_KEY_LENGTH を更新する必要があります。

管理サーバーで設定の変更を適用するには、MigrateAsymKey ツールを使用できます。このツールは、新規の CA キー ペアを作成し、新しく生成されたキー ペアの新規の CA 証明書を追加し、管理サーバー上のローカル エージェントとその他のすべての OV リソース グループ (ovrg) に対する信頼された証明書を更新し、ローカル エージェントとすべての OV リソース グループに対する新規の証明書を作成します。

管理サーバーの場合

管理サーバーで設定変数を更新するには、以下の手順を実行します。

1. 次のコマンドを使用して、設定変数 ASYMMETRIC_KEY_LENGTH を更新します。

   `ovconfchg –ns sec.cm -set ASYMMETRIC_KEY_LENGTH` <RSA Encryption algorithm supported key length>

2. 管理サーバーで以下の場所に移動します。

   Windows の場合:

   `%ovinstalldir%\lbin\seccs\install\`

   HP-UX/Linux/Solaris の場合:

   `%ovinstalldir%/lbin/seccs/install/`

3. 移行ツールを実行します。

   Windows の場合:

   ```
   cscript MigrateAsymKey.vbs -createCAcert
   ```

   HP-UX/Linux/Solaris の場合:

   ```
   ./MigrateAsymKey.sh -createCAcert
   ```

   ASYMMETRIC_KEY_LENGTH の設定変数に設定された新しい鍵長の値に応じて、コマンドにより新しい CA 証明書が作成されます。

   > **注:** 移行ツールをインストールすると、証明書サーバー コンポーネントのアップグレード中に、-createCAcert オプションを使用して移行ツールが実行されます。-createCAcert は、変数 ASYMMETRIC_KEY_LENGTH に設定した新しい設定値に対応する CA 証明書を作成し、サーバーおよびすべての ovrg 上のエージェントに対して作成された CA 証明書を更新するのに使用します。したがって、ASYMMETRIC_KEY_LENGTH 設定変数に対して設定変更を行わない場合は、-createCAcert オプションを使用した移行ツールを実行しないでください。

4. サーバーで管理されているすべてのノードで信頼のおける証明書を更新するには、次のコマンドを実行します。

   ```
   ovcert –updatetrusted
   ```

   > **注:** -createNodecert オプションを使用して MigrateAsymKey ツールを実行する前に、すべての管理ノード上で ovcert – updatetrusted コマンドを実行する必要があります。

5. 管理サーバーおよびその管理サーバー上のすべての ovrg 上のエージェントに対して新規のノード証明書を作成するには、以下のコマンドを実行します。

   Windows の場合:

   ```
   cscript MigrateAsymKey.vbs -createNodecert
   ```

   HP-UX/Linux/Solaris の場合:

   ```
   ./MigrateAsymKey.sh -createNodecert
   ```

   このコマンドは、新規の RSA キー ペアを使用して、管理サーバーおよびすべての ovrg 上の Operations Agent に対して新規のノード証明書を作成します。

> **注:** コマンド `ovcert – certinfo` <certificate_alias> を実行すると、更新されたキーの長さを確認できます。キーの長さは ASYMMETRIC_KEY_LENGTH で更新されます。

エージェントの場合

エージェント (管理ノード) 上の設定変更を適用するには、以下の手順を実行します。

1. 次のコマンドを使用して、設定変数 ASYMMETRIC_KEY_LENGTH を更新します。

   ```
   ovconfchg –ns sec.cm -set ASYMMETRIC_KEY_LENGTH <RSA Encryption algorithm supported key length>
   ```

2. エージェント上の既存のノード証明書を削除するには、以下のコマンドを実行します。

   ```
   ovcert –remove<certificate alias>
   ovcert –remove <CA certificate alias>
   ```

3. 管理サーバーから新しいノード証明書を要求するには、以下のコマンドを実行します。

   ```
   ovcert -certreq
   ```

対象キーのセキュリティ コンポーネントの設定

HP Operations Agent をインストールした場合、セキュリティ コンポーネント OvSecore と OvJSecCore は、暗号化および暗号化解除用のより古いデフォルトのアルゴリズム値を継続して使用します。バージョン 11.10 以降から、Windows 用の HPSharedComponent パッケージには OvSecCore と OvJSecCore の両方が含まれています。

サポートされている対称鍵アルゴリズムは次のとおりです。

- Blowfish
- DES
- DES3
- AES128
- AES192
- AES256

以下の 2 つの設定変数があります。

- **DEF_SYM_KEY_ALGO**: この変数は、暗号化用のデフォルトの対称キー アルゴリズムを設定するのに使用します。サポートされているアルゴリズムは次のとおりです。
  - eBlowfish
  - eDES
  - eDES3
  - eAES128
  - eAES192
  - eAES256
  - eDefault: デフォルト アルゴリズムとして AES128 を使用。

- **ENABLE_DEF_SYM_KEY_ALGO**: この変数は、DEF_SYM_KEY_ALGO に設定したデフォルトの対称キーアルゴリズムの使用を有効にするのに使用します。サポートされている値は TRUE であり、それ以外の値の場合、値が設定されていないと見なされます。

設定変数を使用して、暗号化の設定で指定したアルゴリズムの使用を有効にすることができます。MigrateSymKey ツールを使用して、値を TRUE に設定できます。

管理サーバーおよび HP Operations Agent を使用して実行しているノード上で設定を更新するには、MigrateSymKey ツールを使用できます。MigrateSymKey ツールは ENABLE_DEF_SYM_KEY_ALGO 設定変数を TRUE に設定し、設定変数 DEF_SYM_KEY_ALGO に設定したアルゴリズムに基づいて KeyStore コンテンツを移行します。

管理サーバーで設定を更新するには、以下の手順を実行します。

注: 以下のコマンドを実行して、設定変数 DEF_SYM_KEY_ALGO を、サポートされている以下のいずれかのアルゴリズムに設定します。

```
ovconfchg -ns sec.core -set DEF_SYM_KEY_ALGO <Supported Algorithm>
```

この変数を設定しない場合、デフォルト アルゴリズムとして eAES128 が使用されます。

1. 管理サーバーまたはエージェントをインストールしたノード上で、以下の場所に移動します。

   Windows の場合:

   ```
   %ovinstalldir%\bin\secco\
   ```

   HP-UX/Linux/Solaris の場合:

   ```
   %ovinstalldir%/lbin/secco/
   ```

2. 移行ツールを実行します。

   Windows の場合:

   ```
   MigrateSymKey -sym_key_algo [eBlowfish | eDES | eDES3 | eAES128 | eAES192 | eAES256]
   ```

   HP-UX/Linux/Solaris の場合:

   ```
   ./MigrateSymKey.sh -sym_key_algo [eBlowfish | eDES | eDES3 | eAES128 | eAES192 | eAES256]
   ```

   移行ツールを実行すると、DEF_SYM_KEY_ALGO に設定した新しい値に基づいて KeyStore コンテンツが移行されます。

使用方法:

```
MigrateSymKey

-sym_key_algo [eBlowfish | eDES | eDES3 | eAES128 | eAES192 | eAES256]

-help
```

- -sym_key_algo
  - 設定しない場合は、"ENABLE_DEF_SYM_KEY_ALGO" を TRUE に更新します。

- 対称キー アルゴリズムを指定した場合、DEF_SYM_KEY_ALGO は指定した値で更新されます。
- アルゴリズムを指定しない場合、対称キー アルゴリズムとして eAES128 が使用されます。

ハッシュ アルゴリズムを使ったセキュリティ コンポーネントの設定

HP Operations Agent 12.00 は、ハッシング用の設定可能なハッシュ アルゴリズムをサポートしています。**HASH_ALGO** はハッシュ アルゴリズムを設定するための設定変数です。この設定変数は **sec.core** 名前空間で使用できます。

HASH_ALGO は以下の設定値をサポートしています。

- eMD5
- eSHA1
- eSHA224
- eSHA256
- eSHA384
- eSHA512
- eDefault

以下のコマンドを実行して、設定変数 HASH_ALGO を、サポートされているアルゴリズムのいずれかに設定します。

`ovconfchg -ns sec.core -set HASH_ALGO` <Supported Algorithm>

HASH_ALGO 変数を eDefault (またはその他のサポートされている値) に設定した場合、sec.core は HASH_ALGO 値として eSHA256 を使用します。

Windows イベント ログの監視

HP Operations Agent の Logfile Encapsulator コンポーネントは、Windows イベント ログのイベント転送機能をサポートします。つまり、異なるマシンから転送されるイベントを読み込むことができ、それを使用して Windows イベント ログを監視することもできます。Windows イベント ログ ポリシーは、選択した Windows イベント ログを監視するようにエージェントを設定するために役立ちます。

次の Windows のバージョンには、イベント ログ—アプリケーションおよびサービス ログの新しいカテゴリが用意されています。

- Windows Vista
- Windows Server 2008
- Windows Server 2008 R2
- Windows 7

適切な Windows イベント ログ ポリシーを設定すると、HP Operations Agent 12.00 でこれらのアプリケーションおよびサービス ログを監視できます。

HP Operations Agentは次のタイプのイベント ログを監視できません。

- リモート システムで生成されたイベント (Windows のイベント サブスクリプション機能を使用して収集)
- 保存されたイベント ログ

HP Operations Agentは、次のイベント レベルを持つイベントを監視できます。

- エラー
- 情報
- 警告
- LOG_ALWAYS
- VERBOSE
- 監査失敗
- 監査成功

次の表に、イベントログ フィールドがメッセージ ブラウザでどのように表示されるかを示します。

表 1 イベントログとメッセージ ブラウザ フィールドの相関関係

イベントログ フィールド	メッセージ ブラウザ フィールド	コメント
日付	日付	イベントが管理ノード上で作成された日付。
時間	時間	イベントが管理ノード上で作成された時刻。
イベント ID	メッセージ テキスト	イベント ID は、追加のどのメッセージ テキストよりも前に表示されます。
ソース	アプリケーション	なし
型 エラー エラー 情報 警告 log_always verbose 監査失敗	重要度 危険域 エラー 正常域 注意域 正常域 正常域 注意域	イベント ログ タイプの重要度の Operations Manager のメッセージの重要度へのマッピング。

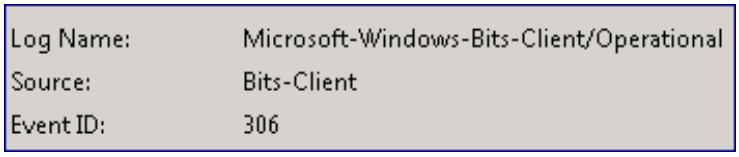
表 1 イベントログとメッセージ ブラウザ フィールドの相関関係 (続き)

イベントログ フィールド	メッセージ ブラウザ フィールド	コメント
監査成功	正常域	
カテゴリ	オブジェクト	なし
説明	メッセージ テキスト	他のすべてのメッセージ テキスト (イベント ID の後)。
ユーザー	マッピングなし	マッピングなし
コンピュータ	ノード	管理サーバーに認識されているノードの名前。
	メッセージ グループ	空

HPOM for Windows からのアプリケーション ログおよびサービス イベント ログの監視

アプリケーションおよびサービス イベント ログを監視するために Windows のイベント ログ ポリシーを作成するには、以下の手順に従います。

1. Windows イベント ログがある Windows ノードにログオンします。
2. Event Viewer ウィンドウを開きます。
3. コンソール ツリーで、イベントを選択します。詳細ペインにイベント ログの名前が表示されます ([Log Name] フィールドの横)。

   ログ ファイルの名前を詳細ペインに表示されているとおりにメモに残します。
4. HPOM for Windows コンソールを開きます。
5. コンソール ツリーの [タイプ別にグループ化されたエージェント ポリシー] で、**[Windows イベント ログ]** を右クリックして、**[新規] > [ポリシー]** をクリックします。

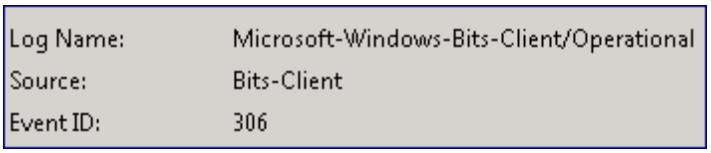
   Windows イベント ログ ポリシー用のポリシー エディタが開きます。
6. [ソース] タブで Windows イベント ログの名前 (手順 3 で記録した名前) を [Event log name] フィールドに入力します。

7. HPOM for Windows オンライン ヘルプの手順に従って、他の詳細をポリシーに指定します。
8. ポリシーを保存します。
9. Windows ノードにポリシーを配布します。

UNIX/Linux 9.xx 上の HPOM からのアプリケーション ログおよびサービス イベント ログの監視

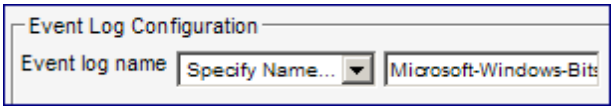
アプリケーションおよびサービス イベント ログを監視するために Windows のイベント ログ ポリシーを作成するには、以下の手順に従います。

1. Windows イベント ログがある Windows ノードにログオンします。
2. Event Viewer ウィンドウを開きます。
3. コンソール ツリーで、イベントを選択します。詳細ペインにイベント ログの名前が表示されます ([Log Name] フィールドの横)。



   ログ ファイルの名前を詳細ペインに表示されているとおりにメモに残します。
4. HPOM for UNIX 管理 UI にログオンします。
5. **[OMU]** をクリックします。
6. **[参照] > [すべてのポリシー タイプ]** をクリックします。
7. [Windows イベント ログ] をクリックします。「**Windows_Event_Log**」というポリシー タイプのページが開きます。
8. をクリックして、**[新規ポリシー]** をクリックします。「Windows_Event_Log ポリシーの追加」というページが開きます。

   [ソース] タブの [Event log name] フィールドで、[Specify Name] を選択します。新しいテキスト ボックスが表示されます。Windows イベント ログの名前 (手順 3 で記録した名前) をテキスト ボックスに入力します。



9. HPOM for UNIX オンライン ヘルプの手順に従って、他の詳細をポリシーに指定します。

10. ポリシーを保存します。
11. Windows ノードにポリシーを配布します。

第2章: 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。

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。

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ト 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 Agent 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 の場合:

```
%ovdatadir%\perf\perfd
```

HP-UX/Linux/Solaris の場合:

```
/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 の場合:

%ovdatadir%\perf\perfd\os\<os_type>\adv

HP-UX/Linux/Solaris の場合:

/var/opt/perf/perfd/os/<os_type>/adv

この例では、<os_type> にはスクリプトを実行するノードのオペレーティング システムを指定します。

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、以下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 の場合:

   %ovinstalldir%\examples\adviser

   HP-UX/Linux/Solaris の場合:

   /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。

   > **ヒント:** リモート システムでスクリプトを使用するときは、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。他のシステムのローカルで adviser スクリプトを実行しないようにできます。詳細については、「アクセスの制限」を参照してください。

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、以下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 の場合:

   %ovinstalldir%\examples\adviser

   HP-UX/Linux/Solaris の場合:

   /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 の場合:

%ovdatadir%\perf

HP-UX/Linux/Solaris の場合:

/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カンマで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「等しい」を、「!=」は「等しくない」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を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
- かっこ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かか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 秒あたりの単位を計測
- `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わからない場合は、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 = ""
```

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。

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、YELLOW は CRITICAL、WARNING と同義です。

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
```

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。

COMPOUND 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。

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。

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。

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 文
```

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 文は LOOP 文を終了します。

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。

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。

メモ: 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、この兆候の名前を表します。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。

第3章: 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、**oacore** または 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、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 **oacore** または他のコレクタにより記録されると、そのデータは `alarmdef` ファイル内のアラーム定義と比較されす。**oacore** または DSI メトリックが定義された条件を満たす場合や超える場合、アラートまたはアクションがトリガされます。

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
- アラート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。

DSI メトリックのアラームの定義の詳細は、「メトリック データストアに記録される DSI データの使用」の「DSI メトリックのアラーム定義」を参照してください。

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。

ただし、データがデータベース 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。詳細については、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。

- オペレーティング システム 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 (NNM) または HPOM に送信されるメッセージ

アラーム ジェネレータ

Performance Collection Component のアラーム ジェネレータ コンポーネントは、`alarmdef` ファイルとローカル システムの利用可能なシステム パフォーマンス データを処理し、必要に応じてアラーム

を生成します。アラーム ジェネレータは以下のコンポーネントで構成されます。

- 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)

アラーム ジェネレータは alarmdef ファイル内の情報を検索し、alarmdef ファイルの設定情報に基づいてアラートを宛先に送信します。agdb データベースには、perfalarm コンポーネントが特定のイベントに対する SNMP トラップを転送するためのターゲット システムのリストが含まれています。perfalarm コンポーネントのデフォルトの動作を変更し、agsysdb ユーティリティを使用して agdb データベースの利用可能なデータにアクセスすることができます。

アラート通知の送信先のターゲット システムのリストを表示するには、次のコマンド ラインを実行します。

```
agsysdb -l
```

perfalarm の有効化

デフォルトでは、アラーム ジェネレータは無効になっています。perfalarm を有効にするには、次のいずれかの方法を使用します。

- **HP Operations Agent をインストールする前:**
  a. 次のように、プロファイル ファイルで変数 ENABLE_PERFALARM を **True** に設定します。

     ```
     set nonXPL.config:ENABLE_PERFALARM=TRUE
     ```

  b. プロファイル ファイルを使用して HP Operations Agent をインストールするには、次のコマンドを実行します。

     Windows の場合:

     cscript oainstall.vbs -i -a -agent_profile <path>\<profile_file> -s <server_ip_address>

     HP-UX/Linux/Solaris の場合:

     ./oainstall.sh -i -a -agent_profile <path>/<profile_file> -s <server_ip_address>

     ここで、

     <profile_file> はプロファイル ファイルの名前です。

     <path> はプロファイル ファイルへの完全パスです。

     <server_ip_address> は、管理サーバーの IP アドレスまたはホスト名です。

- **HP Operations Agent をインストールした後:**
  a. 以下のコマンドを実行して、perfalarm を有効にします。

     Windows の場合:

     ```
     copy "%ovinstalldir%newconfig\alarmdef.mwc" "%ovdatadir%"
     "%ovinstalldir%bin\ovpacmd" start alarm
     ```

     HP-UX/Linux/Solaris の場合:

     ```
     cp/opt/perf/newconfig/alarmdef /var/opt/perf/
     /opt/perf/bin/ovpa start alarm
     ```

AIX の場合:

```
cp/usr/lpp/perf/newconfig/alarmdef /var/opt/perf/
/usr/lpp/perf/bin/ovpa start alarm
```

b. ノードの以下の場所に移動します。

Windows 64 ビットの場合:

```
%ovinstalldir%bin\win64\OpC\install
```

その他の Windows の場合:

```
%ovinstalldir%bin\OpC\install
```

HP-UX/Linux/Solaris の場合:

```
/opt/OV/bin/OpC/install
```

AIX の場合:

```
/usr/lpp/OV/bin/OpC/install
```

c. 以下のコマンドを実行して、HP Operations Agent を再設定します。

Windows の場合:

```
cscript oainstall.vbs -a -configure -agent_profile<path>/<profile_file>
```

HP-UX/Linux/Solaris の場合:

```
./oainstall.sh -a -configure -agent_profile<path>/<profile_file>
```

ここで、

<profile_file> はプロファイル ファイルの名前です。

<path> はプロファイル ファイルへの完全パスです。

注: HP Operations Agent 11.xx から 12.00 にアップグレードした場合、perfalarm は以前の設定のとおりに動作します。

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

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。

HPOM へのメッセージの送信

デフォルトでは、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 HPOM のイベント ブラウザにメッセージを送信します。

次のコマンドを使用すると、HPOM への情報送信を停止するようにデフォルトを変更できます。

```
agsysdb -ovo OFF
```

表 9: HPOM 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

HPOM フラグ	運用監視コンポーネントが動作している場合	運用監視コンポーネントが動作していない場合
off	アラート通知が HPOM に送信されません。	アラート通知が HPOM に送信されません。
on	アラート通知が HPOM に送信されます。	アラート通知が HPOM に送信されません。

ローカル アクションの実行

デフォルトでは、Performance Collection Component は EXEC 文で指定されているローカル コマンドを実行しません。

次のようにデフォルトを変更すると、ローカル アクションを有効にできます。

```
agsysdb -actions always
```

次の表は、HP Operations Manager (HPOM) 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: HPOM 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

ローカル アクション フラグ	運用監視コンポーネントが動作している場合	運用監視コンポーネントが動作していない場合
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	運用監視コンポーネ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。

表 10: HPOM 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 (続き)

ローカル アクション フラグ	運用監視コンポーネントが動作している場合	運用監視コンポーネントが動作していない場合
on	ローカル アクションが HPOM 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

データストア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 (5 章「ユーティリティ コマンド」を参照してください)。データストア内の履歴データを 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。

MAJOR 文はマゼンダで表示されます。

MINOR 文は黄色で表示されます。

WARNING 文はシアン（緑がかった青）で表示されます。

NORMAL 文は緑で表示されます。

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。たとえば、次のようになります。

```
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。

- ALERT - HPOM にメッセージを送信するか、NNM に SNMP トラップを送信する
- EXEC - オペレーティング システム コマンドを実行する
- PRINT - 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

たとえば、次のようになります。

```
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、アプリケーションや変数などの複数インスタンス データによるループを使用すると、より複雑な動作を生成できます(詳細は、「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]
     .
     .
```

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カンマで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

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。たとえば、次のようになります。

```
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。たとえば、次のようになります。

```
application_1:metric_1
```

application、process、disk、netif、transaction、lvolume、cpu、filesystem 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。

注: 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、「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
```

注: 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、カンマによって区別されます。例:

```
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。たとえば、次のようになります。

```
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
- 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
- CPU が最後の 10 分間に 75% の利用率で動作している

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]
```

ここで、

- 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。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。詳細については、『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「等しい」を、「!=」は「等しくない」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。たとえば、次のようになります。

```
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」を指定せずに、複合条件を使用することもできます。たとえば、次のようになります。

```
ALARM gbl_cpu_total_util > 90
gbl_cpu_sys_mode_util > 50 for 5 minutes
```

両方の条件が **True** になるとアラームが生成されます。

FOR は、アラームをトリガするために、条件が **True** のままでなければならない、SECONDS または MINUTES の期間を指定します。

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。

収集インターバルより短い継続時間を設定しないでください。次の収集サイクルが完了するまでメトリックの値は変更されないため、収集インターバルより短い継続時間を設定すると、重複するアラームが不必要に処理されることになります。

parm ファイルで、collectioninterval パラメータにある次のパラメータでは、収集インターバルを制御します。

- global: このパラメータは、oacore および dsi データベース ファイルに記録される、プロセス メトリックを除くすべてのメトリックの収集インターバルを秒単位で示します。デフォルトでは 5 分に設定されています。
- process: このパラメータは、プロセス メトリックの収集インターバルを秒単位で示します。

  collectioninterval パラメータの詳細については、「データ記録間隔の設定」を参照してください。

- REPEAT EVERYnSECONDS, MINUTES は、アラームが繰り返される前の期間を指定します。

  この場合、**n** が期間を示します。例: REPEAT EVERY5SECONDS, MINUTES

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 statement 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 HPOM に送信し、SNMP トラップを NNM に送信します。VAR 文

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 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 NNM 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、NNM への SNMP トラップの送信または HPOM 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、ノード上のローカル 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。

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 > 85
  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 > 85
  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 gbl_cpu_total_util > 85 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。

> **注:** 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 または 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、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 NNM への SNMP トラップの送信や、HPOM へのメッセージの送信をトリガできます。HPOM 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 "system command"
```

system command - 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。たとえば、次のようになります。

```
EXEC "echo 'performance problem detected' "
```

```
EXEC "mkdir c:\\directory\\filename"
```

EXEC 文の構文には、二重引用符で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る場合、パス名を一重引用符で囲んでから、二重引用符でもう一度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 コマンドの引数に一重引用符が含まれている場合、EXEC 文でコマンドを実行するときに最初の一重引用符 (') のペアは二重引用符 (") に変換されるため、プログラムを一重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 文の構文により、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。perfalarm で HPOM へのアラームの送信が失敗します。代わりに一重引用符 (') を使用してください。

たとえば、次のようになります。

```
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 文の構文により、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。HPOM 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"
```

注: 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。

詳細については、「PRINT 文」(78ページ)を参照してください。

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]
```

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「等しい」を、「!=」は「等しくない」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 NNM 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 gbl_cpu_total_util > 85 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、集合変数です。

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

alarmdef ファイルのアラーム定義に次の行を追加すると、alarmdef 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。

> **注:** 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「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

構文では特殊文字や大文字と小文字を使用できないため、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。たとえば、次のようになります。

```
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 **True** である場合、その確率値が SYMPTOM の変数値に追加されます。

詳細については、「SYMPTOM 文」(79ページ)を参照してください。

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、重要度が危険域のアラームが NNM に作成されます。次にプログラムが実行され、システム管理者をページングします。

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 NNM への SNMP トラップの送信や、HPOM 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

   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。

第4章: 安全な環境における HP Operations Agent

HPOM は HP Operations Agentと共に、ネットワーク環境に配備されたシステムとアプリケーションを、中央コンソールから監視、管理できるようにします。HPOM ベースの管理環境では、HP Operations Agentを対象システムにインストールした後、監視を開始することができます。HPOM コンソールからエージェント ノードに配布されたポリシーを使用すれば、エージェントのさまざまな監視機能を有効化できます。

分散環境における HP Operations Agent の主な担当範囲は次のとおりです。

- **データの監視**
  HP Operations Agent は特定のメトリック値を事前設定値と比較し、設定に基づいて必要なアクションを実施することができます。HPOM コンソールからノードに配布されるポリシーは、HP Operations Agentの監視機能を容易に実現する上で主要な役割を果たします。
- **データの収集と格納**
  HP Operations Agent のパフォーマンス データ コレクタをプログラミングし、監視システムの対象データを収集、記録することができます。SPI をインストールすることにより収集機能をさらに追加し、SPI が収集したデータをエージェントのデータストアに記録することができます。

ポリシー

エージェントを使用するには、ポリシーと呼ばれる設定の詳細と仕様の集合体を HPOM コンソールから管理対象ノードに配布する必要があります。配布されたポリシーのタイプに応じて、HP Operations Agent の異なったコンポーネントが有効になります。ポリシーは以下の詳細をエージェントに提供できます。

- **監視ソースの詳細**
  - 監視するオブジェクト
  - オブジェクトを監視するためのポーリング間隔
  - 監視オブジェクトのしきい値
  - 設定したしきい値に対するデータ分析のルールと条件
- **イベントの詳細**
  ポリシーを使用して HP Operations Agentを設定し、監視オブジェクトがしきい値ルールに違反したときにメッセージ、指示、および重要度フラグを伴ったイベントを生成することができます。イベントはメッセージ形式で、HPOM コンソールに転送されます。エージェントを設定し、これらのイベントに対して特定のアクションを実行することができます。

- **データ収集の詳細**
  外部プログラムが収集したデータを監視したい場合、HP Operations Agent をプログラミングしてデータをデータストアに記録することができます。

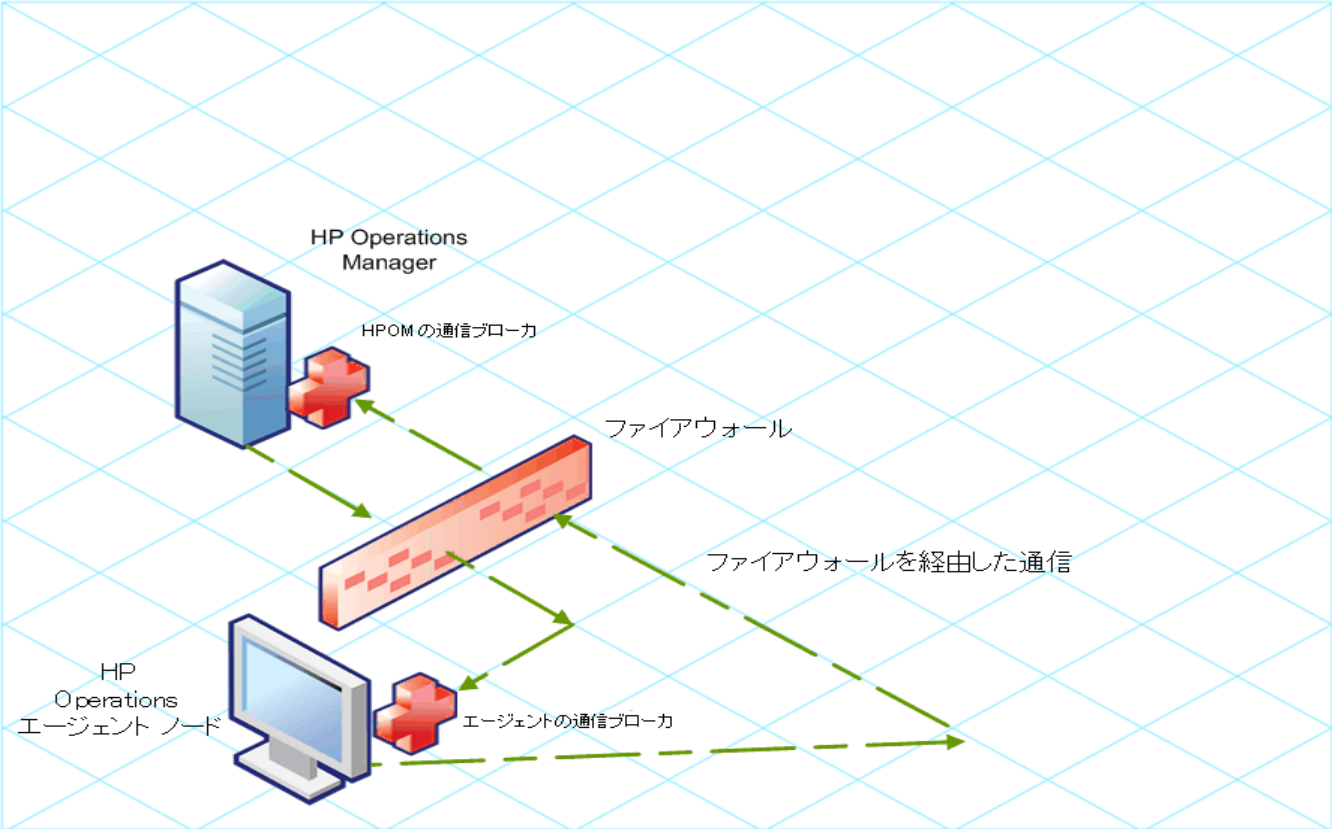
HTTPS モード通信

HP Operations Agentノードは HTTPS モード通信を使用することにより、他の業界標準の製品と同様に、互いに簡単に通信することができます。

HTTPS 通信の利点

- **ファイアウォールを介した通信**
  HTTPS プロトコルを使用すれば、HP Operations Agentノードはファイアウォールを経由して利用可能な他のシステムと通信することができます。HTTP プロキシとファイアウォールで構築された安全な環境に、HP Operations Agentを配備することができます。

  以下の図では、HTTPS 通信を使用してファイアウォールを通過する方法を説明しています。

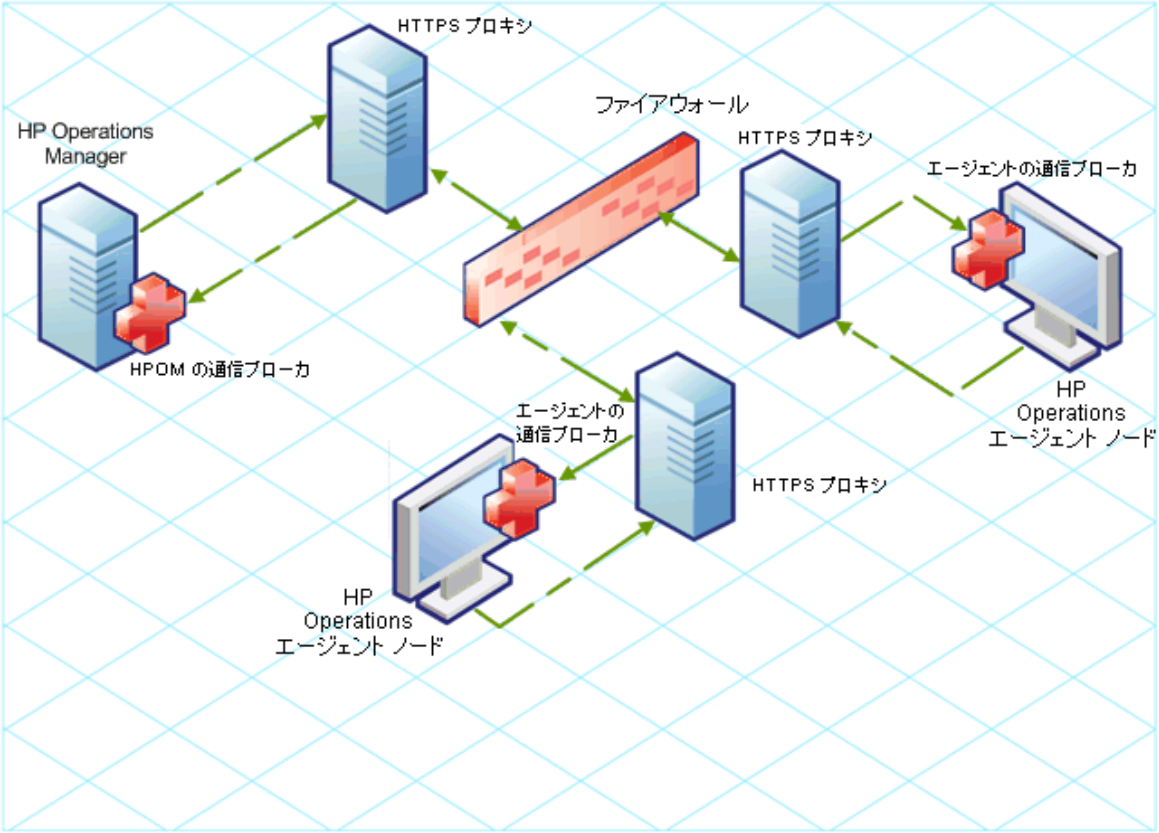


- **高度なセキュリティ**
  HP Operations Agent 製品は SSL (Secure Socket Layer) を使用し、ユーザーのアクセスを制限、制御しています。HP Operations Agent 製品は SSL を使用し、他のシステムとの通信にかかわるすべてのデータを圧縮および暗号化しています。

  さらに、すべてのリモート メッセージは 通信ブローカ コンポーネントを通じて到達し、HP Operations Agent ノードに単一のポート エントリを提供しています。

  HP Operations Agent ノードからメッセージ、ファイル、またはオブジェクトを送信したい場合は、1 つまたは複数の標準 HTTP プロキシを、ファイアウォールを通過するか、またはリモート システムに到達するように設定することができます。

  以下の図では、HTTPS プロキシを使用してファイアウォールを通過する方法を説明しています。



- **オープン スタンダード**
  HP Operations Agent の HTTPS 通信は、業界標準の HTTP 1.1 プロトコルと SSL ソケットをベースに実現されています。HP Operations Agentが HTTP、SSL、SOAP といったオープン スタンダードにこだわるのは、既存の HTTP インフラストラクチャを最大限利用できるようにするためです。
- **スケーラビリティ**
  HP Operations Agentの HTTPS 通信は、環境の規模や送受信するデータ量とは無関係に、適切に機能するように設計されています。また、HP Operations Agentの HTTPS 通信は、組織の要件に合わせて設定することができます。

通信ブローカ

通信ブローカ コンポーネントにより、HP Operations Agentノードの通信を単一ポートで制御することができます。通常の配布シナリオでは、データ通信用に複数のサーバーを HP Operations Agent ノードに登録することができます。HP Operations Agent製品は、ノードに登録されたすべてのサーバーに対するリクエストを通信ブローカを通じて転送します。通信ブローカは HTTP プロキシが HTTP リクエストを転送するのと同じ方法で、登録されたサーバーにリクエストを透過的に転送します。通信ブローカ用のデフォルトのポートは 383 です。HP Operations Agent 製品を設定すれば、通信ブローカ用に別のポートを使用することができます。

UNIX システムのセキュリティを強化するために、通信ブローカは `chroot` を使用してスタートアップを行います。`chroot` は、指定されたパスを root ディレクトリとして動作させることにより、通信ブローカのプロセスが認識できるファイル システムを一部制限します。これにより、未承認のアクセスにさらされる領域を減らすことができます。

通信ブローカは UNIX システムではデーモンとして、Windows システムではサービスとして動作します。

通信ブローカは、ノードで受信データを受け入れるためのポートとして、少なくとも 1 つのポートを使用します。このポートはノードを識別するために、一意のノード識別子 (`OVCoreID`) と関連付けられます。通信ブローカは、高可用性ノード用に複数のポートを使用するよう設定できます。

ファイアウォールのシナリオ

ファイアウォールは、ネットワーク内のシステムを外部の攻撃から保護することができます。通常は、プライベートなイントラネットとインターネットを分離する位置に設置します。ファイアウォールを複数のレベルで実装し、セキュリティへの配慮が低い環境から信頼性の高い環境へのアクセスを制限することができます。

ファイアウォールは、ネットワーク環境を、**安全地帯**と**非武装地帯 (DMZ)** (インターネットなど) の 2 つの基本的なゾーンに分離します。ファイアウォールの設定により、DMZ から安全地帯へのデータ伝送は確実に制限または制御されます。ファイアウォールの設定に応じて、**双方向通信**または**アウトバウンド通信のみ**が許可されます。

環境内のファイアウォールを双方向通信を許可するように設定した場合、ネットワークはファイアウォールをまたいだ HTTPS の双方向通信を、一定の制限付きで許可します。この環境では、以下の設定オプションを使用してファイアウォールを設定することができます。

- プロキシ: ネットワークにおいて、ファイアウォールを介した接続が特定のプロキシ システムにのみ許可されている場合、これらのプロキシを介して HP Operations Agent通信をリダイレクトさせることができます。
- ローカル ポート: ネットワークにおいて、アウトバウンド接続が特定のローカル ポートにのみ許可されている場合、特定のローカル ポートを使用するよう HP Operations Agentを設定できます。
- 通信ブローカ ポート: ネットワークにおいて、インバウンド接続が特定の宛先ポートにのみ許可され、ポート 383 には許可されていない場合、別の通信ブローカ ポートを設定できます。

環境内のファイアウォールがアウトバウンド通信のみを許可している場合は、HP Operations Agent製品に**リバース チャネル プロキシ (RCP)** を設定することができます。HP Operations Agentノードに設定された RCP は HTTP プロキシと同様に機能し、DMZ から安全地帯へのデータ伝送を可能とします。HP ソフトウェア システムと直接通信する代わりに、RCP は通信ブローカとの通信チャネルを確立します。通信ブローカは DMZ から発生した情報を検証および認証し、検査済みの情報を安全地帯に存在する HP Operations Agentノードに伝送します。

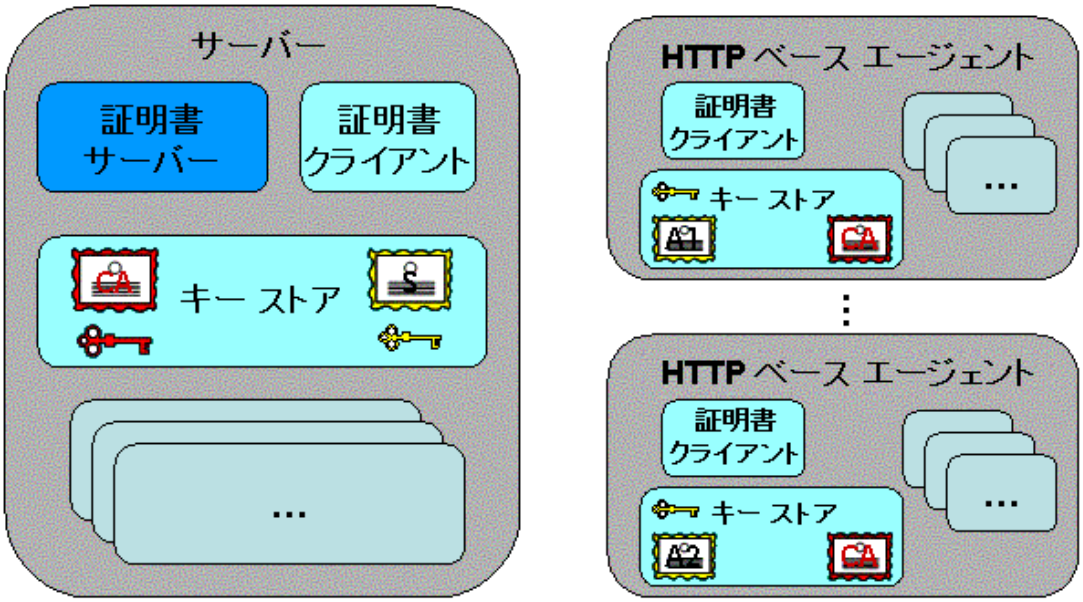
HTTPS ベースのセキュリティ コンポーネント

別の HP Operations Agentノードまたは HPOM サーバーと通信するために、HP Operations Agent ノードは有効な業界標準の X509 証明書を保有する必要があります。各ノードは 1024 ビットの鍵で署名された証明書を交換した後で、互いに通信を行います。証明書の交換は、ノードが管理対象環境の他のノードやサーバーを識別するのに役立ちます。

証明書の作成と管理を担当する主要なコンポーネントは、次のとおりです。

- 証明書サーバー (HPOM サーバー上に存在)
- HP Operations Agentキー ストア
- HP Operations Agent証明書クライアント

以下の図は、これらのコンポーネントを示しています。

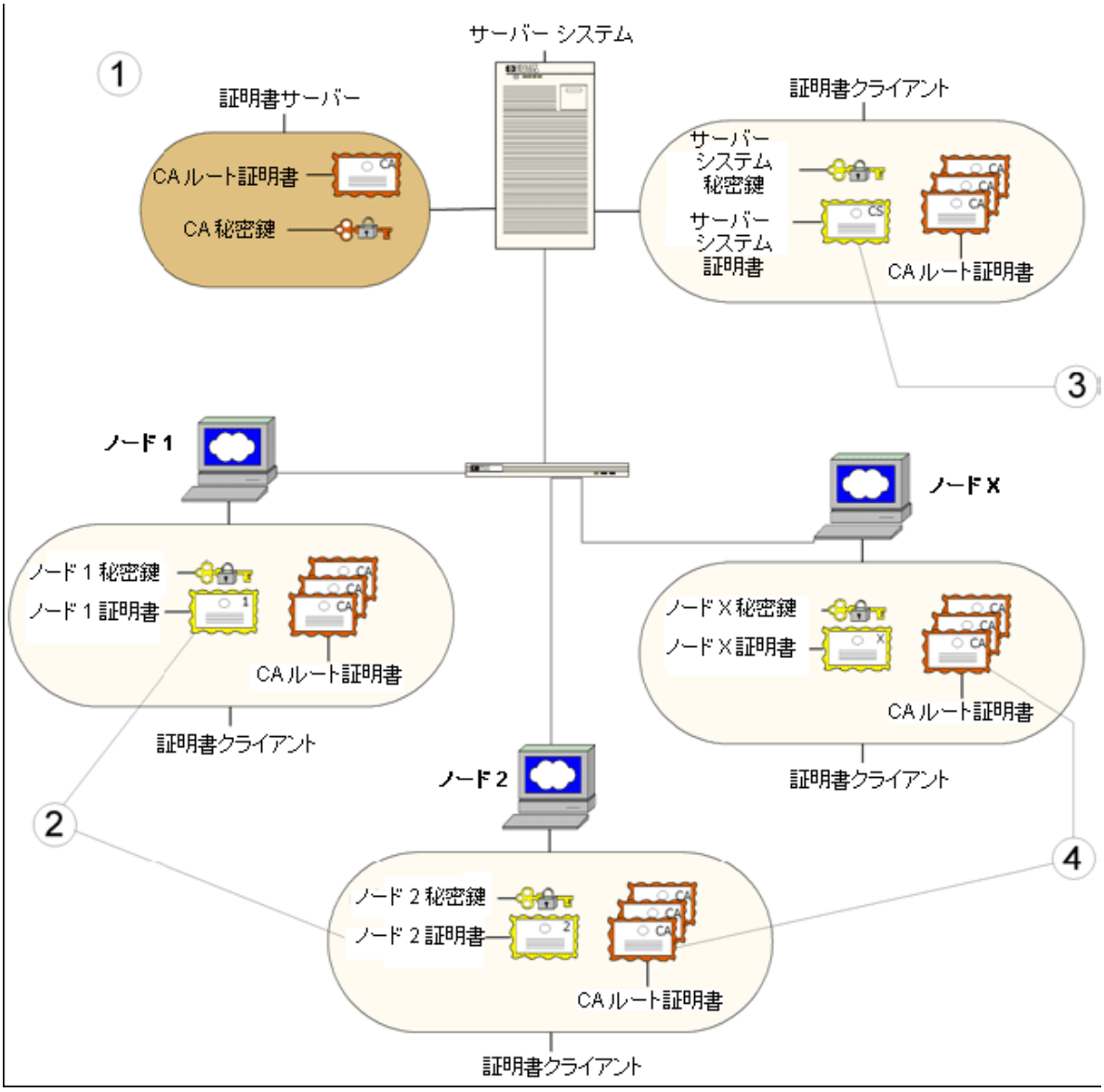


HP Operations Agent をホストする各システムでは、パラメータ `OvCoreId` に一意の ID 値が割り当てられますが、これはそのシステムに HP Operations Agent をインストールするときに作成されます。

注: エージェント ノードの `OvCoreId` パラメータは、システムのホスト名や IP アドレスが変わっても変更されません。

各エージェント ノードの OvCoreId は一意の識別子として使用され、対応するノード証明書に記載されます。OvCoreId の値は、インストール中に割り当てられます。

以下の図は、HP Operations Agent が配備された中で認証された通信を行うための環境を示しています。



1. サーバー システムは証明書サーバーをホストしますが、ここには必要な認証局 (CA) 機能が含まれています。
2. すべてのシステムは、証明書サーバーにより認証局の秘密鍵で署名された証明書を保有しています。
3. サーバー システムにも、自身の ID を管理するための証明書が必要となります。
4. すべてのシステムは信頼済みルート証明書のリストを保有しており、これには少なくとも 1 つの証明書が含まれている必要があります。信頼済みルート (CA) 証明書は、通信相手の ID を検証

するために使用されます。提示された証明書が信頼済み証明書のリストを使用して認証できる場合、通信相手は信頼されます。

証明書クライアントが複数の HPOM サーバーから管理されている場合には、信頼済みルート証明書のリストが必要です。

証明書

HP Operations Agent は、以下の 2 種類の証明書を使用します。

- ルート証明書
- ノード証明書

ルート証明書は自己署名証明書であり、ここには証明書サーバーの認証局の ID が含まれています。ルート証明書が所有する秘密鍵は証明書サーバー システムに格納され、未承認のアクセスから保護されます。認証局はルート証明書を使用し、すべての証明書にデジタル署名を行います。

管理環境内のすべてのエージェント ノードは、証明書サーバーが発行したノード証明書を受け取ります。証明書を発行する際、エージェント ノードで稼働する証明書クライアントは対応する秘密鍵をファイル システムに格納します。

注: ノード証明書には、一意のノード識別子である `OvCoreId` が含まれています。以下に `OvCoreId` の例を示します。
`d498f286-aa97-4a31-b5c3-806e384fcf6e`

各ノードは、ノード証明書により安全に認証されます。ルート証明書を使った署名の確認により、その環境内の他のすべてのノードがノード証明書を検証できます。ノード証明書は、クライアントおよびサーバー認証を使用する 2 つの HTTPS ノード間で SSL ベースの接続を確立するために使用され、すべての通信を暗号化するように設定できます。

証明書クライアントが提供する `ovcert` ツールは、キー ストアの内容を一覧したり、またはインストールされた証明書に関する情報を表示します。

HP Operations Agent 証明書サーバー

証明書サーバーが担当する機能は、次のとおりです。

- 自己署名ルート証明書を作成しインストールする
- ファイル システムから自己署名ルート証明書をインポートする
- ルート証明書の秘密鍵を格納する
- 証明書要求を許可または拒否する
- 新しい証明書と対応する秘密鍵を作成する、または証明書の手動インストールのためにインストール キーを作成する
- クライアントが信頼済みルート証明書を自動的に取得できるサービスを提供する

認証局

注: すべての HPOM サーバーは、自動的に認証局として設定されます。すべてのエージェント ノードに対するデフォルトの証明書サーバーは、そのノードに関連付けられた HPOM サーバーです。

認証局は証明書サーバーを構成する要素のひとつで、証明書管理における信頼性の中心部として機能します。この認証局により署名された証明書は有効な証明書であると判断され、信頼に値すると見なされます。認証局は、安全性の高い場所でホストされる必要があります。デフォルトでは、HPOM をホストするシステムにインストールされます。

認証局は信頼の根幹であるので、自己署名ルート証明書を伴って動作します。このルート証明書と対応する秘密鍵は、認証局としての稼働が可能な保護レベルで作成され、ファイル システムに格納されています。初期化が終了すると、認証局はルート証明書を使用して、承認された証明書要求に署名します。

証明書クライアント

証明書クライアントは、すべてのエージェント システム上で稼働します。

証明書クライアントの機能は、以下のとおりです。

- 証明書クライアントは、そのノードに有効な証明書があるかどうかをチェックします。
- そのノードに証明書がない場合は、ペアとなる新しい公開鍵と秘密鍵を生成し、そのノードの一意の ID (`OvCoreId` の値) を使って証明書リクエストを作成します。証明書クライアントはこの証明書リクエストをその他のノード プロパティと一緒に証明書サーバーへ送り、そのレスポンスを待ちます。
- その他のノード プロパティ (ノードの DNS 名や IP アドレスなど) は、証明書サーバーがリクエストの送信元を特定するのに役立ちます。
- 証明書サーバーが新しい証明書を発行すると、証明書クライアントはその証明書をノードにインストールします。証明書クライアントにより、すべての HTTPS ベース通信がこの証明書を使用することを保証できます。
- リクエストが正常に処理されない場合は、エラー記述が記録され、関連するステータスが設定されます。

さらに、証明書クライアントは以下のタスクを実行します。

- 証明書クライアントは証明書サーバーにアクセスし、サーバーの信頼済みルート証明書を更新します。
- ファイル システムからノード証明書とそれに対応する秘密鍵をインポートする機能をサポートします。
- 信頼済みルート証明書のインポートをサポートします。

- ステータス情報を提供します。ステータスの値には、「OK」、「valid certificate (有効な証明書)」、「no certificate (証明書なし)」、「certificate requested (証明書要求済み)」、および「certificate request denied (証明書要求拒否)」があります。

ルート証明書の更新と配布

たとえば、複数の証明書サーバーをホストする環境では、1 つまたは複数のノードの信頼済みルート証明書を更新する必要がある場合があります。

現在信頼済みのルート証明書をすべて、安全な方法で証明書クライアントに提供することも可能です。通常は、認証局のルート証明書を提供すれば十分です。しかし、たとえば複数の認証局が環境内に存在するときには、選択した証明書クライアントに 1 つまたは複数のルート証明書を追加で配布する必要がある場合があります。

証明書クライアントでコマンド ライン ツール ovcert を使用すれば、信頼済みルート証明書の更新を証明書サーバーに要求することができます。

第III部: HP Operations Agent のパフォーマンス コンポーネントの使用

HP Operations Agent はシステムの重要な要素のヘルス、パフォーマンス、リソース使用率、および可用性を示すメトリックを収集することにより、システムの監視を可能にします。

oacore データ コレクタ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。**parm** ファイルに定義されているクラスに基づいて、**oacore** データ コレクタは、システム全体のリソース使用率情報 (プロセス データ、さまざまなデバイスのパフォーマンス データ、トランザクション データ、論理システム データなど) を収集します。

oacore によって収集されるデータは、メトリック データストアに保存されます。メトリック データストアは、RDBMS (Relational Database Management System) に基づくデータストアです。メトリック データストアに記録されるデータのすべてのクラスに対して、1 つのデータベース ファイルが作成されます。これらのデータベース ファイルは、`datadir/databases/oa` で利用可能です。

Utility や **Extract** などのツールを使用すると、メトリック データストアに保存されている特定の情報を表示できます。履歴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することもできます。走査レポートには、**parm** ファイルの設定、各アプリケーションの名前と定義、アプリケーションでの追加/削除または変更、およびディスク領域の可用性がリストされます。

ベースライン計算のプロセスを使用すると、メトリック データストア に記録された履歴データに基づいて、基準値を計算して返すことができます。ベースライン データは、パフォーマンスの傾向を分析するための基準値を設定し、リソース使用率のパターンを分析するための最適なしきい値を動的に設定します。

第5章: データ収集の管理

HP Operations Agent は組み込まれたデータ コレクタの **oacore** を使用し、システム全体のパフォーマンスおよび状況データを継続的に収集し、収集したデータをメトリック データストアに記録します。extract や utility などのツールを使用すれば、記録したデータの表示、分析が可能です。HP Operations Agent を HP Performance Manager や HP Reporter などのデータ分析ツールと統合し、グラフやレポートを使用してデータを分析することもできます。

構成パラメータ ファイルの **parm** ファイルを使用すると、**oacore** データ 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** ファイルを使用すると、データの記録間隔と記録されたデータのタイプを制御できます。

メトリック データストアの使用

HP Operations Agent バージョン 12.00 では、ログ ファイルに基づくデータストアに代わって、メトリック データストアが使用されます。CODA、Scope、DSI ログ ファイルなどの複数のデータストアが、RDBMS (Relational Database Management System) に基づくデータストアに統合されています。

HP Operations Agent 11.xx から 12.00 へのアップグレード後に、HP Operations Agent 11.xx のデータ (CODA データベース ファイル、Scope ログ ファイル、DSI ログ ファイルに記録されている) は、読み取り専用モードで保持され、以下に保存されます。

```
/var/opt/perf/datafiles/
```

この古いデータには、ovcodautil、extract といったユーティリティ、または HP Performance Manager や HP Reporter といったレポート作成ツールを使用してアクセスできます。

注:

- HP Operations Agent 11.xx の古いデータは、メトリック データストア に移行されません。
- HP-UX AR システムの場合、HP Operations Agent 11.xx を 12.00 にアップグレードすると、HP Operations Agent 11.xx のデータは以下に保存されます。

  ```
  /var/opt/perf/datafiles/
  ```

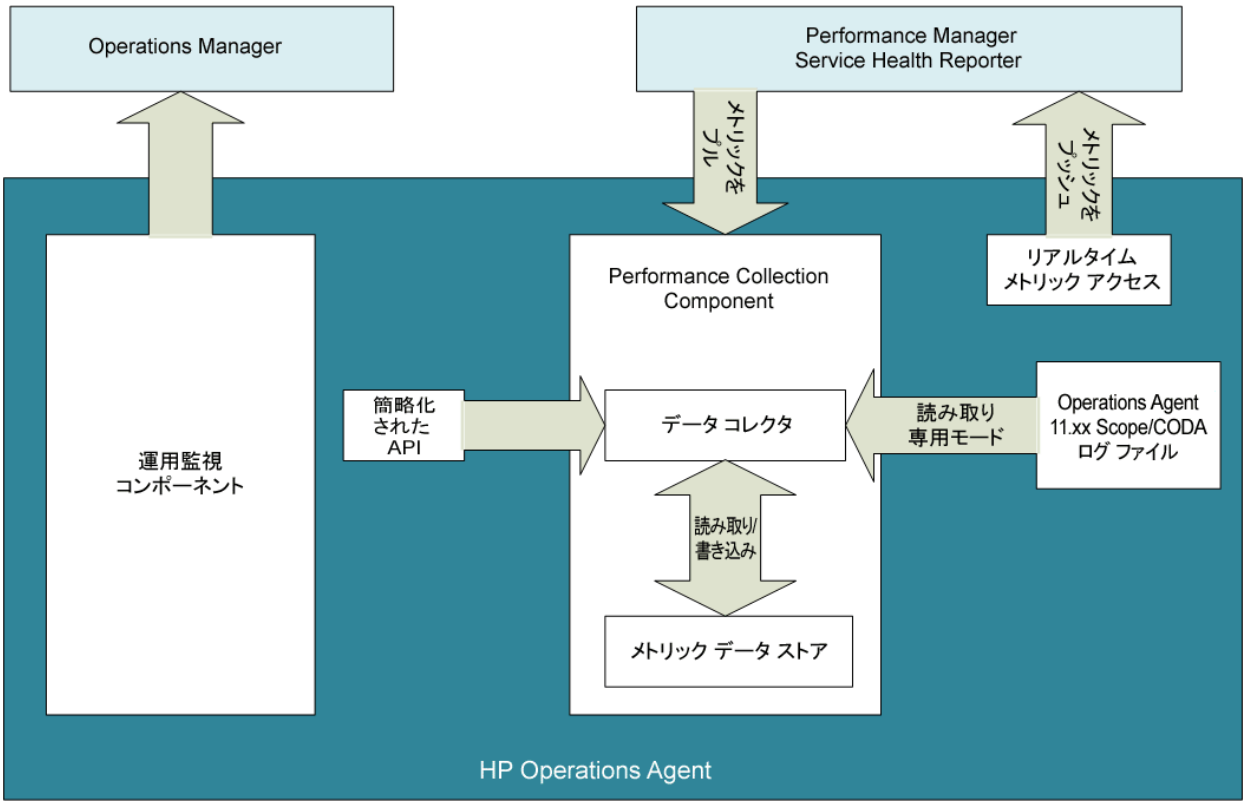
  このデータには、ovcodautil、extract といったユーティリティ、または HP Performance Manager や HP Reporter といったレポート作成ツールを使用してアクセスできます。
- HP-UX IA システムの場合、HP Operations Agent 11.xx を 12.00 にアップグレードすると、HP Operations Agent 11.xx のデータは以下に保存されます。

  ```
  /var/opt/OV/tmp/BackUp/
  ```

  このデータには、ovcodautil、extract、utility、または HP Performance Manager や HP Reporter といったレポート作成ツールを使用してアクセスできません。
- HP Operations Agent 11.xx から 12.00 へのアップグレード後に、HP Operations Agent 11.xx の

論理システム データ (ログファイルに保存される) には、HP Performance Manager や HP Reporter といったツールを使用してアクセスすることはできません。

以下の図に、HP Operations Agent の新しいアーキテクチャの概要を示します。



データの格納およびデータ収集のメカニズムは変更されましたが、ポリシーを使用したしきい値比較プロセスは同じです。

データの収集

現在のバージョンの HP Operations Agent では、**CODA** および **Scope** プロセス (UNIX および Linux ノード上の **scopeux** および Windows ノード上の **scopent**) は、**oacore** と呼ばれる 1 つのプロセスに統合されています。

oacore データ コレクタはさまざまな一連のシステム パフォーマンス メトリックを収集し、システムのヘルスとパフォーマンスに関して幅広い見解を提供します。データ コレクタは次の情報を取得します。

- システム全体のリソース使用率情報
- プロセス データ
- さまざまなデバイスのパフォーマンス データ

- トランザクション データ
- 論理システム データ

oacore データ コレクタは、**parm** ファイルに指定されているデータ クラスに基づいてデータを収集します。**parm** ファイルで定義できるデフォルトのパフォーマンス メトリック クラスは、global、application、process、disk device、lvolume、transaction、configuration、netif、CPU、filesystem、および host bus adapter です。

oacore データ コレクタによって収集されたデータは、クラス固有のデータベース ファイルに格納されます。メトリック データストアに記録されるデータのすべてのクラスに対して、1 つのデータベース ファイルが作成されます。これらのデータベース ファイルは、**datadir/databases/oa** で利用可能です。

oacore プロセスのステータスの検証

HP Operations Agent が HPOM から配布されると、**oacore** プロセスが自動的に起動します。HP Operations Agent がノード上にインストールされている場合は、ノード上で適切なライセンスが利用可能な場合にのみ、**oacore** プロセスが起動します。ライセンスの詳細については、『HP Operations Agent ライセンス ガイド』を参照してください。

oacore プロセスのステータスは、以下の場所にある `System.txt` ファイルに記録されます。

Windows の場合:

```
%ovdatadir%\log\System.txt
```

Linux の場合:

```
/var/opt/OV/log
```

このファイルには、**oacore** データ 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。

HP Operations Agent をインストールしたら、次の手順に従って **oacore** プロセスのステータスをチェックします。

1. HP Operations Agent ノードにログオンします。
2. 次のコマンドを実行します。

   Windows x64 システムの場合:

   ```
   "%ovinstalldir%bin\win64\ovc"
   ```

   Windows x86 システムの場合:

   ```
   "%ovinstalldir%bin\win32\ovc"
   ```

   AIX の場合:

   ```
   /usr/lpp/OV/bin/ovc
   ```

   HP-UX/Linux/Solaris の場合:

   ```
   /opt/OV/bin/ovc
   ```

oacore プロセスが実行中の場合、次の出力が表示されます。

```
oacore      Operations Agent Core              AGENT,OA     (25357)  Running
```

oacore プロセスの起動

oacore プロセスが実行されていない場合は、次の手順に従って **oacore** プロセスを起動します。

1. ノードにログオンします。
2. 以下のディレクトリに移動します。

   Windows x64 システムの場合:

   `%ovinstalldir%bin\win64\`

   Windows x86 システムの場合:

   `%ovinstalldir%bin`

   AIX の場合:

   `/usr/lpp/OV/bin`

   HP-UX/Linux/Solaris の場合:

   `/opt/OV/bin`
3. 以下のコマンドを実行して、プロセスを起動します。

   `ovc -start oacore`

メモ: oacore プロセスを停止するには、以下のコマンドを実行します。

`ovc -stop oacore`

注: oacore は、タイムアウトが発生するよりもずっと前にデータ アクセス要求を完了します。データ アクセス要求は、到着した順序に完了されます。

大きいデータ (例: 200 万個のプロセス レコード) を要求した場合、**oacore** が要求を完了するのに通常よりも長い時間がかかることがあります。**oacore** が大きいデータ アクセス要求を処理している間に、他の要求 (dsilog、extract など) がタイムアウトすることがあります。タイムアウト エラーが発生した場合、もう一度要求する必要があります。

データ ログ機能の検証

oacore データ コレクタによって収集されるメトリック データは、メトリック データストアに記録されます。以下の手順に従って、データ ログ機能を検証します。

1. ノードにログオンします。
2. 次のコマンドを実行します。

   Windows x64 システムの場合:

   `%ovinstalldir%bin\win64\ovcodautil –obj`

   Windows x86 システムの場合:

   `%ovinstalldir%bin\ovcodautil –obj`

AIX の場合:

```
/usr/lpp/OV/bin/ovcodautil -obj
```

HP-UX/Linux/Solaris の場合:

```
/opt/OV/bin/ovcodautil -obj
```

データが記録されている場合、記録されたすべてのメトリックのリストが表示されます。

データベース 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 には、データベース ファイルの自動管理機能があります。

デフォルトのパフォーマンス メトリック クラスを記録するデータベース ファイルにより使用されるディスク スペースの制御

デフォルトのパフォーマンス メトリック クラスを記録するデータベース ファイルのサイズは、**parm** ファイルに指定された最大サイズに依存します。**parm** ファイル内のサイズ指定を変更した場合、**oacore** はその変更を起動時にのみ検出します。

parm ファイルに指定された最大サイズに達した場合、クラスのデータベース ファイルはロール オーバーされます。ロールオーバーの際には、古い方から 20% のデータが削除されます。**parm** ファイルに size が指定されていない場合、データベース ファイルは最大サイズの 1 GB に達したときにロールオーバーされます。

> **注:** データベース ファイルのサイズは 1 MB 以下に設定することはできません。

parm ファイルに指定した最大サイズの変更

データストアに記録されるデータのすべてのクラスに対して、5 つのデータベース ファイルが作成されます。これらのデータベース ファイルのサイズは、**parm** ファイルに指定された最大サイズに依存します。

> **注:** データストアに記録されるデータのすべてのクラスに対して、1 つのデータベース ファイルが最初に作成されます。最初のデータベース ファイルのサイズが **parm** ファイルに指定した最大サイズの約 20% に達すると、2 つ目のデータベース ファイルが作成されます。この処理は、データベース ファイルが 5 つ作成されるまで続きます。

parm ファイル内のサイズ指定を変更した場合、**oacore** はその変更を起動時にのみ検出します。新しいサイズは、新しいデータベース ファイルにのみ有効となります。最終的に、ロールオーバーの際には、古い方のデータベース ファイルが削除されるため、**parm** ファイルに指定した最大サイズの約 20% に達すると、新しいデータベース ファイルが作成されます。

> **注:** 新しいサイズは新しいデータベース ファイルにのみ有効となるため、合計サイズは設定したサイズと異なる場合があります (設定したサイズより大きい場合もあれば小さい場合もありま

す)。ロールオーバーの際には古い方のデータベース ファイルから徐々に削除されるため、合計サイズは **parm** ファイルに指定したサイズに一致します。

例:

parm ファイル内のデータのクラスの最大サイズとして 10MB を指定した場合、それぞれ 2 MB サイズの 5 つのファイルが作成されます。

ロールオーバーの際に **parm** ファイル内に指定した最大サイズ (10 MB) に達すると、最も古いデータベース ファイル (2 MB のサイズ) から削除されます。

シナリオ 1 - 最大サイズを 10 MB から 5 MB に減らす。

変更を有効にするには、Performance Collection Component を再起動する必要があります。

すべての既存データベース ファイルは引き続き 2 MB です。

最終的に、ロールオーバーの際には古い方のデータベース ファイルから削除されるため、最大 1 MB のサイズの新しいデータベース ファイルが作成されます。

すべての古いファイル (再起動前に存在していたファイル) が削除されると、最大 1 MB のサイズの 5 つのデータベース ファイルが作成されます。したがって、**parm** ファイルに指定した 5 MB の最大サイズが得られます。

メモ: 新しいサイズは新しいデータベース ファイルにのみ有効になるため、合計サイズが設定したサイズを超える場合があります。ロールオーバーの際には古い方のデータベース ファイルから徐々に削除されるため、合計サイズは **parm** ファイルに指定したサイズに一致します。

シナリオ 2 - 最大サイズを 10 MB から 15 MB に増やす。

変更を有効にするには、Performance Collection Component を再起動する必要があります。

すべての既存データベース ファイルは引き続き 2 MB です。

最終的に、ロールオーバーの際には古い方のデータベース ファイルから削除されるため、最大 3 MB のサイズの新しいデータベース ファイルが作成されます。

すべての古いファイル (再起動前に存在していたファイル) が削除されると、最大 3 MB のサイズの 5 つのデータベース ファイルが作成されます。したがって、**parm** ファイルに指定した 15 MB の最大サイズが得られます。

メモ: 新しいサイズは新しいファイルにのみ有効となるため、合計サイズは設定したサイズより小さくなる場合があります。ロールオーバーの際には古い方のファイルから徐々に削除されるため、合計サイズは **parm** ファイルに指定したサイズに一致します。

カスタム データを記録するデータベース ファイルが使用するディスク スペースの制御

カスタム データを記録するデータベース ファイルの最大サイズは、デフォルトで 1 GB に設定されています。このサイズは設定できません。

データストアに記録されるデータのすべてのクラスに対して、1 つのデータベース ファイルが最初に作成されます。最初のデータベース ファイルのサイズが 1 GB の約 20% (200 MB) に達した場合、2 つ目のデータベース ファイルが作成されます。この処理は、200 MB のサイズのデータベース ファイルが 5 つ作成されるまで続きます。データベース ファイルは、最大サイズの 1 GB に達するとロールオーバーされます。ロール オーバー中、最も古いデータの 20% が削除されます (最も古いデータベース ファイルは完全に削除されます)。

注:

- UNIX システムの場合、**oacore** プロセスには、オープン ファイル記述子のような適切なリソースが必要です。ほとんどの UNIX システムでは、この制限は 256 (ulimit –n) に設定されます。**oacore** プロセスでは、通常のシナリオでの動作にはこの制限で十分です。カスタム クラスをさらに記録するように設定したシステムでは、オープン ファイル記述子は比例的に設定する必要があります。

  例:

  約 20 の管理クラスがある場合、オープン ファイル記述子を 512 に設定することをお勧めします。

  約 50 の管理クラスがある場合、オープン ファイル記述子を 1024 に設定することをお勧めします。

  オープン ファイル記述子を設定した後は、**oacore** プロセスを再起動する必要があります。
- Solaris プラットフォームでは、`ovc -start oacore` コマンドによって `/opt/perf/bin/runoacore` スクリプトが起動されます。このスクリプトは、**oacore** プロセスを起動する前に、オープン ファイル記述子の制限を 4096 に設定します。

データ収集の終了と再開始

oacore プロセス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(詳細は、「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 レベル目標 (SLO) を変更するとき トランザクション追跡に関する詳細は、「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、**oacore** プロセスの開始後にのみ有効になります。
- 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。行った変更は、**oacore** プロセスの再起動後にのみ有効になります。

データ収集の終了

ovpa スクリプトの stop オプションを使用すると、**oacore** 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、次のコマンドを実行します。

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
```

注: oacore 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。

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

oacore と実行中のその他のプロセスを再開始するには、`<InstallDir>/ovpa restart` を使用します。ここで、InstallDir 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。

oacore を再開始すると、Performance Collection Component は、**oacore** プロセスを停止する前に使用していたデータベース ファイルを使用し続けます。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。

注: 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

夏時間中に、システムの時刻が 1 時間進んでも、データの記録は変更なしで継続します。システムの時刻が 1 時間遅れると、最後に記録されたデータのタイムスタンプにシステムの時刻が同期するまで、データは 1 時間にわたって記録されません。

夏時間を無効に設定すると、システムの時刻が 1 時間進むため、次に記録されるレコードのタイムスタンプが 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 1 時間進めても、データの記録は変更なしで継続します。システムの時刻を 1 時間遅らせると、最後に記録されたデータのタイムスタンプにシステムの時刻が同期するまで、**oacore** は 1 時間にわたって記録を停止します。

> **注:** 収集におけるギャップを回避するには、システムの時刻をデータ収集の時刻と同期させる必要があります。

parm ファイルの使用

parm ファイルは、特定のパフォーマンス データをログに記録するための **oacore** データ コレクタに対する指示を含むテキスト ファイルです。

HP Operations Agent 12.00 を以前のバージョンの HP Operations Agent がインストールされていないノード上にインストールすると、**parm** ファイルが 2 つの異なるディレクトリに配置されます。以前のバージョンの HP Operations Agent から現在のバージョンにアップグレード中に、一方のディレクトリに配置された **parm** ファイルがアップグレードされます。詳細については、次のシナリオで **parm** ファイルの可用性を参照してください。

- HP Operations Agent 12.00 のインストール
- HP Operations Agent 12.00 へのアップグレード

HP Operations Agent 12.00 のインストール

HP Operations Agent 12.00を以前のバージョンの HP Operations Agent がインストールされていないノード上にインストールすると、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配置されます。

Windows の場合:

```
%ovinstalldir%\newconfig
```

```
%ovdatadir%
```

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

HP-UX/Linux/Solaris の場合:

/opt/perf/newconfig

/var/opt/perf

AIX の場合:

/usr/lpp/perf/newconfig

/var/opt/perf

oacore データ コレクタ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。

HP Operations Agent 12.00 へのアップグレード

Windows の場合

ノード上の HP Operations Agent を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。%ovdatadir% 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。この方法を用いることで、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、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** ファイルの変更の詳細は、「parm ファイルの変更」(126ページ)を参照してください。

UNIX/Linux の場合

すべての UNIX/Linux システム上で、m4 マクロ プロセッサ ユーティリティが **parm** ファイルの前処理に使用されます。Performance Collection Componentは、**/var/opt/perf** ディレクトリにある **parm** ファイルを前処理し、テキスト ファイルである **parm** 実行時ファイルを作成します。

次のコマンドを使用すると、**parm** 実行時ファイルを生成できます。

m4 -DPARMOS=<Operatingsystem>/var/opt/perf/parm > parm.m4

<Operatingsystem> には、該当する値を設定します。

注: <Operatingsystem> の値は、大文字と小文字を区別します。オペレーティング システムの該当する値を取得するには、次のコマンドを実行します。

uname

例:

Linux オペレーティング システム上で uname コマンドを実行すると、次の出力が生成されます。

Linux

m4 ユーティリティがない既存の **parm** ファイルは、以前のバージョンの HP Operations Agent で動作します。

システムは、インストール時に m4 をチェックします。m4 が存在しない場合は、m4 をシステムにインストールしてください。

詳細については、『HP Operations Agent および HP Operations Smart Plug-ins for Infrastructure インストール ガイド』の HP Operations Agent をインストールするための前提条件に関する項を参照してください。

注: カスタム アプリケーション定義を外部ファイルに追加し、#include(var/opt/perf/parm.apps) コマンドを使用して、それを **parm** ファイルに含めることができます。アプリケーションの定義方法の詳細は、「アプリケーション定義パラメータ」(146ページ)を参照してください。

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。
- 空白またはカンマを使用して、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 を再起動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
```

> **注:** `/opt/perf/bin/ovpa -restart scope` コマンドを使用することで、Performance Collection Component を再起動できます。このコマンドは、以前のバージョンから HP Operations Agent 12.00 にアップグレード後の、後方互換用にのみ保持されています。

AIX の場合:

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

> **注:** `/usr/lpp/perf/bin/ovpa -restart scope` コマンドを使用すると、Performance Collection Component を再起動できます。このコマンドは、以前のバージョンから HP Operations Agent 12.00 にアップグレード後の、後方互換用にのみ保持されています。

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** プロセスを再起動する必要があります。

Windows の場合:

```
%ovinstalldir%bin\ovpacmd stop RTMA
```

```
%ovinstalldir%bin\ovpacmd start 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

データ コレクタの **oacore** 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、次の操作を行います。

- 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

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 **oacore** で記録できます (「parm ファイルの変更」を参照してください)。

> **注:** システム設定パラメータの収集は、システムからのデバイスの起動、追加、または削除時にのみ行われます。

次の表の **parm** ファイル パラメータが oacore 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、「パラメータの説明」および「アプリケーション定義パラメータ」を参照してください。

oacore 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
パラメータの説明	
id	system ID
log	global application [=prm] [=all] ([=prm] は HP-UX のみ) process device=disk,lvm,cpu,filesystem,all (lvm は HP-UX および Linux のみ) transaction=correlator, resource (resource は HP-UX のみ) **注:** 以前のバージョンから HP Operations Agent 12.00 へのアップグレード後に、高レベルのトランザクション クラス メトリックのみが記録されます。 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
subprocinterval	値は秒単位 (HP-UX は対象外)
javaarg	true|false

oacore 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) IO=rate (値は KB/秒単位) nonew nokilled shortlived
appthreshold	cpu=percent
diskthreshold	util=rate
bynetifthreshold	iorate=rate
fsthreshold	util=rate
lvthreshold	iorate=rate
bycputhreshold	cpu=percent
fstypes	データの記録に特定のファイル システムのみを使用するには、構文 <file_system1>, <file_system2>, <file_system3> ... を使用します。 ファイル システムを除外するには、構文 !<file_system> を使用します。
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)
size	サイズ (値はメガバイト単位) process=nn (最大値は 4096) 以下のクラスの最大値は 2048 です。 global=nn application=nn device=nn transaction=nn logicalsystem=nn
collectioninterval	process=ss （値は秒 単位） global=ss

oacore 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
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)
project_app **メモ:** Solaris のみ	true false (Solaris 10 以降のみ)
proccmd	0 (プロセス コマンドのロギングを無効化) 1 (プロセス コマンドのロギングを有効化) 記録されるプロセス コマンドの最大長は常に 4096 です。
proclist **メモ:** Solaris のみ	all (グローバル ゾーンの Performance Collection Component は、グローバル ゾーンおよび非グローバル ゾーンで実行中のすべてのプロセスを監視) local (グローバル ゾーンの Performance Collection Component は、グローバル ゾーンで実行中のプロセスのみを監視) このパラメータは非グローバル ゾーンでは無効となります。
appproc **メモ:** Solaris のみ	all (グローバル ゾーンおよび非グローバル ゾーンに属するアプリケーションのプロセスを使用して APP_ メトリックを計算するように Performance Collection Component を設定) local (グローバル ゾーンのみに属するアプリケーションのプロセスを使用して APP_ メトリックを計算するように Performance Collection Component を設定) このパラメータは非グローバル ゾーンでは無効となります。

oacore 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
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） **メモ:** このパラメータは HP-UX では無効となります。上記のモードを切り替えるには、HP-UX で **midaemon -ignore_mt** コマンドを実行する必要があります。詳細情報は、「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」を参照してください。
cachemem	f または free (GBL_MEM_UTIL メトリックの値を計算する場合、HP Operations Agentはバッファ キャッシュ サイズを含めません) u または user (GBL_MEM_UTIL メトリックの値を計算する場合、HP Operations Agent はバッファ キャッシュ サイズを含めます)
アプリケーション定義パラメータ	
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 [,]
または	
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)

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

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データストアに収められます。最大 39 文字を指定できます。

Log

log パラメータは、**oacore** で収集するデータ型を指定します。

- **log global** を指定すると **oacore** でデータストアにグローバル 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 **oacore** でデータストア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` パラメータを指定すると、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 **oacore** によってインターバルごとにデータストアに記録されます。`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)。

  `log application=prm` パラメータを指定すると、**oacore** により、アクティブな Process Resource Manager (PRM) グループをデータストア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、**oacore** 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 **oacore** でデータストア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、**oacore** によって、各ディスク、論理ボリューム (HP-UX と Linux のみ)、CPU、ファイル システムに関する情報がデータストアに記録されます。

  > **注:** `lvm` は、HP-UX または Linux オペレーティング システムを実行しているシステム上でのみ使用してください。

  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、論理ボリューム、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。

  > **注:** デフォルトでは、netif レコードは **parm** ファイルに指定されていない場合でも記録されます。

  例:

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, netif
```

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、**oacore** 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 デバイスが、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データストアに記録されます。

- **log transaction** を指定すると **oacore** でデータストアに ARM トランザクションを記録できます。**oacore** 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。

  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。両方とも、**log transaction=resource**、**correlator** を指定することで記録できます。

  > **注:** HP Operations Agent 12.00 にアップグレードした場合、log transaction を指定すると、**oacore** で高レベルのトランザクション クラス メトリックのみ記録できます。

- **log logicalsystems** を指定すると **oacore** でデータストア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。

  > **注:** BYLS メトリックは、Windows および Linux プラットフォームではサポートされていません。
  >
  > HP Operations Agent 12.00 は、Xen、KVM、VMware vSphere、Hyper-V、および他の仮想化ドメインから BYLS メトリックを収集しません。

  AIX 6.1 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できます。詳細については、「oacore が使用する parm ファイル パラメータ」(128ページ)を参照してください。

> **注:** オプションなしで log を指定すると、**oacore** 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、**oacore** 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、重要な情報のみをデータストアに記録します。

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、**oacore** 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、**oacore** 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、**oacore** で記録されるデータを少なくできます。

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。

- 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、(カンマで区切って) 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。 **例:** 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	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。 **注:** このオプションは、Linux または Windows では使用できません。 **例:** disk=8.0 を指定すると、1 秒あた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。

プロセス データ用の procthreshold オプション (続き)

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。
IO	IO 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。 値はキロバイト単位です。 **例:** IO=100 を指定すると、1 秒あたりのプロセス I/O 率 (PROC_IO_BYTE_RATE) が 100 KB を超えたプロセスが記録されます。
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。 **oacore** 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

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。

- 最後の時間間隔内の 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 秒あた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データストアに記録されます。

parm ファイルのデフォルトの設定を使用すると、**oacore** 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、**oacore** 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 秒あた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、**oacore** は 1 秒あたりに 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、**oacore** 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、**oacore** はディスク容量の 70% を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 秒あたりの 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、**oacore** は 1 秒あたりに 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、**oacore** 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。

subprocinterval

subprocinterval パラメータを指定すると、**oacore** 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、**oacore** 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。詳細については、「データ記録間隔の設定」(152ページ)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、**oacore** 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 **oacore** 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。

> **注:** 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。

fstypes

fstypes パラメータを使うと、システムにある特定のファイル システムを監視できます。デフォルトでは、**oacore** コレクタは、システムに接続されているすべてのファイル システムのデータのログを記録します。fstypes パラメータを指定すると、特定のファイル システムだけのデータ収集を有効にできます。

parm ファイルでは、監視するファイル システムの名前に fstypes パラメータを設定する必要があります。カンマで区切ることで、複数のファイル システム名を指定できます。

このパラメータの構文は次のようになります。

fstypes=`<file_system1>,<file_system2>, ...`

`<file_system1>` および `<file_system2>` はファイル システムのタイプです。

HP-UX、Linux、AIX の場合、利用可能なすべてのファイル システムとファイル システム タイプのリストは、**/etc/fstab** ファイルにあります。

AIX の場合、利用可能なすべてのファイル システムとファイル システム タイプのリストは、**/etc/filesystems** ファイルにあります。

Solaris の場合、利用可能なすべてのファイル システムとファイル システム タイプのリストは、**/etc/vfstab** ファイルにあります。

Windows でファイル システム タイプを確認するには、Windows Explorer でディスク ドライブを右クリックして、**[プロパティ]** をクリックします。表示されるファイル システム値は、fstypes パラメータで指定できるファイル システム タイプです。

このパラメータには包含リストや除外リストを設定できます。ファイル システム タイプをカンマ区切りでリストにすると包含リストになります。このリストを使って、指定したファイル システムのデータだけを監視するようにエージェントを設定できます。

fstypesパラメータの設定中、オペレーティング システムのコマンドから返されるファイル システム名を使用します。このパラメータでは除外リストも設定できます。これは、! の文字で始まるファイル システム タイプのカンマ区切りリストです。! の文字をリストのはじめに指定すると、そのリストに属するすべてのファイル システムを監視しないようにエージェントを設定できます。

例 1:

fstypes = tmpfs, mvfs, nfs

例 2:

fstypes = !tmpfs, mvfs

例 3:

fstypes =

または

fstypes = *

* または blank 文字を指定すると、利用可能なすべてのファイル システムを監視するように HP Operations Agent を設定できます。

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、プロセスの詳細がデータストア 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「oacore が使用する parm ファイル パラメータ」(128ページ)を参照してください。

例:

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データストアに記録されます。

size

size パラメータは、データベース 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。**oacore** コレクタは、開始時にこれらの指定を読み込みます。

parm ファイルに指定された最大サイズに達した場合、デフォルトのパフォーマンス メトリック クラスを記録するデータベース ファイルはロール オーバーされます。**parm** ファイルに size が指定されていない場合、データベース ファイルは最大サイズの 1 GB に達したときにロール オーバーされます。

カスタム データを記録するデータベース ファイルの最大サイズは、デフォルトで 1 GB に設定されています。これらのデータベース ファイルは、最大サイズの 1 GB に達するとロール オーバーされます。

データベース ファイルにより使用されるディスク スペースの制御の詳細は、「データベース ファイルにより使用されるディスク スペースの制御」を参照してください。

javaarg

javaarg パラメータは、**proc_proc_argv1** メトリックの値値にのみ影響するフラグです。このパラメータはデフォルトでは true に設定されます。

javaarg が true である場合、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のに非常に便利です。プロセスのコマンド文字列にクラス名があれば、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。

javaarg が false に設定されているか、**parm** ファイルに定義されていない場合、proc_proc_argv1 メトリックは、プロセスのコマンド文字列の最初の引数の値と共に記録されます。

例:

次のプロセスを実行する場合:

```
java -XX:MaxPermSize=128m -XX:+CMSClassUnloadingEnabled -Xms8000m -Xmx8000m -
Dserver_port=1099 -jar ./ApacheJMeter.jar
```

- javaarg が **True** に設定されていれば、proc_proc_argv1 の値は -jar ./ApacheJMeter.jar になります。
- javaarg が **False** に設定されていれば、proc_proc_argv1 の値は -XX:MaxPermSize=128m になります。

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、**oacore** 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。

project_app

project_app パラメータを true に設定すると、Performance Collection Component は各 Solaris プロジェクトをアプリケーションとして見なします (プロジェクト ID をアプリケーション ID と見なします)。Solaris プロジェクトを無視するには、このパラメータを false に設定します。

注: project_app パラメータは、Solaris 10 以降のバージョンでのみサポートされています。

proclist

このパラメータは Solaris グローバル ゾーンでのみ使用できます。グローバル ゾーン以外で実行されている HP Operations Agent には影響しません。

グローバル ゾーンでこのパラメータを all に設定すると、Performance Collection Component はグローバル ゾーンとグローバル ゾーン以外にあるすべてのプロセスを監視します。グローバル ゾーンに属するプロセスのみを監視するには、このパラメータを local に設定します。

appproc

appproc パラメータは、Solaris でのみ使用できます。このパラメータはグローバル ゾーンでのみ使用できます。グローバル ゾーン以外で実行されている HP Operations Agent には影響しません。

グローバル ゾーンでこのパラメータを `all` に設定すると、すべての APP_ メトリックの値を計算しているときに、Performance Collection Component はグローバル ゾーンとグローバル ゾーン以外のアプ

リケーションのプロセスを含めます。APP_ メトリックの計算にグローバル ゾーンのアプリケーションだけを含めるには、このパラメータを **local** に設定します。

proccmd

proccmd パラメータを使用すると、データストアにプロセス コマンドを記録できます。デフォルトでは、このプロセスの値は 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。プロセス コマンドの記録を有効にするには、このパラメータの値を 1 に設定します。

> **メモ:** proccmd パラメータの記録は、このパラメータの値が 1 以上の場合にオンになります。記録されるプロセス コマンドの長さは、このパラメータに指定された値に関わりなく、常に 4096 です。

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。

Linux マシンでは、このパラメータはデフォルトでは true に設定されます。

このパラメータは HP-UX では無効となります。モードを切り替えるには、HP-UX で midaemon -ignore_mt コマンドを実行する必要があります。詳細については、「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」(159ページ)を参照してください。

HP-UX システム上で ignore_mt フラグをコマンドライン パラメータとして midaemon に設定すると、いくつかのメトリックが影響を受けます。影響を受けるメトリックのリストの詳細は、「parm ファイル」(314ページ)を参照してください。

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。

> **注:** Windows、Linux、Solaris システムで、同時マルチスレッディング (Simultaneous Multi-Threading (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。

HP-UX システム上で ignore_mt フラグを midaemon に設定中に影響を受けるメトリック

グローバル メトリック クラス	
GBL_CPU_TOTAL_UTIL	GBL_CPU_CSWITCH_UTIL_CUM
GBL_CPU_TOTAL_UTIL_CUM	GBL_CPU_CSWITCH_UTIL_HIGH
GBL_CPU_TOTAL_UTIL_HIGH	GBL_CPU_CSWITCH_TIME

GBL_CPU_TOTAL_TIME	GBL_CPU_CSWITCH_TIME_CUM
GBL_CPU_TOTAL_TIME_CUM	GBL_CPU_INTERRUPT_UTIL
GBL_CPU_SYS_MODE_UTIL	GBL_CPU_INTERRUPT_UTIL_CUM
GBL_CPU_SYS_MODE_UTIL_CUM	GBL_CPU_INTERRUPT_UTIL_HIGH
GBL_CPU_SYS_MODE_TIME	GBL_CPU_INTERRUPT_TIME
GBL_CPU_SYS_MODE_TIME_CUM	GBL_CPU_INTERRUPT_TIME_CUM
GBL_CPU_TRAP_TIME	GBL_CPU_VFAULT_TIME
GBL_CPU_TRAP_TIME_CUM	GBL_CPU_VFAULT_TIME_CUM
GBL_CPU_TRAP_UTIL	GBL_CPU_VFAULT_UTIL
GBL_CPU_TRAP_UTIL_CUM	GBL_CPU_VFAULT_UTIL_CUM
GBL_CPU_TRAP_UTIL_HIGH	GBL_CPU_VFAULT_UTIL_HIGH
GBL_CPU_USER_MODE_TIME	GBL_CPU_IDLE_UTIL
GBL_CPU_USER_MODE_TIME_CUM	GBL_CPU_IDLE_UTIL_CUM
GBL_CPU_USER_MODE_UTIL	GBL_CPU_IDLE_UTIL_HIGH
GBL_CPU_USER_MODE_UTIL_CUM	GBL_CPU_IDLE_TIME
GBL_CPU_NICE_UTIL	GBL_CPU_IDLE_TIME_CUM
GBL_CPU_NICE_UTIL_CUM	GBL_CPU_NORMAL_UTIL
GBL_CPU_NICE_UTIL_HIGH	GBL_CPU_NORMAL_UTIL_CUM
GBL_CPU_NICE_TIME	GBL_CPU_NORMAL_UTIL_HIGH
GBL_CPU_NICE_TIME_CUM	GBL_CPU_NORMAL_TIME
GBL_CPU_NNICE_UTIL	GBL_CPU_NORMAL_TIME_CUM
GBL_CPU_NNICE_UTIL_CUM	GBL_CPU_SYSCALL_UTIL
GBL_CPU_NNICE_UTIL_HIGH	GBL_CPU_SYSCALL_UTIL_CUM
GBL_CPU_NNICE_TIME	GBL_CPU_SYSCALL_UTIL_HIGH
GBL_CPU_NNICE_TIME_CUM	GBL_CPU_SYSCALL_TIME
GBL_CPU_REALTIME_UTIL	GBL_CPU_SYSCALL_TIME_CUM
GBL_CPU_REALTIME_UTIL_CUM	GBL_CPU_WAIT_UTIL

GBL_CPU_REALTIME_UTIL_HIGH	GBL_CPU_WAIT_TIME
GBL_CPU_REALTIME_TIME	GBL_CPU_WAIT_TIME_CUM
GBL_CPU_REALTIME_TIME_CUM	GBL_CPU_WAIT_UTIL_CUM
GBL_CPU_CSWITCH_UTIL	GBL_CPU_WAIT_UTIL_HIGH
アプリケーション メトリック クラス	
APP_PRM_CPU_TOTAL_UTIL_CUM	APP_CPU_NORMAL_UTIL
APP_CPU_NNICE_UTIL	APP_CPU_USER_MODE_UTIL
APP_CPU_NNICE_TIME	APP_CPU_TOTAL_TIME
APP_CPU_TOTAL_UTIL	APP_CPU_SYS_MODE_TIME
APP_CPU_TOTAL_UTIL_CUM	APP_CPU_NICE_TIME
APP_CPU_SYS_MODE_UTIL	APP_CPU_REALTIME_TIME
APP_CPU_NICE_UTIL	APP_CPU_NORMAL_TIME
APP_CPU_REALTIME_UTIL	APP_CPU_USER_MODE_TIME
PROC メトリック クラス	
PROC_CPU_TOTAL_UTIL	PROC_CPU_REALTIME_UTIL
PROC_CPU_TOTAL_UTIL_CUM	PROC_CPU_REALTIME_UTIL_CUM
PROC_CPU_TOTAL_TIME	PROC_CPU_REALTIME_TIME
PROC_CPU_TOTAL_TIME_CUM	PROC_CPU_REALTIME_TIME_CUM
PROC_CPU_SYS_MODE_UTIL	PROC_CPU_CSWITCH_UTIL
PROC_CPU_SYS_MODE_UTIL_CUM	PROC_CPU_CSWITCH_UTIL_CUM
PROC_CPU_SYS_MODE_TIME	PROC_CPU_CSWITCH_TIME
PROC_CPU_SYS_MODE_TIME_CUM	PROC_CPU_CSWITCH_TIME_CUM
PROC_CPU_USER_MODE_UTIL	PROC_CPU_INTERRUPT_UTIL
PROC_CPU_USER_MODE_UTIL_CUM	PROC_CPU_INTERRUPT_UTIL_CUM
PROC_CPU_USER_MODE_TIME	PROC_CPU_INTERRUPT_TIME
PROC_CPU_USER_MODE_TIME_CUM	PROC_CPU_INTERRUPT_TIME_CUM
PROC_CPU_NICE_UTIL	PROC_CPU_NORMAL_UTIL

PROC_CPU_NICE_UTIL_CUM	PROC_CPU_NORMAL_UTIL_CUM
PROC_CPU_NICE_TIME	PROC_CPU_NORMAL_TIME
PROC_CPU_NICE_TIME_CUM	PROC_CPU_NORMAL_TIME_CUM
PROC_CPU_NNICE_UTIL	PROC_CPU_SYSCALL_UTIL
PROC_CPU_NNICE_UTIL_CUM	PROC_CPU_SYSCALL_UTIL_CUM
PROC_CPU_NNICE_TIME	PROC_CPU_SYSCALL_TIME
PROC_CPU_NNICE_TIME_CUM	PROC_CPU_SYSCALL_TIME_CUM
PROC_CPU_ALIVE_TOTAL_UTIL	
PROC_CPU_ALIVE_USER_MODE_UTIL	
PROC_CPU_ALIVE_SYS_MODE_UTIL	
BYCPU メトリック クラス	
BYCPU_CPU_TOTAL_UTIL	BYCPU_CPU_INTERRUPT_TIME
BYCPU_CPU_TOTAL_UTIL_CUM	BYCPU_CPU_INTERRUPT_TIME_CUM
BYCPU_CPU_TRAP_TIME	BYCPU_CPU_INTERRUPT_UTIL
BYCPU_CPU_TRAP_TIME_CUM	BYCPU_CPU_INTERRUPT_UTIL_CUM
BYCPU_CPU_TRAP_UTIL	BYCPU_CPU_CSWITCH_TIME
BYCPU_CPU_TRAP_UTIL_CUM	BYCPU_CPU_CSWITCH_TIME_CUM
BYCPU_CPU_USER_MODE_TIME	BYCPU_CPU_CSWITCH_UTIL
BYCPU_CPU_USER_MODE_TIME_CUM	BYCPU_CPU_CSWITCH_UTIL_CUM
BYCPU_CPU_USER_MODE_UTIL	BYCPU_CPU_VFAULT_TIME
BYCPU_CPU_USER_MODE_UTIL_CUM	BYCPU_CPU_VFAULT_TIME_CUM
BYCPU_CPU_NICE_TIME	BYCPU_CPU_VFAULT_UTIL
BYCPU_CPU_NICE_TIME_CUM	BYCPU_CPU_VFAULT_UTIL_CUM
BYCPU_CPU_NICE_UTIL	BYCPU_CPU_REALTIME_TIME
BYCPU_CPU_NICE_UTIL_CUM	BYCPU_CPU_REALTIME_TIME_CUM
BYCPU_CPU_NNICE_TIME	BYCPU_CPU_REALTIME_UTIL
BYCPU_CPU_NNICE_TIME_CUM	BYCPU_CPU_REALTIME_UTIL_CUM

BYCPU_CPU_NNICE_UTIL	BYCPU_CPU_NORMAL_TIME
BYCPU_CPU_NNICE_UTIL_CUM	BYCPU_CPU_NORMAL_TIME_CUM
BYCPU_CPU_TOTAL_TIME	BYCPU_CPU_NORMAL_UTIL
BYCPU_CPU_TOTAL_TIME_CUM	BYCPU_CPU_NORMAL_UTIL_CUM
BYCPU_CPU_SYS_MODE_TIME	BYCPU_CPU_SYSCALL_TIME
BYCPU_CPU_SYS_MODE_TIME_CUM	BYCPU_CPU_SYSCALL_TIME_CUM
BYCPU_CPU_SYS_MODE_UTIL	BYCPU_CPU_SYSCALL_UTIL
BYCPU_CPU_SYS_MODE_UTIL_CUM	BYCPU_CPU_SYSCALL_UTIL_CUM
SYSCALL メトリック クラス	
SYSCALL_CPU_TOTAL_TIME	SYSCALL_CPU_TOTAL_TIME_CUM

cachemem

parm ファイルの cachemem パラメータを使うと、メモリ使用率合計のデータをレポートするときに、バッファ キャッシュ サイズを含めるようにエージェントを設定できます。

次のいずれかを行うときに、cachemem パラメータを設定できます。

1. **cachemem パラメータを free (f) に設定します。**

   デフォルトでは、この値は free (f) に設定されます。以下のメトリック値が影響を受けます。

   GBL_MEM_UTIL - バッファ キャッシュ サイズを含めません。

   GBL_MEM_FREE_UTIL - バッファ キャッシュ サイズを含めます。

2. **cachemem パラメータを user (u) に設定します。**

   以下のメトリック値が影響を受けます。

   GBL_MEM_UTIL - バッファ キャッシュ サイズを含めます。

   GBL_MEM_FREE_UTIL - バッファ キャッシュ サイズを含めません。

注:

AIX マシンでは、cachemem パラメータを **free (f)** に設定して topas コマンドと一致させる必要があります。

Solaris マシンでは、cachemem パラメータは ZFS の Adaptive Replacement Cache (ARC) にのみ適用されます。

cachemem パラメータはデフォルトでは **user (u)** に設定されます。parm ファイルでこのパラメータを **free (f)** に設定した後で、データ コレクタを再起動します。parm ファイルの cachemem パラメータが **free** に設定されている場合、GBL_MEM_UTIL は ZFS ARC を除外します。

アプリケーション定義パラメータ

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

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。

> **注:** 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。

例:

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。

> **注:** 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。

アプリケーション名

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。

アプリケーションを指定するときは、次の規則と表記規則が適用されます。

- 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価記号 (=) は省略できます。
- application 名は、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。
- 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。
- 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。

例:

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

```
application xyz
file xyz*,startxyz
```

ここで、

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)。

> **注:** 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。

- **UNIX/Linux の場合:**
  - `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。
- **Windows の場合**:
  - `winword` 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) 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、すべてのファイル指定で最大限の 1000 指定しても 1 つの指定と数えられます。

注: Windows の場合、parm ファイルでアプリケーションの実行可能ファイルを定義する場合、ファイル拡張子は不要です。たとえば、parm ファイルでは、.exe 拡張子なしで winword を定義できます。

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価記号 (=) は省略できます。

ファイル名は、1 つのパラメータ行に (カンマで区切って) 複数入力しても、ファイル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。

例:

UNIX/Linux の場合:

```
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
```

Windows の場合:

```
application payroll
file account1,basepay,endreport
application Office
file winword* excel*
file 123* msaccess*
```

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。

注: cmd 修飾子を除くと、**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。

argv1 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。

例:

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
```

次の例は、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、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。コマンド文字列は、実行済みのプログラムおよびその引数 (パラメータ) から構成されています。このパラメータでは、アスタリスクの使用以外にも詳細なワイルドカード文字が使用できます。

正規表現と同じように、詳細なパターン マッチが使用できます。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。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。

以下の例は、cmd パラメータの使用例を示しています。

```
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。

形式は以下のようになります。

```
application <application_name>

file <file_name>

user [<Domain_Name>]\<User_Name>
```

user パラメータのドメイン名はオプションです。ローカル以外のシステムのユーザー名を指定する場合は、ドメイン名を指定する必要があります。

例:

```
application test_app

file test

user TestDomain\TestUser
```

ドメイン名を指定せずにユーザー名を指定すると、すべてのユーザー名がローカル システムのユーザー名であると認識されます。

例:

```
application Prog_Dev
file vi,xb,abb,ld,lint
user ted,rebecca,test*
```

ワイルドカード アスタリスク (*) を使用すると、アプリケーションに属する、アスタリスク (*) の前後に同じ文字列が使用されるユーザー名を指定できます。

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。

例:

```
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。

例:

```
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。

例:

```
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、HP Operations Agent 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。

> **注: 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

コマンド文字列	ユーザー ログイン	アプリケーション
./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

oacore 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 データおよびデータの他のすべてのクラスの場合が 300 秒です。**parm** ファイルの collection interval 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。

フレームに対するデータ収集の設定

HP Operations Agent を 1 つのみの LPAR ノード上にインストールすることで、1 つのフレームで使用可能なすべての AIX LPAR から、パフォーマンス データを収集できます。監視対象のすべての LPAR が存在する AIX フレームからパフォーマンス データを収集するようにエージェントを設定することもできます。

フレームの監視を有効にすると、次の利点があります。

- 以下の設定情報を収集できます。
  - 監視対象の LPAR が存在するフレームの名前と UUID
  - フレームのモデル、シリアル番号、タイプ
  - フレームの CPU 設定とメモリ容量
- 追加の情報を使用して、フレームのリソース使用率を分析できます
- HP Performance Manager などの HP データ分析ツールを使用して、フレームとすべての LPAR の CPU 使用率を分析できます

タスク 1: パスワードなしの SSH アクセスを設定する

ハードウェア管理コンソール (HMC) システムでは、LPAR ノード (エージェントをインストールしたノード) と HMC システム間にパスワードなしの SSH アクセスを設定できます。ユーザーに対してパスワードなしの SSH アクセスを設定すると、認証のためのパスワードを入力する必要なく、LPAR ノードからリモートで HMC システム上の SSH プロトコルを使用してコマンドを実行できます。

以下の手順に従ってください。

パスワードなしの SSH アクセスの設定

LPAR ノードの場合

1. root として LPAR ノードにログオンします。
2. 以下のコマンドを実行してディレクトリを作成します。

   ```
   # mkdir .ssh
   ```

   注: 次のコマンドを実行して `.ssh` ディレクトリを削除できます: `# rm -rf .ssh`

3. 次のコマンドを実行して、公開/秘密 RSA キー ペアを生成します。

   ```
   # ssh-keygen -t rsa
   ```

4. キーの保存先となるファイルを指定します。

   ```
   //.ssh/<filename>_rsa
   ```

   秘密キーは `//.ssh/ <filename>_rsa` に保存され、公開キーは `//.ssh/ <filename>_rsa.pub` に保存されます。

5. 以下のコマンドを実行して、キーを表示します。

   ```
   # cat ./ssh
   ```

   ```
   # cat .ssh/<filename>_rsa.pub
   ```

生成されたキーが表示されます。

HMC マシンの場合

1. ユーザー資格情報を使用して、HMC マシンにログオンします。
2. 以下のコマンドを実行して、パスワードなしのアクセスを設定します。

   ```
   mkauthkeys -a 'ssh-rsa<key>’
   ```

   ここで、`<key>` は手順 3 で生成したキーです。

   注: キー全体を 1 つの行に記述する必要があります。

パスワードなしの SSH アクセスの確認

パスワードなしの SSH アクセスがユーザーに対して作成されたかどうかを確認するには、次の手順を実行します。

1. root として LPAR ノードにログオンします。
2. 次のコマンドを実行します。

   ```
   ssh hmcuser@hmchost.example.domain.com lssyscfg r sys
   ```

コマンドを実行したときにパスワードの入力を促すプロンプトが表示されずにフレーム リストが表示されれば、ユーザーは正しく設定されており、フレーム使用率を監視することができます。

タスク 2: HMC システムでフレーム使用率の監視を有効にする

1. root として HMC システムにログオンします。
2. 次のコマンドを実行します。

   ```
   chlparutil -r config -s 60
   ```

タスク 3: HP Operations Agent を設定する

1. LPAR ノードにログオンします。
2. `/var/opt/perf` ディレクトリへ移動します。
3. 新しいファイルを作成します。
4. 作成したファイルをテキスト エディタで開き、このファイルに次の内容を追加します。

   <hmcusername>@<hmc_fqdn>

   ここで、

   <hmcusername> は「タスク 1: パスワードなしの SSH アクセスを設定する」(153ページ)で HMC システムへのパスワードなしの SSH アクセス許可を付与されたユーザーです。

   <hmc_fqdn> は、HMC ホストの完全修飾ドメイン名です。

   たとえば、次のようになります。

   ```
   hmcusername@hmchost.example.domain.com
   ```

5. このファイルを `hmc` として `/var/opt/perf` ディレクトリに保存します。
6. `parm` ファイルの `logicalsystem` パラメータを設定して、収集プロセスを開始します。`parm` ファイルの設定の詳細は、「parm ファイルの使用」(124ページ) を参照してください。
7. Performance Collection Component を再起動します。

   HP Operations Agent はフレーム固有のパフォーマンス データを 5 分間隔で収集します。

Linux 上での GlancePlus に対するグローバルおよびプロセスのシステム コール メトリックの有効化

Linux マシンでは、GlancePlus に対する GBL_SYSCALL、SYSCALL、および PROCSYSCALL のパフォーマンス メトリック値の収集を有効にできます。メトリック収集の設定を開始する前に、Linux マシン上で FTRACE を使用するすべてのプロセスを停止してください。

注: これらのメトリックは、RHEL/OEL/CentOS 6.1 以降、SLES 11 SP1 以降、Ubuntu 11.10 以降、Debian 6.0 以降でサポートされています。

GlancePlus のメトリック収集は、次のいずれかのメソッドで有効にできます。

- `init_ftrace.sh` を使用したメトリック コレクションの設定
- 手動でのメトリック コレクションの設定

init_ftrace.sh を使用したメトリック コレクションの設定

Linux マシンで `init_ftrace.sh` を使用してメトリック コレクションを設定するには、以下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. Linux マシンのすべての Performance Collection Component プロセスを停止するには、以下のコマンドを実行します。

   ```
   /opt/perf/bin/ovpa stop all
   ```

3. Linux マシンのすべての `midaemon` プロセスを停止します。`midaemon` プロセスをチェックするには、以下のコマンドを実行します。

   ```
   ps -ef I grep midaemon
   ```

   Linux マシンで `midaemon` が実行中の場合、以下のコマンドを実行します。

   ```
   killall midaemon
   ```

   これですべての `midaemon` プロセスが停止します。
4. メトリック コレクションを設定するには、以下のコマンドを実行します。

   ```
   /opt/perf/bin/init_ftrace.sh
   ```

   The command mounts debugfs and enables FTRACE on your Linux machine.(コマンドが debugfs をマウントし、Linux マシン上で FTRACE を有効にします)If FTRACE is already running on your machine with any other application, message appears as:(他のアプリケーションによって FTRACE がマシン上ですでに実行されている場合、次のメッセージが表示されます)

   Do you want to reset the FTRACE interface for use with midaemon?(midaemon で使用するために FTRACE インターフェイスをリセットしますか?)(Y/N)

次のいずれかを選択できます。

Y – Linux マシン上の FTRACE インターフェイスをリセットします。

N – メトリック コレクションを設定しません。メトリックは GlancePlus に表示されません。

5. すべての Performance Collection Component プロセスを `midaemon` プロセスとメトリック コレクションとともに起動するには、以下のコマンドを実行します。

   ```
   /opt/perf/bin/ovpa start all
   ```

GlancePlus 内の次の画面で、パフォーマンス メトリックを確認します。

Y - SYSCALL メトリックを確認します

L - PROCSYSCALL メトリックを確認します

> **注:** GlancePlus がシステム コール画面にない場合、debugfs がマウントされ FTRACE が有効であっても、システムのパフォーマンスには影響しません。**/sys/kernel/debug** から debugfs をマウント解除するには、`umount /sys/kernel/debug` コマンドを実行します。

手動でのメトリック コレクションの設定

Linux マシンで手動でメトリック コレクションを設定するには、以下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. Linux マシンのすべての Performance Collection Component プロセスを停止するには、以下のコマンドを実行します。

   ```
   /opt/perf/bin/ovpa stop all
   ```

3. Linux マシンのすべての `midaemon` プロセスを停止します。`midaemon` プロセスをチェックするには、以下のコマンドを実行します。

   ```
   ps -ef I grep midaemon
   ```

   Linux マシンで `midaemon` が実行中の場合、以下のコマンドを実行します。

   ```
   killall midaemon
   ```

   これですべての `midaemon` プロセスが停止します。

4. 手動による Linux マシンへの debugfs のマウント

   > **注:** Linux マシン上に debugfs がマウントされている場合、この手順を省略します。debugfs がマウントされているかどうかを確認するには、以下のコマンドを実行します。
   >
   > ```
   > cat /proc/mounts |grep debugfs
   > ```
   >
   > コマンドによって、以下のいずれかの出力が生成されます。
   >
   > **nodev /sys/kernel/debug debugfs rw,relatime 0 0** - マシン上に debugfs がマウントされています。
   >
   > **nil** - マシン上に debugfs がマウントされていません。

証明書をバックアップするには、以下の手順に従います。

i. 次のコマンドを実行します。

```
mount -t debugfs nodev /sys/kernel/debug
```

以下のように出力されます。

nil - マシン上に debugfs が正常にマウントされています。

マシン上に debugfs が正常にマウントされていないと、エラー メッセージが表示されます。

5. 手動による Linux マシンでの FTRACE の有効化

> **注:** Linux マシン上で FTRACE が有効になっている場合は、この手順を省略します。FTRACE が有効になっているかどうかを確認するには、以下のコマンドを実行します。
>
> ```
> cat /proc/sys/kernel/ftrace_enabled
> ```
>
> コマンドによって、以下のいずれかの出力が生成されます。
>
> **1** - マシン上で FTRACE が有効になっています。
>
> **0** - マシン上で FTRACE が有効になっていません。

手動で FTRACE を有効にするには、以下の手順を実行します。

i. 次のコマンドを実行します。

```
echo "1" >/proc/sys/kernel/ftrace_enabled
```

以下のように出力されます。

nil - マシン上で FTRACE が正常に有効になっています。

マシン上で FTRACE が正常に有効になっていないと、エラー メッセージが表示されます。

6. すべての Performance Collection Component プロセスを `midaemon` プロセスとメトリック コレクションとともに起動するには、以下のコマンドを実行します。

```
/opt/perf/bin/ovpa start all
```

GlancePlus 内の次の画面で、パフォーマンス メトリックを確認します。

Y - SYSCALL メトリックを確認します

L - PROCSYSCALL メトリックを確認します

> **注:** GlancePlus がシステム コール画面にない場合、debugfs がマウントされ FTRACE が有効であっても、システムのパフォーマンスには影響しません。**/sys/kernel/debug** から debugfs をマウント解除するには、`umount /sys/kernel/debug` コマンドを実行します。

トラブルシューティング

ここでは、メトリック コレクションを設定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。

問題:

midaemon プロセス エラーが発生する場合、内容の詳細は /var/opt/perf/status.mi を参照してください。midaemon がその共有メモリ セグメント内の領域を使い果たした場合、midaemon プロセス エラーが発生する場合があります。

対策:

この問題を回避するには、不要な共有メモリをクリアする必要があります。共有メモリをクリアするには、次の手順を実行します。

1. 次のコマンドを実行します。

   ```
   ipcs -m | grep 0x0c6629c9
   ```

   コマンドによって、2 番目のデータ フィールドに変数値とともに出力が生成されます。共有メモリをクリアするには、以下のコマンドを実行します。

   ```
   ipcrm –m <field_value>
   ```

   例

   以下の例は、共有メモリをクリアする方法を示しています。

   ```
   ipcs -m | grep 0x0c6629c9
   output= 0x0c6629c9 18841617 root 640 8704448 7
   ipcrm –m 18841617
   ```

2. midaemon を再起動するには、以下のコマンドを実行します。

   ```
   /opt/perf/bin/ovpa restart
   ```

ハイパースレッディングまたは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 Agent 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、以下の手順を実行します。

HP-UX の場合

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。

   注: 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。

   ```
   /opt/perf/bin/midaemon -ignore_mt
   ```

5. 以下のコマンドを実行し、HP Operations Agent を起動します。

   ```
   /opt/OV/bin/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。

HP Operations Agentを再起動すると、Performance Collection Component はスレッドを基にした正規化を使用して CPU データを再びログに記録し始めるため、前述の手順を使用してもう一度 Performance Collection Component を設定する必要があります。コアに基づく正規化を常に使用するようにエージェントを有効にするには、以下の手順を実行します。

1. エージェント ノードで、次の場所に移動します。

   /var/opt/perf

2. テキスト エディタで次のファイルを開きます。

   vppa.env

3. MIPARMS パラメータを ignore_mt に設定します。
4. ファイルを保存します。
5. 以下のコマンドを実行してエージェントを再起動します。

   ```
   /opt/OV/bin/opcagt -start
   ```

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. 以下のコマンドを使用して、HP Operations Agent を起動します。

   Windows の場合:

   ```
   %ovinstalldir%bin\opcagt -start
   ```

   Linux および 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。

第6章: utility プログラムの使い方

Utility プログラムは、収集パラメータ (parm) ファイルおよびアラーム定義 (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。utility ツールを使用すると、次の作業を行うことができます。

- HP Operations Agent データストアのスキャン、および以下の項目を含んだレポートの生成
  - 扱われている日時
  - 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
- 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データストアに保存されたデータの処理と、履歴データ内のアラーム状態の検出

utility プログラムの実行

HP Operations Agent 12.00 では、utility プログラムの実行はコマンド ライン モードでのみサポートされます。コマンド オプションとそれに関連する引数は、コマンド ライン インターフェイスを通じて utility プログラムに引き渡されます。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、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すべてのプラットフォームで機能します。

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。

表 1: コマンド ラインの引数

コマンド オプション	引数	説明
-f	filename	出力ファイル名を指定します。 **例:** **utility** のすべてのレポートの出力ファイルとして utilrept を指定するには、次のコマンドを実行します。 utility -f utilrept -d -xs 詳細については、「filename」を参照してください。

コマンド オプション	引数	説明
-D		analyze、scan、および parm ファイルの確認で詳細を有効にします。 詳細については、「detail」コマンドを参照してください。
-d		analyze および parm のファイル チェックの詳細を無効にします。 詳細については、「detail」コマンドを参照してください。
-T		簡易出力レポート フォーマットを生成します。
-v		コマンドをコマンド ラインで実行するとおりに表示します。
-xp	parmfile	parm ファイルの構文をチェックします。 **例:** parm ファイルの構文を確認するには、次のコマンドを実行します。 `utility -xp` 詳細については、「parmfile」コマンドを参照してください。
-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。 **例:** 詳細については、「checkdef」コマンドを参照してください。
-xa		データストアに保存されたデータを alarmdef ファイルと比較して、分析します。 **例:** アラームのイベントと要約を参照するには、次のコマンドを実行します。 `utility -xa -D` 詳細については、「analyze」コマンドを参照してください。

コマンド オプション	引数	説明
-xs	datastore	データストアをスキャンし、レポートを生成します。 **例:** データをスキャンするには、次のコマンドを実行します。 utility -D -xs 詳細については、「scan」コマンドを参照してください。
-? または ?		コマンド ライン構文を表示します。

utility 走査レポート

utility プログラムの scan コマンドは、データストアを読み込み、その内容に関するレポートを作成します。utility 走査レポートは、データの各クラスが使用するディスク容量に関する情報を提供します。

例:

utility -xs コマンドを実行すると、以下の図に示すように、utility 走査レポートまたはクラス要約レポートが生成されます。

```
                    # utility -xs
      CLASSNAME                          RECORDS     STARTTIME           ENDTIME           HH:MM:SS
-----------------------------------------------------------------------------------------------------
SCOPE::CORE                                    0 0000/00/00 00:00:00  0000/00/00 00:00:00  00:00:00
SCOPE::FILESYSTEM                         209150 2015/07/13 15:13:43  2015/07/21 16:01:00  192:47:17
SCOPE::CPU                                 41830 2015/07/13 15:13:43  2015/07/21 16:01:00  192:47:17
SCOPE::GLOBAL                              20915 2015/07/13 15:13:43  2015/07/21 16:01:00  192:47:17
SCOPE::APPLICATION                         45116 2015/07/13 15:13:43  2015/07/21 16:01:00  192:47:17
SCOPE::PROCESS                            106857 2015/07/13 15:13:43  2015/07/21 16:01:00  192:47:17
SCOPE::DISK                                44536 2015/07/13 15:13:43  2015/07/21 16:01:00  192:47:17
SCOPE::NETIF                               41830 2015/07/13 15:13:43  2015/07/21 16:01:00  192:47:17
SCOPE::LVOLUME                                 0 0000/00/00 00:00:00  0000/00/00 00:00:00  00:00:00
SCOPE::TRANSACTION                             0 0000/00/00 00:00:00  0000/00/00 00:00:00  00:00:00
SISPI::LOGINS                               1690 2015/07/14 16:00:02  2015/07/21 16:00:02  168:00:00
SISPI::GLOBAL                                169 2015/07/14 16:00:03  2015/07/21 16:00:03  168:00:00
```

utility -xs -D コマンドを実行すると、以下の出力が表示されます。

走査レポート (バージョン 11.xx と 12.00) 間の比較については、「HP Operations Agent 12.00 と以前のバージョンの比較」の章の「Utility 走査レポート」に示します。

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詳細に説明します。

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に基づいてデータストア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ことができます。

コマンド ライン モードのアラーム定義ファイルは、**/var/opt/perf/alarmdef** です。

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。

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。

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。 コマンド ライン モードでは、detail off がデフォルトです。アラームのイベントと要約を参照するには、detail on (-D) コマンドを指定します。

analyze コマンドを実行中に、alarm start、end、repeat status などのアラーム イベントがリストされます。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、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。

例:

Performance Alarm Summary レポートを表示するには、次のコマンドを実行します。

```
utility -xa
```

以下の出力が表示されます。

```
Performance Alarm Summary:

Alarm   Count   Minutes
    1       3        25
    3       4        20
    4       16      125

Analysis coverage using "/var/opt/perf/alarmdef":

Start:05/05/2014 00:00:00    Stop:05/05/2014 16:55:00

Total time analyzed:Days:0  Hours:16  Minutes:55
```

アラームのイベントと要約を参照するには、次のコマンドを実行します。

```
utility -xa -D
```

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。

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 -xc
```

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データストア内のデータと比較して、アラーム定義を処理できます。

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、コマンド ライン モードでは detail off です。

パラメータ

detail	on	-D	**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	-d	**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、collector 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。

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。

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。

例:

Utility プログラムのコマンド ラインの引数を表示するには、次のコマンドを実行します。

```
utility -?
```

filename

filename 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、filename 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。

例:

```
utility -f <file name> -d -xs
```

utility のすべてのレポートの出力ファイルとして utilrept を指定するには、次のコマンドを実行します。

```
utility -f utilrept -d -xs
```

filename コマンドの <file name> 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す必要があります。出力は既存のファイルの最後に追加されます。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 filename コマンドを実行します。

出力ファイルが standard output 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表示されます。

> **注:** AIX システムの場合、utility -xs コマンドで -f オプションを使用して、出力ファイルを指定することはできません。

parmfile

parmfile コマンドは、データ収集に使用する**parm** ファイルの設定値の構文を表示および確認するときに使用します。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。

例

parm ファイルの構文を確認するには、次のコマンドを実行します。

```
utility -xp
```

以下の出力が表示されます。

```
33 file  names used to define applications
1 user  names used to define applications
0 group names used to define applications

Parm File:"/var/opt/perf/parm" had 0 Warnings.
```

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。

記録しているパラメータの設定のリストを表示するには、次のコマンドを実行します。

```
utility -xp -v
```

scan

scan コマンドは、データストアを読み取り、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。

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。

詳細については、「detail」および「list」を参照してください。

例

データをスキャンするには、次のコマンドを実行します。

```
utility -D -xs
```

第7章: Extract プログラムの使い方

Extract プログラムは、HP Operations Agent のデータストアに記録された履歴データの取得と分析に使用します。Extract プログラムはエクスポート機能を実行します。このプログラムは、データストアからデータを読み取り、結果を ASCII 形式の出力ファイルにエクスポートします。

注: oacore プロセスは、extract プログラムが機能するために動作している必要があります。

コマンド ライン インターフェイスを使用した Extract プログラムの実行

HP Operations Agent 12.00 は、Extract プログラムを実行するためにコマンド ライン モードのみをサポートしています。データをデータストアから取得するには、コマンド ライン モードで Extract プログラムを実行します。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、一般的な UNIX 環境に適合します。

注: HP Operations Agent バージョン 11.xx から 12.00 へのアップグレードの際に、CODA データベース ファイル、scope ログ ファイル、DSI ログ ファイルに記録されている古いデータは、読み取り専用モードで保持されます。Extract プログラムは、古いデータストアと新しいデータストアの両方からデータを読み取ることができます。

次の表は、コマンド ライン オプションと引数の一覧です。

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	export 関数の開始日時を指定します。 構文: mm/dd/yyyy hh:mm:ss
-B		UNIXstart time	export 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	export 関数の終了日時を指定します。 構文: mm/dd/yyyy hh:mm:ss
-E		UNIX 終了時刻	export 関数の UNIX フォーマットでの終了時刻を指定します。

表 3: 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
-l	logfile		DSI データの入力ログ ファイルを指定します。
-C	classname		自己記述 (DSI) データを export に指定します。
gapcdznituyhx GADZNITUYHX			エクスポートするデータのタイプを指定します。 g = グローバル詳細 a = アプリケーション詳細 p = プロセス詳細 c = 設定詳細 d = ディスク デバイス詳細 z = lvolume 詳細 n = netif 詳細 i = 論理システム詳細 **注:** 論理システム詳細は、Windows および Linux プラットフォームではサポートされていません。 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 h = Host Bus Adapter (HBA) 詳細 x = コア詳細 G = グローバル概要 A = アプリケーション概要 D = ディスク デバイス概要 Z = lvolume 概要

表 3: 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
			N = netif 概要 I = 論理システム概要 **注:** 論理システム概要は、Windows および Linux プラットフォームではサポートされていません。 T= トランザクション概要 U = CPU 概要 Y = ファイルシステム概要 H = HBA 概要 X = コア概要
-r	export template file		export 関数で使用するエクスポート テンプレート ファイルを指定します。 **注:** エクスポート テンプレート ファイルのパスにスペースを含める場合は、必ず二重引用符で囲むようにします。
-f	filename		出力ファイル名を指定します。
-xp	xopt		データを ASCII フォーマット ファイルにエクスポートします。
-? または ?			コマンド ライン構文を表示します。

Export 関数の使用

export コマンドはデータベースに記録されているデータを読み込み、結果を出力ファイルにエクスポートします。出力ファイルは、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。

データをエクスポートする方法

以下の手順に従ってください。

1. クラスのタイプまたはデータのタイプを指定します。「デフォルトのパフォーマンス メトリック クラス」(171ページ)を参照してください。

   例: -a (アプリケーション クラス)、-g (グローバル クラス)

2. データをエクスポートする出力ファイルを指定します。

   注:

   出力ファイルの名前を指定しないと、エクスポートしたデータはデフォルトの出力ファイルに書き込まれます。デフォルトの出力ファイルは、エクスポートされたデータのクラスによって異なります。

   例: 詳細グローバル クラス データをエクスポートするには、デフォルトの出力ファイル xfrdGLOBAL.asc が使用されます。「表 5: デフォルトの出力ファイル」(173ページ)を参照してください。

3. export コマンドを実行します。
4. エクスポートされたデータが表示されます。「エクスポート ファイルの出力」(177ページ)を参照してください。

注: Performance Collection Component には、エクスポート テンプレート ファイルとして reptfile 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。Reptfile には、コメントから外される定義済みのメトリック セットが含まれています。このファイルは、デフォルトで共通のエクスポート タスクを実行するのに使用されます。reptall ファイルでは、すべてのメトリックがコメント アウトされ、特定のメトリックをエクスポートするのに使用されます。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には、「カスタマイズした出力ファイルの作成」を参照してください。

例:

2014 年 7 月 26 日から収集された CPU 詳細データをエクスポートするには、以下のコマンドを入力します。

```
extract -u -b 7/26/14 -xp
```

ここで、

-u	CPU の詳細を指定します。
-b	export 関数の開始日を指定します。
-xp	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。

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。

デフォルトのパフォーマンス メトリック クラス

次のデータ クラスをエクスポートできます。

表 4: クラス タイプ

クラス タイプ	要約間隔
global	5、15、30、60、180、360、720、1440、10080 分。
application	5、15、30、60、180、360、720、1440、10080 分。
process	このメトリック クラスは要約されません。
disk device	5、15、30、60、180、360、720、1440、10080 分。
lvolume	5、15、30、60、180、360、720、1440、10080 分。
transaction	5、15、30、60、180、360、720、1440、10080 分。
configuration	5、15、30、60、180、360、720、1440、10080 分。
netif	5、15、30、60、180、360、720、1440、10080 分。
cpu	5、15、30、60、180、360、720、1440、10080 分。
filesystem	5、15、30、60、180、360、720、1440、10080 分。
host bus adapter	5、15、30、60、180、360、720、1440、10080 分。
core	5、15、30、60、180、360、720、1440、10080 分。

注: 表に示されているもの以外の要約間隔を記述した場合、extract プログラムは最も近い要約間隔に要約します。

出力ファイル

データをエクスポートするのにデフォルトの出力ファイルを使用するか、または出力ファイルを指定してから、export コマンドを発行することができます。

- export コマンドを発行する前に出力ファイルを指定した場合は、すべてのデータはこの 1 つのファイルにエクスポートされます。

  例:

  詳細グローバル データを myout という名前の出力ファイルにエクスポートするには、次のコマンドを実行します。

  ```
  extract -g -f myout -xp
  ```

  ここで、

-g	グローバルの詳細を指定します。
-f	出力ファイル名を指定します。

myout	出力ファイルの名前です。
-xp	データをエクスポートします。

- 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、エクスポートしたデータがさまざまなデフォルトの出力ファイルに分割されます。

  例:

  ```
  extract -xp -g
  ```

  export コマンドにより、データのグローバル クラスが xfrdGLOBAL.asc ファイルにエクスポートされます。

次の表は、デフォルトのすべての出力ファイルの一覧です。

表 5: デフォルトの出力ファイル

デフォルトの出力ファイル	説明
xfrdGLOBAL.asc	グローバル詳細データ ファイル
xfrsGLOBAL.asc	1 時間ごとのグローバル要約データ ファイル
xfrdAPPLICATION.asc	アプリケーション詳細データ ファイル
xfrsAPPLICATION.asc	1 時間ごとのアプリケーション要約データ ファイル
xfrdPROCESS.asc	プロセス詳細データ ファイル
xfrdDISK.asc	ディスク デバイス詳細データ ファイル
xfrsDISK.asc	1 時間ごとのディスク デバイス要約データ ファイル
xfrdVOLUME.asc	論理ボリューム詳細データ ファイル
xfrsVOLUME.asc	論理ボリューム要約データ ファイル
xfrdNETIF.asc	netif 詳細データ ファイル
xfrsNETIF.asc	netif 要約データ ファイル
xfrdCPU.asc	CPU 詳細データ ファイル
xfrsCPU.asc	CPU 要約データ ファイル
xfrdFILESYSTEM.asc	filesystem 詳細データ ファイル
xfrsFILESYSTEM.asc	filesystem 要約データ ファイル
xfrdTRANSACTION.asc	トランザクション詳細データ ファイル
xfrsTRANSACTION.asc	トランザクション要約データ ファイル

表 5: デフォルトの出力ファイル (続き)

xfrdCONFIGURATION.asc	構成データ ファイル
xfrdHBA.asc	Host Bus Adapter (HBA) 詳細データ ファイル
xfrsHBA.asc	HBA 要約データ ファイル
xfrdCORE.asc	コア詳細データ ファイル
xfrsCORE.asc	コア要約データ ファイル

ここで、ext は asc (ASCII) です。

注: 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。

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。データによって、接頭辞が xfrd または xfrs になります。xfrd は詳細データに使用され、xfrs は要約データに使用されます。

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.asc	ACCT_INFO の詳細 ASCII データ
xfrsACCTG_INFO.asc	ACCT_INFO の要約 ASCII データ

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。

```
HEADINGS    [ON]
            [OFF]
SEPARATOR= "|"
SUMMARY=VALUE
OUTPUT=Filename
DATA TYPE DATATYPE
METRICS
```

例:

```
HEADINGS ON
SEPARATOR="|"
SUMMARY=60

**********************************************************************
**
** The rest of the file consists of specifics about the metrics to be
** exported for a single data type. Each exportable class may have one
** set of definitions in which the following lines appear.
**
** DATA TYPE specifies the type (class) of data. (required)
**
** OUTPUT    specifies the name of the output file where the exported data is
**           to be written.  OUTPUT may be specified for each class named.
**           (optional)
**
**
**  Individual metric names which belong to the class. Metrics will be exported
**  in the order listed whenever possible. In order to be exported, metrics
**  must not be commented out (that is, they must not contain an asterisk before
**  the metric name).
**
** This sample report was generated with a section for each data type which
** is available in the currently open log file. Following the DATA TYPE
** line, each metric which is available for export for this data type is
** listed, but commented out. To select metrics for your report, delete
** the asterisk (*) in the first column.
**
**********************************************************************
**********************************************************************

**********************************************************************
DATA TYPE GLOBAL

**............................... Global Record Identification Metrics

* GBL_PROC_SAMPLE
* GBL_SYSTEM_UPTIME_HOURS
* GBL_SYSTEM_UPTIME_SECONDS
* GBL_STATTIME
* GBL_INTERVAL
* GBL_CSWITCH_RATE
* GBL_INTERRUPT_RATE
* GBL_INTERRUPT
```

パラメータ

headings	export 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。 headings on を指定すると、ASCII 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。
separator	ASCII フォーマットのデータの各フィールド間に出力される文字を指定します。 デフォルトの区切り記号は縦棒です。多くのプログラムではフィールドの区切り記号としてカ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	各要約間隔に対して要約されるデータの分数を指定します。 デフォルトの間隔は、5、15、30、60、180、360、720、1440、10080 分です。他の要約間隔を記述した場合、extract プログラムは最も近い要約間隔に要約します。
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出力ファイル名を置くことで、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。
data type	エクスポート可能なデータ型 (global、application、process、disk、transaction、lvolume、netif、configuration、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。
metrics	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。各データ型は、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。

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。

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。

名前 (application、netif、lvolume、transaction のいずれか)	headings on を指定した場合
ヘッダー行 1	headings on を指定した場合
ヘッダー行 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。

例:

グローバル クラス データをエクスポートすると、次の出力が表示されます。

```
                Time|          |    System|      User|      Nice|      Idle|      Wait|          Phys|      Phys|      Phys|         |
               Stamp|      CPU%|      CPU%|      CPU%|      CPU%|      CPU%|      CPU%|           IOs|        Wr|        KB|    Disk%|
   09/02/14 12:08:00|    100.00|     33.33|     66.66|      0.00|      0.00|      0.00|             0|      0.00|      0.00|     0.00|
   09/02/14 12:13:00|      6.79|      3.06|      3.72|      0.00|     93.19|      0.00|           320|      1.00|      9.90|     0.01|
   09/02/14 12:18:00|      6.93|      3.05|      3.87|      0.00|     93.06|      0.00|           262|      0.80|      7.60|     0.00|
   09/02/14 12:23:00|      6.97|      3.21|      3.75|      0.00|     93.02|      0.00|           237|      0.70|      6.90|     0.00|
   09/02/14 12:28:00|      7.03|      3.13|      3.89|      0.00|     92.95|      0.00|           231|      0.70|      6.60|     0.00|
   09/02/14 12:33:00|      6.94|      3.07|      3.87|      0.00|     93.04|      0.00|           246|      0.80|      7.10|     0.00|
   09/02/14 12:38:00|      6.83|      2.89|      3.93|      0.00|     93.15|      0.00|           229|      0.70|      6.60|     0.00|
   09/02/14 12:43:00|      6.69|      2.63|      4.06|      0.00|     93.29|      0.00|           248|      0.80|      7.10|     0.00|
   09/02/14 12:48:00|      6.62|      2.57|      4.05|      0.00|     93.36|      0.00|           235|      0.70|      6.80|     0.00|
   09/02/14 12:53:00|      6.66|      2.91|      3.74|      0.00|     93.33|      0.00|           248|      0.80|      7.00|     0.00|
   09/02/14 12:58:00|      6.59|      2.96|      3.62|      0.00|     93.39|      0.00|           245|      0.80|      6.70|     0.00|
   09/02/14 13:03:00|      6.94|      2.95|      3.98|      0.00|     93.04|      0.00|           266|      0.80|      7.40|     0.00|
   09/02/14 13:08:00|      6.92|      3.14|      3.77|      0.00|     93.07|      0.00|           252|      0.80|      6.80|     0.00|
   09/02/14 13:13:00|      7.03|      3.13|      3.89|      0.00|     92.95|      0.00|           259|      0.80|      7.10|     0.00|
   09/02/14 13:18:00|      6.99|      2.97|      4.01|      0.00|     93.00|      0.00|           267|      0.80|      7.20|     0.00|
   09/02/14 13:23:00|      6.99|      3.09|      3.89|      0.00|     93.00|      0.00|           259|      0.80|      7.30|     0.00|
   09/02/14 13:28:00|      6.79|      2.71|      4.07|      0.00|     93.19|      0.00|           263|      0.80|      7.10|     0.00|
   09/02/14 13:33:00|      6.64|      2.61|      4.02|      0.00|     93.35|      0.00|           272|      0.90|      7.50|     0.00|
   09/02/14 13:38:00|      6.58|      2.61|      3.97|      0.00|     93.40|      0.00|           259|      0.80|      6.80|     0.00|
   09/02/14 13:43:00|      6.76|      2.95|      3.80|      0.00|     93.22|      0.06|           360|      0.90|     14.30|     0.07|
   09/02/14 13:48:00|      6.93|      3.02|      3.90|      0.00|     93.06|      0.00|           275|      0.90|      7.30|     0.00|
   09/02/14 13:53:00|      7.01|      3.04|      3.96|      0.00|     92.98|      0.00|           272|      0.90|      7.40|     0.00|
   09/02/14 13:58:00|      7.04|      3.23|      3.80|      0.00|     92.94|      0.00|           254|      0.80|      6.80|     0.00|
   09/02/14 14:03:00|      7.04|      3.07|      3.97|      0.00|     92.94|      0.00|           278|      0.90|      7.70|     0.00|
   09/02/14 14:08:00|      7.08|      2.99|      4.09|      0.00|     92.90|      0.00|           248|      0.80|      6.70|     0.00|
   09/02/14 14:13:00|      6.99|      2.79|      4.19|      0.00|     92.99|      0.00|           278|      0.90|      7.40|     0.00|
   09/02/14 14:18:00|      7.06|      2.90|      4.15|      0.00|     92.92|      0.00|           274|      0.90|      7.30|     0.00|
   09/02/14 14:23:00|      6.64|      2.64|      3.99|      0.00|     93.34|      0.00|           271|      0.90|      7.40|     0.00|
   09/02/14 14:28:00|      6.76|      2.82|      3.93|      0.00|     93.23|      0.00|           261|      0.80|      6.90|     0.00|
   09/02/14 14:33:00|      6.92|      2.88|      4.04|      0.00|     93.06|      0.00|           271|      0.90|      7.40|     0.00|
   09/02/14 14:38:00|      7.18|      3.09|      4.08|      0.00|     92.80|      0.00|           264|      0.80|      7.10|     0.00|
   09/02/14 14:43:00|      7.08|      3.16|      3.91|      0.00|     92.91|      0.00|           265|      0.80|      7.30|     0.01|
   09/02/14 14:48:00|      7.18|      3.07|      4.10|      0.00|     92.81|      0.00|           266|      0.80|      7.20|     0.00|
   09/02/14 14:53:00|      7.14|      3.12|      4.01|      0.00|     92.85|      0.00|           275|      0.90|      7.40|     0.00|
   09/02/14 14:58:00|      7.06|      2.98|      4.07|      0.00|     92.93|      0.00|           261|      0.80|      6.90|     0.00|
   09/02/14 15:03:00|      7.18|      3.00|      4.17|      0.00|     92.81|      0.00|           287|      0.90|      7.70|     0.00|
   09/02/14 15:08:00|      6.73|      2.67|      4.06|      0.00|     93.25|      0.00|           268|      0.80|      7.00|     0.00|
   09/02/14 15:13:00|      6.74|      2.77|      3.97|      0.00|     93.25|      0.00|           279|      0.90|      7.40|     0.00|
   09/02/14 15:18:00|      6.83|      2.91|      3.92|      0.00|     93.15|      0.00|           278|      0.90|      7.40|     0.00|
   09/02/14 15:23:00|      7.21|      2.99|      4.21|      0.00|     92.78|      0.00|           277|      0.90|      7.30|     0.00|
   09/02/14 15:28:00|      7.05|      3.03|      4.01|      0.00|     92.93|      0.00|           264|      0.80|      6.90|     0.00|
   09/02/14 15:33:00|      7.17|      3.10|      4.06|      0.00|     92.82|      0.00|           283|      0.90|      7.60|     0.00|
   09/02/14 15:38:00|      7.17|      3.05|      4.12|      0.00|     92.81|      0.00|           258|      0.80|      6.80|     0.00|
   09/02/14 15:43:00|      7.16|      3.13|      4.03|      0.00|     92.83|      0.00|           280|      0.90|      7.40|     0.00|
'xfrdGLOBAL.asc" 327L, 44989C
```

注:

reptfile または reptall で **gbl_statdate** と **gbl_stattime** メトリックを選択していなくても、エクスポート ファイルには常に 24 時間制でタイム スタンプが含まれます。

HP Operations Agent 12.00 では、Extract 出力に表示されるメトリック ヘッダーは、perfd や Glance などのリアルタイム ツールによって表示されるメトリック ヘッダーと一致します。

カスタマイズした出力ファイルの作成

Performance Collection Component には、エクスポート テンプレート ファイル `reptall` が用意されています。このファイルは `/var/opt/perf/` ディレクトリにあります。`reptall` を使用すると、以下のように出力ファイルをカスタマイズできます。

エクスポート ファイルをカスタマイズする:

エクスポート ファイルをカスタマイズするには、以下の手順に従います。

1. エクスポート テンプレート ファイルで、エクスポートするメトリックをコメントから外します。
2. ファイルを保存してエクスポートします。

ASCII フォーマットに関する注意

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。`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揃っています。

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カンマを含むデータ項目もあります。このようなカンマにより、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。

注: 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。

ヒント: 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。

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例を挙げて説明します。

表 6: extract のコマンド

コマンド オプション	説明
application	application オプションは、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。 **たとえば、次のようになります。** 詳細なアプリケーション データをエクスポートするには、以下のコマンドを実行します。 `extract -a -r /var/opt/perf/myrept -xp` 出力ファイルが指定されていないため、アプリケーション データは xfrdAPPLICATION.asc にエクスポートされます。 出力ファイルには、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すべてのアプリケーション メトリックが含まれます。
cpu	cpu オプションは、CPU の要約レベルを指定するときに使用します。 **たとえば、次のようになります。** 2014 年 7 月 26 日から収集された CPU 詳細データをエクスポートするには、以下のコマンドを実行します。 `extract -u -b 7/26/14 -xp` この例では、出力ファイルが指定されていないため、CPU データは xfrdCPU.asc に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 で指定されているすべての CPU メトリックが出力ファイルに含まれます。

表 6: extract のコマンド (続き)

disk	disk オプションは、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。 **たとえば、次のようになります。** 2014 年 7 月 5 日から収集されたディスク詳細データをエクスポートするには、以下のコマンドを実行します。 `extract -d -b 7/5/14 -xp` この例では、出力ファイルが指定されていないため、ディスク詳細データは xfrdDISK.asc に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。
export	export オプションは、出力ファイル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。 **たとえば、次のようになります。** グローバルな詳細データをエクスポートするには、以下のコマンドを実行します。 `extract -xp -g` データのグローバル クラスが xfrdGLOBAL.asc ファイルにエクスポートされます。
filesystem	filesystem オプションは、エクスポートするファイルシステム データの要約レベルを指定するときに使用します。 **たとえば、次のようになります。** 2014 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をエクスポートするには、以下のコマンドを実行します。 `extract -y -b 7/26/14 -xp` ファイルシステム データが xfrdFILESYSTEM.asc に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 で指定されているすべてのファイルシステム メトリックが出力ファイルに含まれます。

表 6: extract のコマンド (続き)

global	`global` オプションは、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。 **たとえば、次のようになります。** 詳細なグローバル データをエクスポートするには、以下のコマンドを実行します。 `extract -g -r /var/opt/perf/myrept -f myout -xp` グローバル データが `myout` という名前の出力ファイルにエクスポートされます。 出力ファイルには、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すべてのグローバル メトリックが含まれます。
help	`help` オプションは、オンライン ヘルプにアクセスするときに使用します。
lvolume	`lvolume` オプションは、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。 **メモ:** このオプションは、HP-UX および Linux システム上でのみサポートされています。 **たとえば、次のようになります。** 詳細な論理ボリューム データをエクスポートするには、以下のコマンドを実行します。 `extract -z -r /var/opt/perf/myrept -xp` 出力ファイルが指定されていないため、論理ボリューム データは `xfrdVOLUME.asc` にエクスポートされます。 出力ファイルには、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メトリックが含まれます。

表 6: extract のコマンド (続き)

netif	netif オプションは、エクスポート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。 **たとえば、次のようになります。** 詳細なネットワーク インターフェイス データをエクスポートするには、以下のコマンドを実行します。 `extract -n -r /var/opt/perf/myrept -xp` 出力ファイルが指定されていないため、ネットワーク インターフェイス データは xfrdNETIF.asc にエクスポートされます。 出力ファイルには、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メトリックが含まれます。
process	process オプションは、プロセス データ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。 **たとえば、次のようになります。** 詳細なプロセス データをエクスポートするには、以下のコマンドを実行します。 `extract -p -r /var/opt/perf/myrept -xp` 出力ファイルが指定されていないため、プロセス データは xfrdPROCESS.asc にエクスポートされます。 出力ファイルには、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すべてのプロセス メトリックが含まれます。
start	start オプションは、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。 First および last オプションは、start (-b) または stop (-e) コマンドと共に使用するときにサポートされます。 **たとえば、次のようになります。** 1. 1999 年 6 月 5 日 8:00 am から開始される詳細なグローバル データをエクスポートするには、以下のコマンドを実行します。 `extract -g -b 06/05/99 8:00 -f myout -xp` グローバル データが myout という名前の出力ファイルに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 で指定されているすべてのグローバル メトリックが出

表 6: extract のコマンド (続き)

	力ファイルに含まれます。
	2. 最後のグローバル データをエクスポートするには、以下のコマンドを実行します。
	`extract -g -b last-f myout -xp`
	3. 最初と最後のグローバル データを `xfrdGLOBAL.asc` ファイルにエクスポートするには、以下のコマンドを実行します。
	`extract -g -b first -e last -xp`

stop	`stop` オプションは、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。 **たとえば、次のようになります。** 2014 年 6 月 5 日 8:00 am から 2014 年 6 月 5 日 5:00 pm までの詳細なグローバル データをエクスポートするには、以下のコマンドを実行します。 `extract -g -b 6/5/14 8:00 -e 6/5/14 17:00  -f myout -xp` グローバル データが `myout` という名前の出力ファイルに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。
transaction	`transaction` オプションは、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。 **たとえば、次のようになります。** 詳細なトランザクション データをエクスポートするには、以下のコマンドを実行します。 `extract -t -r /var/opt/perf/myrept -xp` 出力ファイルが指定されていないため、トランザクション データは `xfrdTRANSACTION.asc` にエクスポートされます。 出力ファイルには、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すべてのトランザクション メトリックが含まれます。
host bus adapter	`host bus adapter` オプションは、エクスポートする HBA データの量を指定するときに使用します。 **たとえば、次のようになります。** 2014 年 7 月 26 日から収集された HBA 詳細データをエクスポートするに

表 6: extract のコマンド (続き)

	は、以下のコマンドを実行します。 `extract -h -b 7/26/14 -xp` この例では、出力ファイルが指定されていないため、HBA データは `xfrdHBA.asc` にエクスポートされます。 **注:** • デフォルトのエクスポート テンプレート ファイル reptfile には、HBA メトリックは含まれません。HBA データをエクスポートするには、手動で HBA メトリックを追加します。 • このコマンドは、HP-UX システム上でのみ使用されます。
classname and logfile	`classname` オプションは、エクスポートする DSI データのクラスを指定するときに使用します。 **構文** extract -xp -C <dsi class name> -l <log file path>
core	`core` オプションは、エクスポートする bycore データの種類を指定するときに使用します。 **メモ:** このオプションは、Linux、Windows、および Solaris の各システム上でのみサポートされています。 **たとえば、次のようになります。** 詳細なアプリケーション データをエクスポートするには、以下のコマンドを実行します。 `extract -x -r /var/opt/perf/myrept -xp` 出力ファイルには、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すべてのアプリケーション メトリックが含まれます。

第8章: 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 Agent 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 <system_name>
   ```

   この <system_name> 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_address>
   ```

   この <ip_address> には、リモート システムの IP アドレスを指定します。

   > **メモ:** 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 ユーティリティによって、ローカル システムのパフォーマンス データに他のシステムがアクセスしないようにできます。詳細については、「アクセスの制限」(47ページ)を参照してください。

   cpsh プロンプトが表示されます。

   見やすい形式でメトリック 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。

   たとえば、次のようになります。

   ```
   cpsh -t
   ```

   または

   ```
   cpsh -n <system_name> -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** <metric_class> と入力します。
2. そのクラスに現在設定されているすべての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** <metric_class> 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。

   cpsh プロンプトに **delete** <metric_name> と入力します。

   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。

   cpsh プロンプトに **add** <metric_name> と入力します。

   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** <metric_class> 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** <metric_class> と入力します。
3. **init** <metric_name> <metric_name> <metric_name> ... 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、各リアルタイム 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 <metric_class> と入力します。
3. cpsh プロンプトに **help** <metric_name> 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** <interval> と入力します。この <interval> には、要約の間隔を秒単位で指定します。<interval> 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)

しきい値とフィルタ条件の有効化

Perfd プロセスの要件を基にしきい値とフィルタ オプションを設定して、設定した適合基準で提供されるデータを表示できます。

以下のオプションを設定します。

Threshold: このオプションは GLOBAL クラスのメトリックに対してのみ使用可能です。グローバルしきい値を指定すると、設定したグローバルしきい値を超えるデータを表示できます。

Filter: フィルタ オプションは、すべてのクラスのメトリックに対して使用できます。

以下のように条件を設定できます。

- **filter** <metric> <operator> <value>
- **threshold** <metric> <operator> <value>

両方の条件を設定して、設定した条件を満たすデータのみを表示することもできます。

例 1: 使用率が 4% を超えるプロセス数を一覧します。

フィルタ条件を使用してデータを表示します。

フィルタ処理されたデータを表示するには、次の手順に従います。

1. cpsh プロンプトを表示します。
2. cpsh プロンプトに **class proc** と入力します。
3. **filter proc_cpu_total_util > 4** と入力します。

   cpsh ユーティリティは、CPU 使用率が 4% を超えるプロセスを表示します。
4. **push** と入力します。

次のように結果が表示されます。

```
User   App     Interest        Process         ProcessName
Name   PID     ID Reason       Start Time      CPU Name
root  1028     4       05/07/2013 07:17:35     5.2 find
```

例 2: 使用率が 4% を超え、しきい値が 1% に設定されたプロセスの数を一覧します。

しきい値オプションとフィルタ オプションを設定して、データを表示します。

フィルタ処理されたデータを表示するには、次の手順に従います。

1. cpsh プロンプトを表示します。
2. cpsh プロンプトに **class proc** と入力します。
3. **threshold proc_cpu_total_util > 1** と入力します。
4. **filter proc_cpu_total_util > 3** と入力します。

   cpsh ユーティリティは、しきい値が1 以上に設定され、CPU 使用率が 3% を超えるプロセスを表

示します。

5. **push** と入力します。

次のように結果が表示されます。

```
User   App     Interest        Process         ProcessName
Name   PID     ID Reason       Start Time      CPU Name
root  1028     4       05/07/2013 07:17:35     4.9 find
```

第9章: ベースライン計算の概要

ベースライン計算とは、履歴データに基づいて基準値を計算して返すプロセスのことです。[1] 特定の期間のベースライン データを計算するには、前の週の対応する期間に収集されたメトリック データが用いられます。ベースライン データには、最小値、最大値、平均値、標準偏差値が含まれます。ベースライン データは、毎時末に計算され、メトリック データストアに保存されます。

ベースライン データは次の用途に用いられます。

- 毎日のパフォーマンスを監視するための基準値の設定
- パフォーマンスの傾向を分析するための基準値の設定
- リソース使用率のパターンを分析するための最適なしきい値の動的な設定

HP Operations Agent によって計算されたベースライン データは、SI-AdaptiveThresholdingMonitor ポリシーによって、パフォーマンスとリソース使用率を監視するために用いられます。詳細については、『HP Operations Smart Plug-in for Systems Infrastructure ユーザー ガイド』の適応しきい値ポリシーについてのトピックを参照してください。

HP Operations Agent ノードでのベースラインの設定

デフォルトでは、ベースラインは HP Operations Agent ノード上で有効ではありません。HP Operations Agent ノードでベースラインを設定するには、以下の手順を実行します。

1. `baseline.cfg` ファイルをアップデートします

   注: `baseline.cfg` は、監視するメトリックを定義するのに使用されるプレーン テキスト ファイルです。

   a. HP Operations Agent ノードにログオンします。

   b. 以下のディレクトリに移動します。

      Windows の場合:

      `%ovdatadir%`

      HP-UX/Linux/Solaris の場合:

      `/var/opt/perf/`

   c. `baseline.cfg` ファイルをアップデートして、監視するメトリックを次の形式で定義します。

      `<Class>:<Metric>`

      ここで、

- `<Class>` はメトリック クラスです。
- `<Metric>` はベースライン データの計算が必要なメトリックです。

注: ベースライン データは、ゲージ メトリックに対してのみ計算されます。

例:

Global:GBL_CPU_TOTAL_UTIL

ここで、

`<Class>`	Global
`<Metric>`	GBL_CPU_TOTAL_UTIL

2. HP Operations Agent ノード上でベースラインを有効にします
   a. 以下のディレクトリに移動します。

      Windows x64 の場合:

      `%ovinstalldir%bin\win64`

      Windows x86 の場合:

      `%ovinstalldir%bin`

      AIX の場合:

      `/usr/lpp/OV/bin`

      HP-UX/Linux/Solaris の場合:

      `/opt/OV/bin`

   b. 次のコマンドを実行します。

      ```
      ovconfchg -ns oacore -set ENABLE_BASELINE TRUE
      ```

      デフォルトでは、この値は FALSE です。

ベースライン データは、毎時末にのみ計算されます。ベースラインを有効にした直後にベースライン データを計算するには、**oacore** プロセスを再起動する必要があります。以下のコマンドを実行して、**oacore** を起動します。

```
ovc -restart oacore
```

注: また、XPL 設定を使用してもベースラインを有効にできます。以下の手順に従ってください。

1. HPOM コンソールで、**[ポリシー管理] → [ポリシー グループ] → [Infrastructure Management] → [v12.0] → [Settings and Thresholds] → [Agent Settings] → [OPC_PERL_INCLUDE_INSTR_DIR]** の順に選択します。
2. 変数 ENABLE_BASELINE を TRUE に設定して、必要なすべてのノードにポリシーを配布します。

詳細については、『HP Operations Smart Plug-in for Systems Infrastructure ユーザー ガイド』を参照してください。

メトリックの監視

注: ベースライン データを計算したら、Measurement Threshold ポリシーを使用して目的のメトリックを監視します。ベースライン メトリックを監視するには、SI-ConfigureBaselining ポリシーや SI-AdaptivethresholdingMonitor ポリシーなどのデフォルトのインフラストラクチャ SPI ポリシーを使用できます。詳細については、『HP Operations Smart Plug-in for Systems Infrastructure ユーザー ガイド』を参照してください。

`baseline.cfg` ファイルに定義したクラスに基づいて、対応するベースライン クラスが メトリック データストア 内に作成されます。

注: ベースライン クラスの末尾には _BASELINE が付いています。

例:

`baseline.cfg` ファイルに次のメトリックを指定すると

`Disk:BYDSK_UTIL`

`Global:GBL_CPU_TOTAL_UTIL`

次の 2 つのベースライン クラスが作成されます。

`DISK_BASELINE`

`GLOBAL_BASELINE`

ベースライン クラスについての詳細を表示するには、`utility -xs` コマンドを実行します。

```
# utility -xs

CLASSNAME              RECORDS    STARTTIME          ENDTIME            HH:MM:SS
-----------------------------------------------------------------------------------
SCOPE::DISK              295      2015/07/31 12:30:40  2015/07/31 14:51:00   2:20:20
SCOPE::NETIF             426      2015/07/31 12:30:40  2015/07/31 14:51:00   2:20:20
SCOPE::TRANSACTION       994      2015/07/31 12:30:40  2015/07/31 14:51:00   2:20:20
SCOPE::PROCESS         60262      2015/07/31 12:30:40  2015/07/31 14:52:00   2:21:20
SCOPE::CPU               284      2015/07/31 12:30:40  2015/07/31 14:51:00   2:20:20
SCOPE::CORE                0      0000/00/00 00:00:00  0000/00/00 00:00:00   00:00:00
SCOPE::APPLICATION       579      2015/07/31 12:30:40  2015/07/31 14:51:00   2:20:20
SCOPE::FILESYSTEM       4118      2015/07/31 12:30:40  2015/07/31 14:51:00   2:20:20
SCOPE::GLOBAL            142      2015/07/31 12:30:40  2015/07/31 14:51:00   2:20:20
SCOPE::LVOLUME             0      0000/00/00 00:00:00  0000/00/00 00:00:00   00:00:00
SCOPE::DISK_BASELINE       6      2015/07/31 13:00:00  2015/07/31 14:00:00   1:00:00
SCOPE::GLOBAL_BASELINE     2      2015/07/31 13:00:00  2015/07/31 14:00:00   1:00:00
```

ベースライン クラス

`baseline.cfg` ファイルに指定したすべてのメトリックに対して、16 個のベースライン メトリックが作成されます。

例:

baseline.cfg ファイルに GBL_CPU_TOTAL_UTIL を指定すると、16 個のグローバル ベースライン メトリックが作成されます。

注: ベースライン メトリックを表示するには、ovcodautil-obj コマンドを実行します。

GLOBAL_BASELINE クラスに対して作成したベースライン メトリックを表示するには、次のコマンドを実行します。

```
ovcodautil -ds SCOPE -o GLOBAL_BASELINE -obj
```

```
GLOBAL_BASELINE        ATT   I64   INSTANCEID
GLOBAL_BASELINE        KEY   UTF8  GBL_SYSTEM_ID
GLOBAL_BASELINE        GGE   R64   GBL_CPU_TOTAL_UTIL_MAX_BL
GLOBAL_BASELINE        GGE   R64   GBL_CPU_TOTAL_UTIL_MIN_BL
GLOBAL_BASELINE        GGE   R64   GBL_CPU_TOTAL_UTIL_AVG_BL
GLOBAL_BASELINE        GGE   R64   GBL_CPU_TOTAL_UTIL_STDDEV_BL
GLOBAL_BASELINE        GGE   R64   GBL_CPU_TOTAL_UTIL_STDDEV_BL_LASTHOUR
GLOBAL_BASELINE        GGE   I64   GBL_CPU_TOTAL_UTIL_BL_SAMPLESIZE
GLOBAL_BASELINE        GGE   R64   GBL_CPU_TOTAL_UTIL_MAX_BL_LASTHOUR
GLOBAL_BASELINE        GGE   R64   GBL_CPU_TOTAL_UTIL_MIN_BL_LASTHOUR
GLOBAL_BASELINE        GGE   R64   GBL_CPU_TOTAL_UTIL_AVG_BL_LASTHOUR
GLOBAL_BASELINE        GGE   I64   GBL_CPU_TOTAL_UTIL_BL_SAMPLESIZE_LASTHOUR
GLOBAL_BASELINE        GGE   I64   GBL_CPU_TOTAL_UTIL_S0
GLOBAL_BASELINE        GGE   I64   GBL_CPU_TOTAL_UTIL_S0_LASTHOUR
GLOBAL_BASELINE        GGE   R64   GBL_CPU_TOTAL_UTIL_S1
GLOBAL_BASELINE        GGE   R64   GBL_CPU_TOTAL_UTIL_S1_LASTHOUR
GLOBAL_BASELINE        GGE   R64   GBL_CPU_TOTAL_UTIL_S2
GLOBAL_BASELINE        GGE   R64   GBL_CPU_TOTAL_UTIL_S2_LASTHOUR
```

指定可能な要約のタイプには、平均値 (Avg)、最小値 (Min)、最大値 (Max)、標準偏差値 (STDDEV)、サンプル サイズ (SAMPLESIZE) があります。

末尾に _BL が付くベースライン メトリックには、履歴データと過去 1 時間のデータが含まれます。

末尾に _BL_LASTHOUR が付くベースライン メトリックには、過去 1 時間のデータのみが含まれます。

Measurement Threshold ポリシーを作成して、過去 1 時間のベースライン データと履歴データを比較することができます。

例: Measurement Threshold ポリシーを使用して、GBL_CPU_TOTAL_UTIL_AVG_BL_LASTHOUR と GBL_CPU_TOTAL_UTIL_AVG_BL を比較できます。

ベースライン メトリック S0、S1、S2 は、標準偏差を計算するために使用されます。

注: HP Operations Agent ノード上でベースライン機能を停止するには、次のコマンドを実行します。

```
ovconfchg –ns oacore –set ENABLE_BASELINE FALSE
```

ベースライン計算が動作していないときのトラブルシューティングの方法

以下の手順に従ってください。

1. **ベースラインが有効であるかどうかを確認します。**
   a. HP Operations Agent ノードにログオンします。
   b. 以下のディレクトリに移動します。

      32 ビット版 Windows の場合:

      "%ovinstalldir%/bin"

      64 ビット版 Windows の場合:

      "%ovinstalldir%/bin/win64"

      AIX の場合:

      /usr/lpp/OV/bin

      HP-UX/Linux/Solaris の場合:

      /opt/OV/bin
   c. 次のコマンドを実行します。

      ovconfget oacore ENABLE_BASELINE

   ベースラインが有効な場合、出力は **TRUE** です。
2. **ベースライン メトリックの ovcodautil 出力を確認します。**
   a. 以下のディレクトリに移動します。

      32 ビット版 Windows の場合:

      "%ovinstalldir%/bin"

      64 ビット版 Windows の場合:

      "%ovinstalldir%/bin/win64"

      AIX の場合:

      /usr/lpp/OV/bin

      HP-UX/Linux/Solaris の場合:

      /opt/OV/bin
   b. 次のコマンドを実行します。

      ovcodautil -ds SCOPE -o <CLASS>_BASELINE -rawonly

      たとえば、次のようになります。

      ovcodautil -ds SCOPE -o GLOBAL_BASELINE -rawonly

   次に、目的のベースライン メトリックが一覧されている場合、出力を確認します。

3. **ベースライン データの計算が必要なメトリックがベースライン設定ファイルに記述されているかどうかを確認します。**
   a. メトリックが正しい形式で記述されているかを以下のファイルで確認します。

      Windows の場合:

      %ovdatadir%/baseline.cfg

      HP-UX/Linux/Solaris/AIX の場合:

      /var/opt/perf/baseline.cfg

      > **メモ:**
      >
      > ベースライン データの計算が必要なメトリックは、次のいずれかの形式で指定する必要があります。
      >
      > [Baseline]
      >
      > <Class>:<Metric>
      >
      > **たとえば、次のようになります。**
      >
      > Global:GBL_CPU_TOTAL_UTIL

4. **ベースライン設定ファイルに記述されているメトリックが使用しているプラットフォームで有効であることを確認します。**

   > **注:** ベースライン データは、ゲージ メトリックに対してのみ計算されます。

   a. 以下のディレクトリに移動します。

      32 ビット版 Windows の場合:

      "%ovinstalldir%/bin"

      64 ビット版 Windows の場合:

      "%ovinstalldir%/bin/win64"

      AIX の場合:

      /usr/lpp/OV/bin

      HP-UX/Linux/Solaris の場合:

      /opt/OV/bin

   b. 次のコマンドを実行します。

   ovcodautil -ds SCOPE –o <Class> -m <Metrics> -rawonly

   ここで、

   <Class> はメトリック クラスです。

   <Metrics> はベースライン データの計算が必要なメトリックです。

   ベースライン設定ファイルに記述されているメトリックが出力に一覧されているかどうかを確認します。

[1]履歴データは、1 時間前までに収集されたデータで、メトリック データストアに格納されます。

第10章: ノード解決の概要

ノードには、1 つまたは複数のネットワーク インターフェイスを使用できます。各ネットワーク インターフェイスに関連付けられる IP アドレスには、ホスト名が付けられている場合があります。HP Operations Agent は、システム設定に基づき、オペレーティング システム固有の API (gethostname /getaddrinfo /getnameinfo など) を使用して、ホスト名を選択します。これは、HP Operations Agent が使用するローカル ホスト名の値を提供する、OPC_NODENAME パラメータを自動的に設定します。OPC_NODENAME は内部パラメータであり、設定はできません。ノードに複数の IP アドレスまたはホスト名がある場合、OPC_IP_ADDRESS を特定の IP アドレスに設定することで、デフォルトの設定を上書きできます。 指定した IP アドレスに関連付けられているホスト名は、OPC_NODENAME に割り当てられます。IP アドレスがホスト名に関連付けられていない場合、OPC_NAMESRV_LOCAL_NAME を特定のホスト名に設定できます。OPC_NODENAME および OPC_IP_ADDRESS の定義済みの値は、ポリシー内のいくつかの変数を特定するために使用されます。メッセージの変数は次のとおりです。

- <$MSG_GEN_NODE>: メッセージを送信するノードの IP アドレスを返します。
- <$MSG_GEN_NODE_NAME>: メッセージを送信するノードのホスト名を返します。
- <$MSG_NODE>: 元のイベントが発生したノードの IP アドレスを返します。
- <$MSG_NODE_NAME>: 元のイベントが発生したノードのホスト名を返します。

いくつかの例を次に示します。

例 1:

複数のネットワーク インターフェイス環境での IP アドレスとホスト名を次に示します。

```
IP1  abc.test.com  abc

IP2  xyz.test.com  xyz
```

ここで、IP1 と IP2 は 2 つの異なる IP アドレスです。

abc.test.com と xyz.test.com は、完全修飾ドメイン名 (FQDN) です。

abc と xyz はホスト名です。

xyz は、ローカル システムのホスト名です。

OPC_NODENAME、<$MSG_GEN_NODE_NAME> の 2 つのパラメータは、FQDN 値を自動的に xyz.test.com に設定し、<$MSG_GEN_NODE> パラメータは IP アドレスを自動的に IP2 に設定します。xyz がローカル システムのホスト名であるため、これらの設定はデフォルトで実行されます。

次のコマンドを使用して、別の IP アドレス (IP1 など) の設定を選択します。

ovconfchg –ns eaagt –set OPC_IP_ADDRESS IP1

この設定の結果、パラメータの OPC_NODENAME と <$MSG_GEN_NODE_NAME> は、FQDN 値を abc.test.com に設定します。<$MSG_GEN_NODE> は、IP アドレスを IP1 に設定します。

例 2:

複数のネットワーク インターフェイス環境での IP アドレスとホスト名を次に示します。

```
IP1  xyz.test.com  xyz
IP2  xyz.test.com  xyz
```

ここで、IP1 と IP2 は 2 つの異なる IP アドレスです。

xyz.test.com は、完全修飾ドメイン名 (FQDN) です。

xyz は、ローカル システムのホスト名です。

<$MSG_GEN_NODE> パラメータは、IP アドレスを自動的に IP1 に設定します。xyz がローカル システムのホスト名であり、IP1 が xyz に関連付けられている最初の IP アドレス (IP アドレスの順序に従って) であるため、これらの設定はデフォルトで実行されます。

次のコマンドを使用して、別の IP アドレス (IP2 など) の設定を選択します。

ovconfchg –ns eaagt –set OPC_IP_ADDRESS IP2

この設定の結果、<$MSG_GEN_NODE> は IP アドレスを IP2 に設定します。

例 3:

複数のネットワーク インターフェイス環境では、Windows IPv6 システムには、次の順序で 4 つの設定済みの IP アドレスがあります。

```
IP1
IP2
IP3
IP4
```

これらの IP アドレスは、次の順序でホスト名に関連付けられています。IP1 はホスト名に関連付けられていません。

```
IP3  xyz.test.com  xyz
IP2  xyz.test.com  xyz
IP4  xyz.test.com  xyz
```

デフォルトでは、<$MSG_GEN_NODE> パラメータは、IP アドレスを自動的に IP2 に設定します。IP2 がローカル システム名に関連付けられている最初の設定済みの IP アドレス (IP アドレスの順序に従って) であるためです。

次のコマンドを使用して、別の IP アドレスの設定を選択します。

ovconfchg –ns eaagt –set OPC_IP_ADDRESS <IP_address>

この設定の結果、<$MSG_GEN_NODE> は特定の IP アドレスに設定されます。

例 4:

単一のネットワーク インターフェイス環境での IP アドレスとホスト名を次に示します。

```
IP1  abc.test.com  abc
IP1  xyz.test.com  xyz
```

ここで、IP1 は IP アドレスです。

abc.test.com と xyz.test.com は、完全修飾ドメイン名 (FQDN) です。

abc と xyz はホスト名です。

xyz は、ローカル システムのホスト名です。

デフォルトでは、OPC_NODENAME と <$MSG_GEN_NODE_NAME> の 2 つのパラメータは、FQDN 値を自動的に xyz.test.com に設定します。xyz がローカル システムのホスト名であるためです。

次のコマンドを使用して、IP アドレス IP1 の設定を選択できます。

ovconfchg – ns eaagt – set OPC_IP_ADDRESS IP1

この設定の結果、OPC_NODENAME と <$MSG_GEN_NODE_NAME> は、FQDN 値を abc.test.com に設定します。abc.test.com が IP アドレス IP1 に関連付けられている最初の FQDN であるため、これらの設定が実行されます。

例 5:

複数のネットワーク インターフェイス環境での IP アドレスを次に示します。

```
IP1
IP2  xyz.test.com  xyz
```

ここで、IP1 と IP2 は 2 つの異なる IP アドレスです。

IP1 はホスト名に関連付けられていません。

xyz.test.com は、完全修飾ドメイン名 (FQDN) です。

xyz は、ローカル システムのホスト名です。

デフォルトでは、OPC_NODENAME、<$MSG_GEN_NODE_NAME> の 2 つのパラメータは、FQDN 値を自動的に xyz.test.com に設定し、<$MSG_GEN_NODE> パラメータは IP アドレスを自動的に IP2 に設定します。

次のコマンドを使用して、別の IP アドレス (IP1 など) の設定を選択します。

ovconfchg – ns eaagt – set OPC_IP_ADDRESS IP1

この設定の結果、OPC_IP_ADDRESS は IP1 に設定され、ホスト名には関連付けられません。次のコマンドを使用して、OPC_IP_ADDRESS に対応するホスト名を設定できます。

ovconfchg – ns eaagt – set OPC_NAMESRV_LOCAL_NAME<host_name>

ここで、<host_name> は、abc.test.com などの任意の名前です。両方のコマンドを実行すると、OPC_NODENAME と <$MSG_GEN_NODE_NAME> の 2 つのパラメータは、FQDN 値を abc.test.com に設定します。

メモ: NAT 環境で設定された OPC_IP_ADDRESS パラメータは、前述の例と同様に動作します。

例 6:

単一のネットワーク インターフェイス環境での IP アドレスとホスト名を次に示します。

```
IP1  abc.test.com  abc xyz
IP1  xyz.test.com  xyz
```

ここで、IP1 は IP アドレスです。

abc.test.com と xyz.test.com は、完全修飾ドメイン名 (FQDN) です。

abc と xyz はホスト名です。

xyz は、ローカル システムのホスト名です。

デフォルトでは、OPC_NODENAME と <$MSG_GEN_NODE_NAME> の 2 つのパラメータは、FQDN 値を自動的に abc.test.com に設定します。名前が xyz のノードから外部イベントが発信される場合、<$MSG_NODE_NAME> パラメータは自動的に xyz に設定されます。

次のいずれかのオプションを実行することで、OPC_NODENAME、<$MSG_GEN_NODE_NAME>、および <$MSG_NODE_NAME> を FQDN 値の xyz.test.com に設定することを選択できます。

- hosts ファイル内のエントリの順序を交換します。
- 最初のエントリからエイリアス **xyz** を削除します。

例 7:

2 つのシステムの IP アドレスとホスト名を次に示します。

```
IP1  abc.test.com  abc
IP2  xyz.test.com  xyz
```

ここで、IP1 と abc は、リモート システムの IP アドレスとホスト名です。

IP2 と xyz は、ローカル システムの IP アドレスとホスト名です。

abc.test.com と xyz.test.com は、完全修飾ドメイン名 (FQDN) です。

IP1 からイベントが発信されると (リモート ノード abc からトラップが発信されるなど)、デフォルトで次の設定が実行されます。

- パラメータ <$MSG_NODE> は元のイベントが発生したノードの IP アドレスを返すため、<$MSG_NODE> は IP1 に設定されます。
- パラメータ <$MSG_NODE_NAME> は元のイベントが発生したノードのホスト名を返すため、<$MSG_NODE_NAME> は abc に設定されます。
- パラメータ <$MSG_GEN_NODE> はメッセージを送信するノードの IP アドレスを返すため、<$MSG_GEN_NODE> は IP2 に設定されます。

- パラメータ <$MSG_GEN_NODE_NAME> はメッセージを送信するノードのホスト名を返すため、<$MSG_GEN_NODE_NAME> は xyz に設定されます。

第11章: 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 Agent 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 Agent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 Agent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` ファイルは次の場所にあります。

Windows の場合:

`%ovdatadir%log`

HP-UX/Linux/Solaris の場合:

`/var/opt/OV/log`

さらに、HP Operations Agent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 の場合:

- `%ovdatadir%\log\System.txt`
- `%ovdatadir%\status.perfalarm`
- `%ovdatadir%\status.ttd`
- `%ovdatadir%\status.mi`
- `%ovdatadir%\status.perfd-<port>`
- `%ovdatadir%\hpcs\hpcstrace.log`

> **ヒント:** この場合、<port> は `perfd` で使用するポートです。デフォルトでは、`perfd` はポート 5227 を使用します。`perfd` のデフォルト ポートを変更するには、「RTMA コンポーネントの設定」を参照してください。

HP-UX/Linux/Solaris の場合:

- `/var/opt/OV/log/System.txt`
- `/var/opt/perf/status.perfalarm`
- `/var/opt/perf/status.ttd`

- /var/opt/perf/status.mi
- /var/opt/perf/status.perfd
- /var/opt/OV/hpcs/hpcstrace.log

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 Agent のメッセージ記録ポリシーを設定できます。

デフォルトの記録ポリシーを変更するには、次の手順を実行します。

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 の場合:

   %ovdatadir%conf\xpl\log

   HP-UX/Linux/Solaris の場合:

   /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、運用監視コンポーネントを再起動します。
   a. ovc -kill
   b. ovc -start

トレース

HP Operations Agent 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。

次の表で、トレース用のコマンド ラインのオプションと引数を示します。

コマンド ライン オプション

コマンド オプション	説明
-host	接続先のマシンのホスト名を指定します。 構文: -host <host_name>

コマンド ライン オプション (続き)

コマンド オプション	説明
-off	すべてのトレースをオフにします。
-cf|-configuration	動的変更用の設定ファイルの名前を指定します。 構文: -cf <config_file_name>
-vc|-viewconfig	すべてのアプリケーションのトレース設定を表示します。
-app|-application	設定対象のアプリケーション名のリストを指定します。
-cm|-component	設定対象のコンポーネント名のリストを指定します。
-sink	トレース出力ファイル名を指定します。 構文: -sink <filename>
-gc|-generate_configuration	生成対象の設定ファイルの名前を指定します。 構文: -gc <file_name>
-rd|-resetdefault	すべての .ini ファイルをデフォルトの内容にリセットします。
-h|-help	コマンド ライン構文を表示します。
-version	ツールのバージョン番号を表示します。

HP Operations Agent 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. アプリケーションの特定
2. トレース タイプの設定
3. オプション: 設定ファイルの作成

アプリケーションの特定

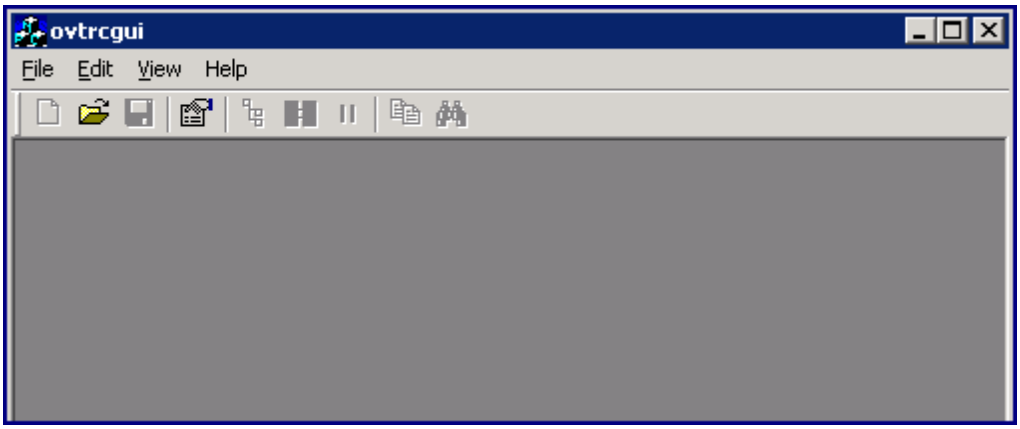
トレースする HP ソフトウェア アプリケーションを、管理対象システムで特定します。ovtrccfg -vc オプションを使用して、すべてのトレース対応アプリケーションの名前、および各トレース対応アプリケーションに定義されているコンポーネントとカテゴリを表示します。

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。

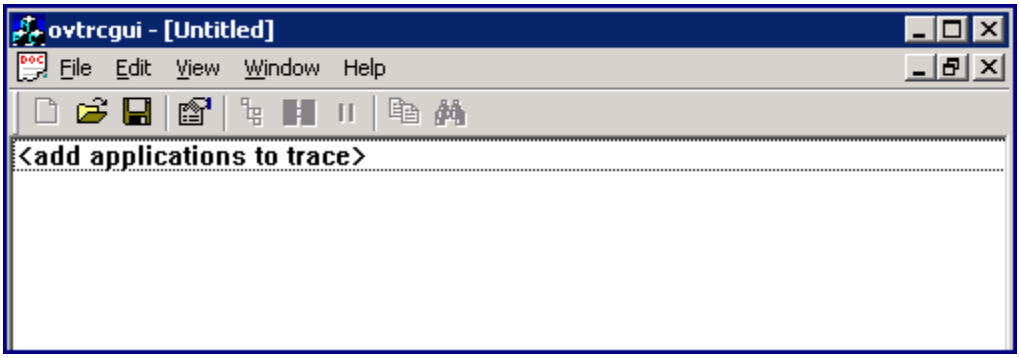
注: ovtrcgui ユーティリティは、Windows システムでのみサポートされています。

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。

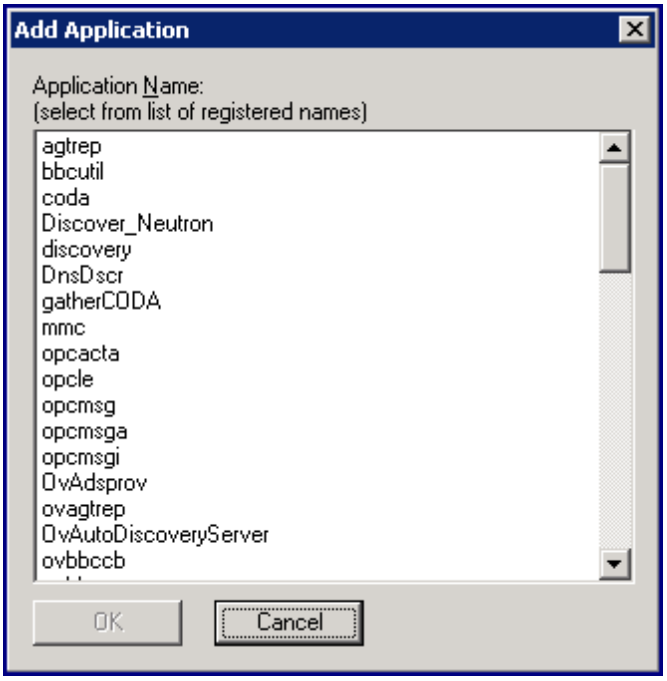
1. `ovtrcgui.exe` ファイルを `%OvInstallDir%\support` ディレクトリから実行します。ovtrcgui ウィンドウが表示されます。



2. ovtrcgui ウィンドウで **[File (ファイル)]** → **[New (新規)]** → **[Trace Configuration (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 **[Edit (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. <data_dir>/conf/xpl/trc/ の場所に移動します。
2. <application_name>.ini ファイルを見つけます。ファイルがあれば、手順 3 に進みます。<application_name>.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。

     ヒント: 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。たとえば、次のようになります。

     ```
     DoTrace=
     UpDateTemplate=
     DynamicTracing=
     ```

   - ファイルを保存します。

3. <application_name>.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 Version <version_number>
APP:"<application_name>"
SINK:File "<file_name>" "maxfiles=[1..100];maxsize=[0..1000];"
TRACE:"<component_name>" "<category_name>" <keyword_list>
```

構文の各行については、この後の章で説明します。

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、この項を省略して後述の「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 コマンドを使用して、設定ファイルに記載されている特定のアプリケーション用のトレースを動的に有効にします。
   /opt/OV/support/ovtrccfg -cf <configuration_file_name>

   ここで、<configuration_file_name> は、「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。

注: 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。

```
/opt/OV/support/ovtrccfg -app <application>[-cm <component>]
```

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。
     ```
     /opt/OV/support/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。
     ```
     /opt/OV/support/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。
     ```
     /opt/OV/support/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。
   ```
   /opt/OV/support/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、サフィックス n が付いています (ここで n は 1 から99999 の整数です)。

例:

```
SINK:File "coda.trc" "force=0;maxfiles=10;maxsize=10;"
root@/var/opt/OV/tmp/trc->lrt
total 93092
-rw-r----- 1 root bin 10475620 Aug 12 15:30 coda_00013.trc
-rw-r----- 1 root bin 10475528 Aug 12 15:30 coda_00014.trc
-rw-r----- 1 root bin 10475528 Aug 12 15:30 coda_00015.trc
-rw-r----- 1 root bin 10475528 Aug 12 15:30 coda_00016.trc
-rw-r----- 1 root bin 10475528 Aug 12 15:30 coda_00017.trc
-rw-r----- 1 root bin 10475528 Aug 12 15:30 coda_00018.trc
-rw-r----- 1 root bin 10475528 Aug 12 15:30 coda_00019.trc
-rw-r----- 1 root bin 10475528 Aug 12 15:30 coda_00020.trc
-rw-r----- 1 root bin 10475528 Aug 12 15:31 coda_00021.trc
```

```
-rw-r----- 1 root bin  1027187 Aug 12 15:31 coda_00022.trc
root@/var/opt/OV/tmp/trc->
```

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

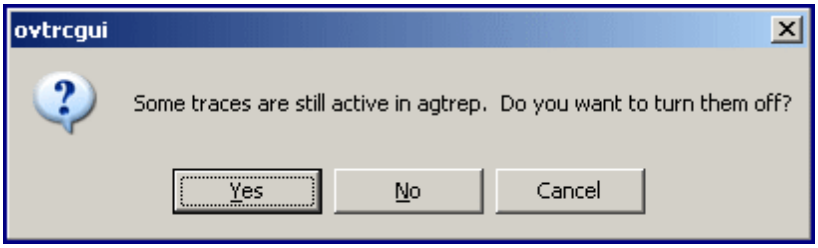
Windows ノードでは、ovtrcgui ユーティリティを使用して、トレースを設定し、トレース メッセージを表示できます。

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. 「トレース GUI の使用」(214ページ)のステップ 1 ~ ステップ 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、**[No]** をクリックします。

   表示されるメッセージに従います。



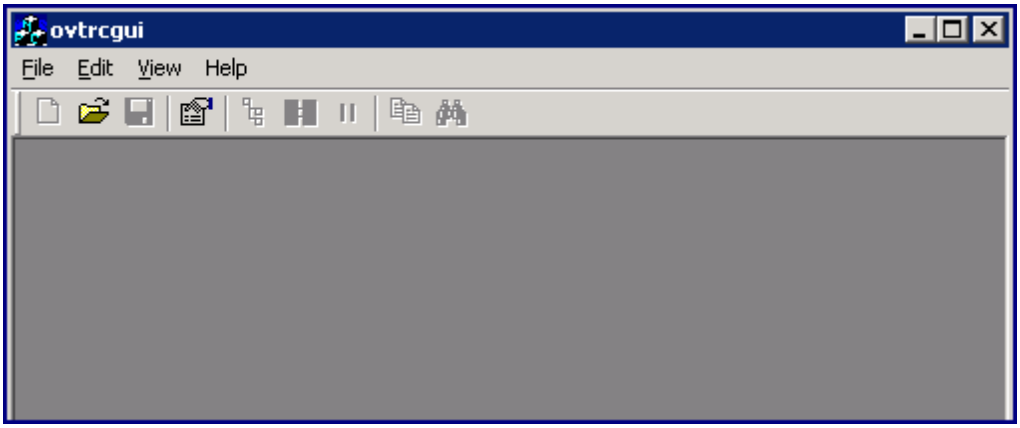
4. **[No]** をクリックします。**[Yes]** をクリックすると、ovtrcgui ユーティリティは直ちにトレース機構を無効にします。

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、**[File (ファイル)]** → **[Open (開く)]** をクリックして、トレース設定ファイルを選択し、**[Open (開く)]** をクリックします。

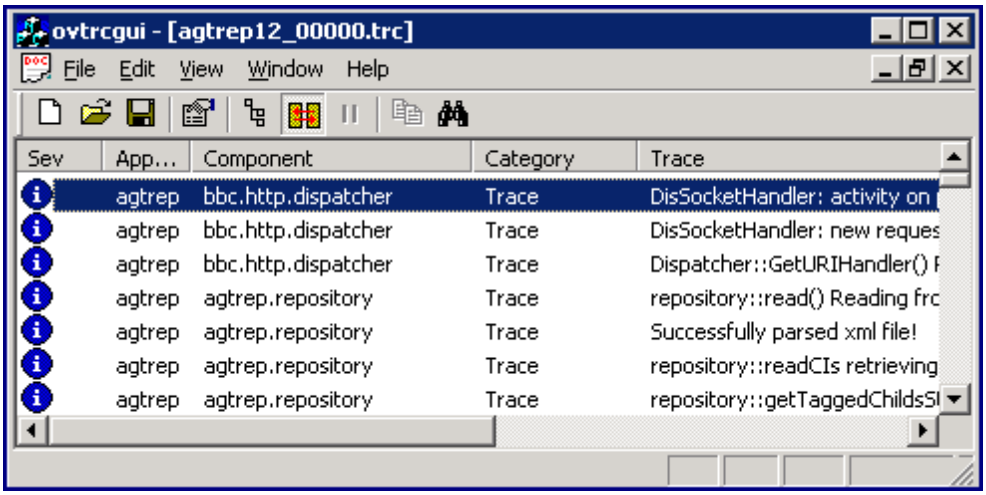
トレース メッセージの表示

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。

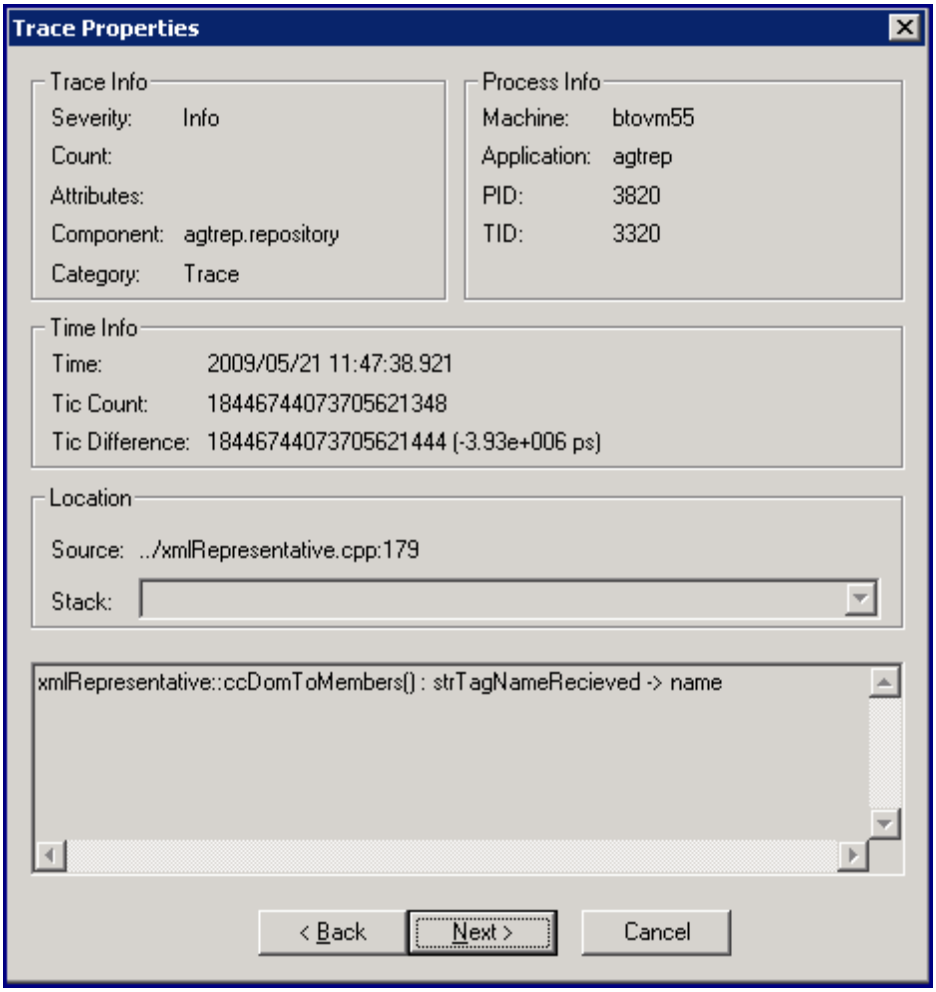


2. **[File (ファイル)] → [Open (開く)]** をクリックします。[Open (開く)] ダイアログ ボックスが表示されます。
3. トレース出力ファイルが配置されている場所に移動し、`.trc` ファイルを選択して、**[開く]** をクリックします。ovtrcgui ユーティリティに `.trc` ファイルの内容が表示されます。



`.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。
  - [属性]:トレース メッセージの属性です。
  - Component (コンポーネント): 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。
  - Category (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。
  - Application (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。

  - 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 (期間): 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)
  - Source (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内の呼び出しスタックの説明。

5. **[Next (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、**[Cancel (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。

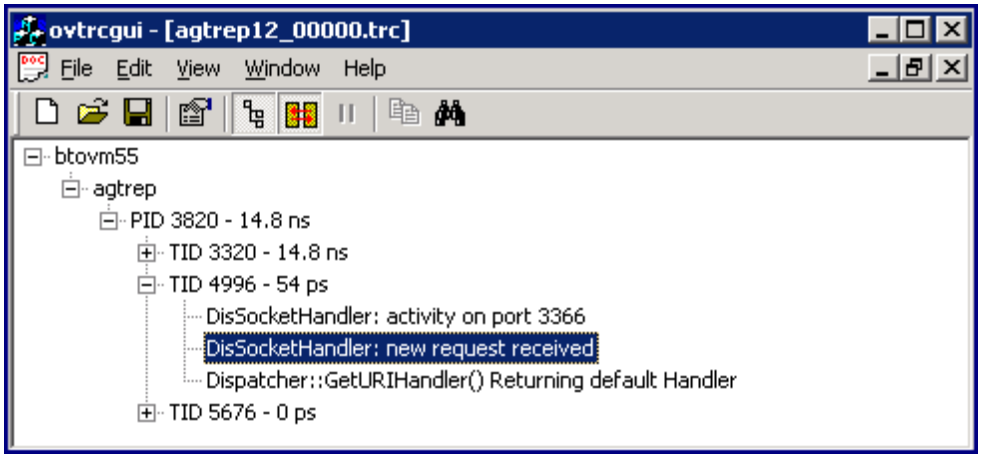
このトレース リスト ビューには、すべてのトレース メッセージが次の列と共に表示されます。

表 13: トレース リスト ビュー

カラム	説明	
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。	
コンポーネント	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べ替えられ、ツリー ビューの形式でデータが表示されます。

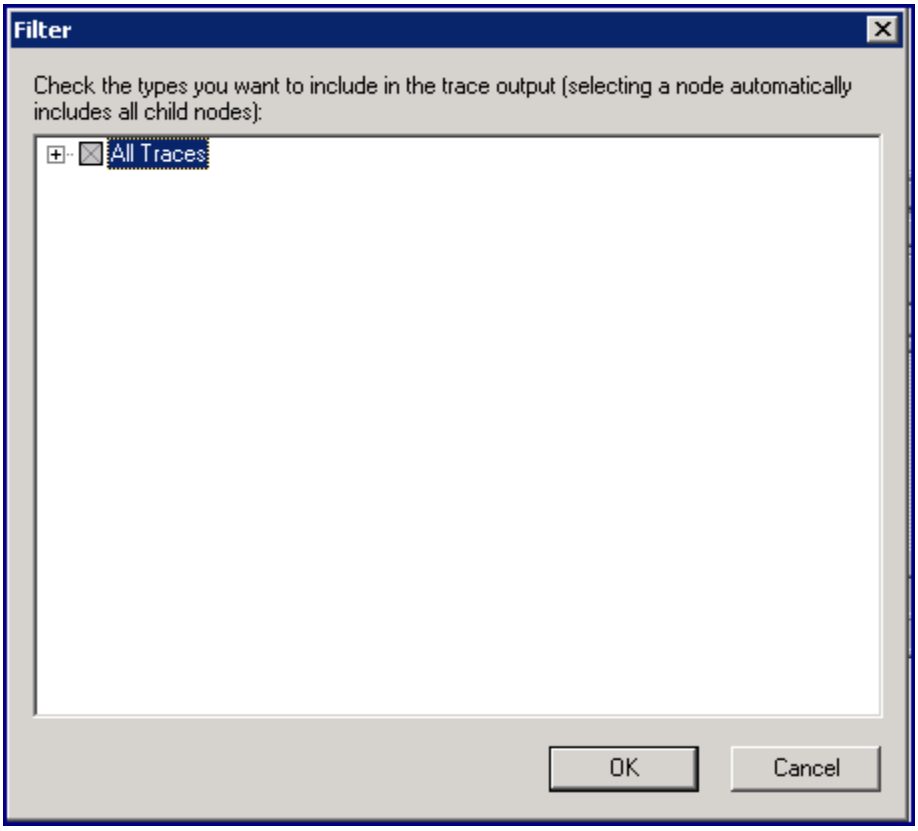


プロセス ID とスレッド ID を展開して、トレース メッセージを表示できます。トレース リスト ビューに戻るには、 をクリックします。

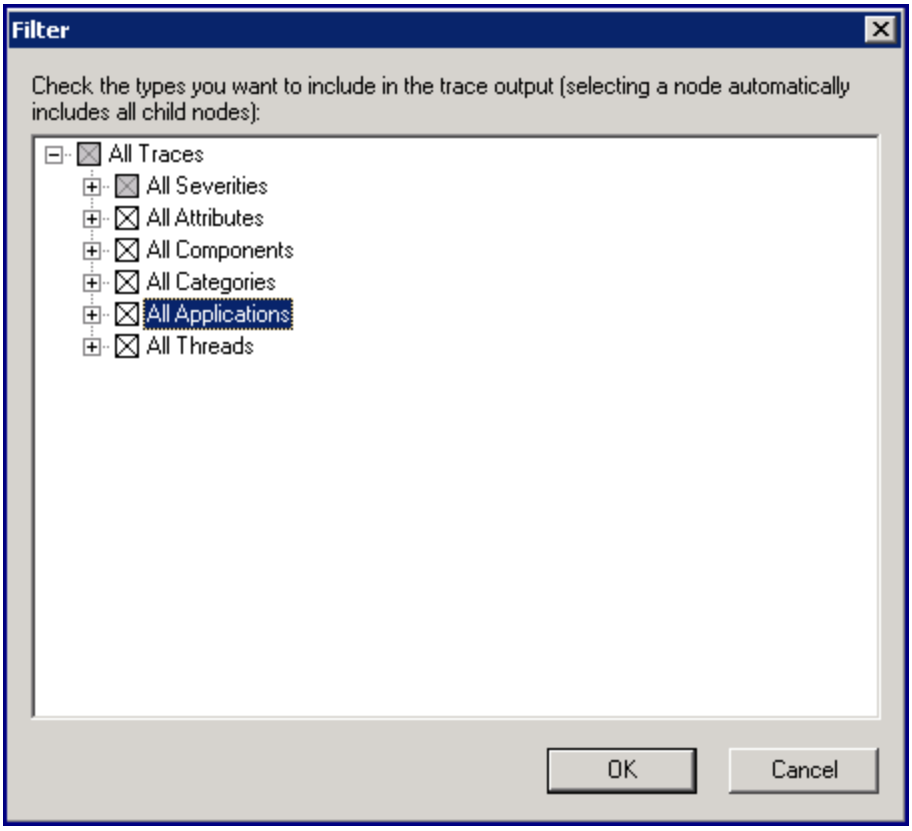
トレースのフィルタリング

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 **[View (表示)]** → **[Filter (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。

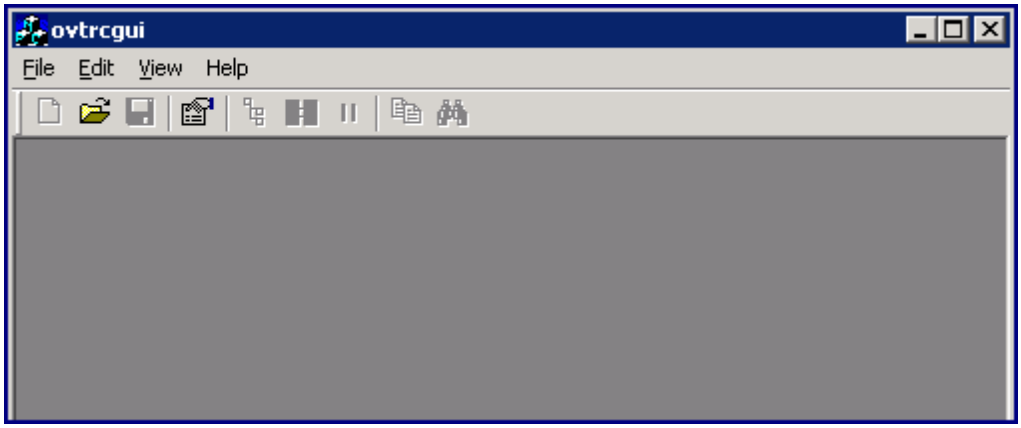


3. パラメータを展開して、トレース メッセージをフィルタリングする項目を選択します。
4. **[OK]** をクリックします。ovtrcgui コンソールにはフィルタリングされたメッセージのみが表示されます。

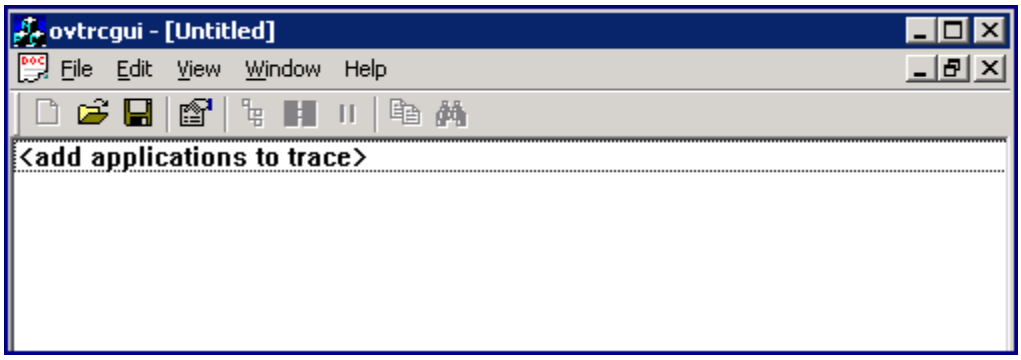
トレース GUI の使用

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。

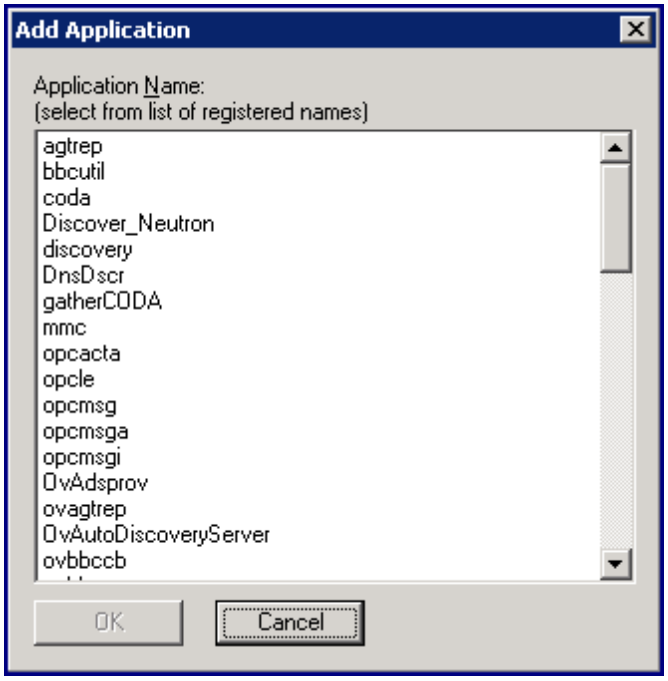
1. `ovtrcgui.exe` ファイルを `%OvInstallDir%\support` ディレクトリから実行します。ovtrcgui ウィンドウが表示されます。



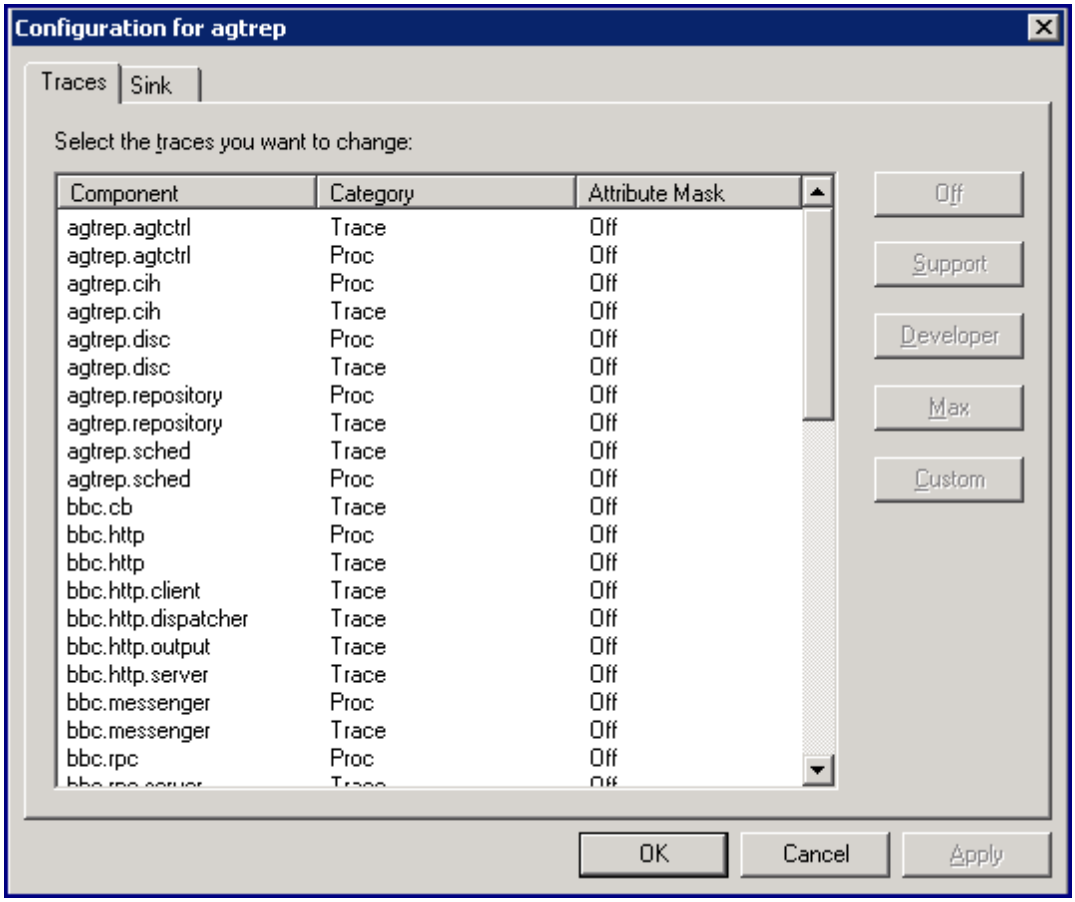
2. ovtrcgui ウィンドウで **[File (ファイル)]** → **[New (新規)]** → **[Trace Configuration (トレースの設定)]** をクリックします。新しいトレース設定エディタが開きます。



3. ovtrcgui ウィンドウで **[Edit (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。

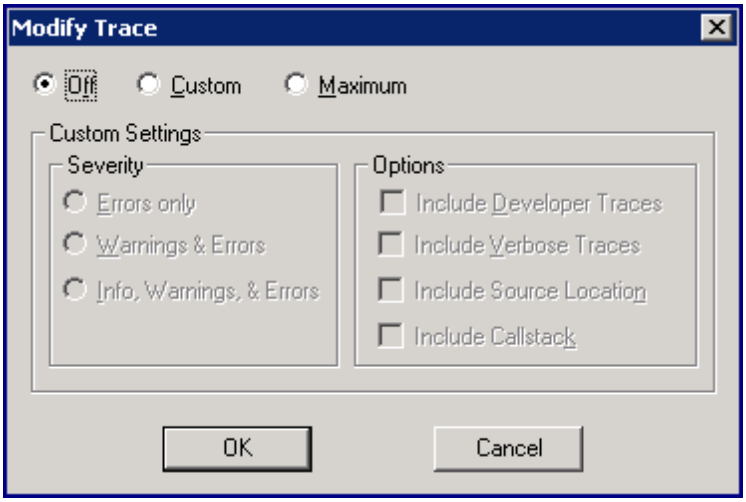


4. トレースするアプリケーションを選択し、**[OK]** をクリックします。[Configuration for <application> (<アプリケーション> の設定)] ウィンドウが表示されます。



[Configuration for <application> (<アプリケーション> の設定)] ウィンドウの [Trace (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. [Trace (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。

> **ヒント:** [Configuration for <application> (<アプリケーション> 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. [File Name (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。

   > **ヒント:** `.trc` ファイルの完全パスを指定します。

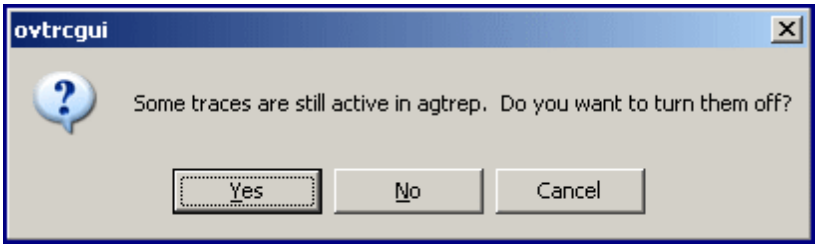
9. ドロップダウン リストから履歴ファイルの数を指定します (maxfiles を参照)。
10. ドロップダウン リストから最大ファイル サイズを指定します (maxsize を参照)。
11. **[Apply (適用)]** をクリックします。
12. **[OK]** をクリックします。

    > **注:** `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. **[File (ファイル)]** → **[Save (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。

    [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。

14. `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. [**No**] 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。**[Yes]** 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。

第IV部: カスタム データの記録

HP Operations Agent を使用すると、カスタム データをデフォルトのパフォーマンス メトリック クラスと共にメトリック データストアに記録できます。送信用の API または DSI のいずれかを使用すると、カスタム データをメトリック データストアに記録できます。

カスタム データのメトリック データストアへのログ記録のプロセスは、Perl ベースのスクリプト可能な API の使用によって簡略化されています。API を使用することで、マルチインスタンス データおよび 64 ビット データ タイプを記録できます。API の詳細については、「Perl アプリケーション プログラミング インターフェイスによるカスタム データの送信」を参照してください。

DSI を使用してカスタム データを送信するには、データ記録用のモデルを定義するクラスの仕様ファイルを作成し、データをメトリック データストアに記録する必要があります。DSI を使用して、64 ビット データ タイプおよびマルチインスタンス データを記録することはできません。DSI の詳細については、「データ ソース統合の概要」を参照してください。

注: カスタム データを メトリック データストア に記録するには、API を使用することを推奨します。

第12章: Perl アプリケーション プログラミング インターフェイスによるカスタム データの送信

カスタム データのログ記録のプロセスは、Perl ベースのスクリプト可能な API の使用によって簡略化されています。API を使用することで、カスタム データをメトリック データストアにシームレスに記録できます。API を使用することで、64 ビット データ タイプおよびマルチインスタンス データをデータストアに記録できます。

注: API を使用してメトリック データストアにデータを記録できるのは、**oacore** プロセスが実行されている場合だけです。

HP Operations Agent 12.00 には、カスタム データをメトリック データストアに記録するために、次の API が用意されています。

API タイプ	要約タイプ	データ型	API タイプの選択方法
AddCounter	要約間隔の最後の値を提供します。	Integer	Counter メトリックを使用して、CPU 時間、物理 IO、ページング、ネットワーク パケット数などのアクティビティの累積カウントをデータストアに送信します。
AddGauge	要約間隔におけるすべての値の平均を提供します。	Integer、Real	Gauge メトリックを使用して、実行キュー、ユーザー数、ファイル システムのスペース使用率など、監視時における即時値をデータストアに送信します。
AddAttribute	頻繁に変化しないデータを提供します。要約は適用できません。 **注:** 属性の履歴データはデータストアには保存されません。属性が変更されると、必ずデータストアで置換されます。	Integer、Real、String	Attribute メトリックを使用して、OS 名、バージョン、リリース、物理メモリ、CPU クロック速度などの静的な定義または値をデータストアに送信します。

注:

- 異なる API タイプを使用して、メトリックを送信しないようにしてください。

  次の場合:

  ```
  AddCounter("Website:Global:UserLogins", "www.abcdefg.com", "Cumulative User Logins", 6000);
  AddGuage("Website:Global:UserLogins", "www.abcdefg.com", "Cumulative User Logins", 5000);
  ```

  Counter および Gauge の両方の API タイプを使用して「UserLogins」メトリックを送信すると、結果が不正になります。
- メトリックの送信後に、メトリックのタイプを変更することはできません。

たとえば、次のようになります。メトリックを Counter として送信後に、Gauge または Attribute に変更することはできません。

このドキュメントでは、以下の条件が使用されています。

- MetricObservation オブジェクトには、データストアに記録するデータが、収集タイムスタンプと共に含まれている。
- MetricObservationList には、いくつかの MetricObservation オブジェクトが含まれている。
- OAAccess は、データストアに接続するクライアントである。
- SubmitObservations API は、MetricObservationList をメトリック データストアに送信するために使用される。

  SubmitObservations は、次のいずれかの値を返す。

  - 0 - MetricObservationList が正常にメトリック データストアに送信された
  - -1 - MetricObservationList の送信が失敗した
  - -2 - **oacore** プロセスが実行されていない

カスタム データの送信

構文

```
<AddMethod>(<data source name>:<class name>:<metric name>, <Instance Identifier>, <Label of the metric>,<value of metrics>)
```

API の引数

次の表は、引数を示しています。

引数	説明

完全修飾メトリック名	完全修飾メトリック名を指定します。これには、データソース名、クラス名、およびメトリック名が含まれます。 <data source name>:<class name>:<metric name>
Instance Identifier	クラスのインスタンスを指定します。整数、実数または文字列値のいずれかを指定できます。
Label of the metric	メトリックのラベルを指定します。
Value of metrics	送信するメトリックの値を指定します。 **注:** AddGauge メソッドの場合、Value は整数または実数のいずれかになります。 AddAttribute メソッドの場合、Value は整数、実数または文字列のいずれかになります。

例:

AddGauge("MySQL:Global:MySQLMemUtil", "MyHost:3306", "MySQL Total CPU utilization percentage", 20.12);

ここで、

AddMethod	AddGauge
Data source name	MySQL
Class name	Global
Metric name	MySQLMemUtil
Instance Identifier	MyHost:3306
Label of the metric	MySQL Total CPU utilization percentage
Value of metrics	20.12

カスタム データをメトリック データストアに送信する方法

データをメトリック データストアに送信するには、MetricObservation オブジェクトを作成します。MetricObservation オブジェクトを MetricObservationList に追加します。

注: 任意の数の MetricObservation オブジェクトを MetricObservationList に追加できます。

SubmitObservations API を使用して、MetricObservationList をメトリック データストアに送信します。

注: SubmitObservations API を呼び出すには、OAAccess オブジェクトを作成する必要があります。

次の表は、データをメトリック データストアに送信する手順を例と共に示します。

番号	カスタム データをメトリック データストアに送信する手順	例
1.	OAAccess を作成します。	$access = oaperlapi::OAAccess->new();
2.	MetricObservationList オブジェクトを作成します。	$mol = oaperlapi::MetricObservationList->new();
3.	MetricObservation オブジェクトをタイムスタンプ付きで作成します。 **注:** 異なるタイムスタンプを使用して、複数の MetricObservation オブジェクトを作成できます。時間が 0 の場合、現在の時間が取られます。 送信するデータのすべてのクラスに対して、MetricObservations のタイムスタンプが昇順であることを確認します。	$interval = time; $mo = oaperlapi::MetricObservation->new($interval);
4.	AddGauge または AddAttribute などのメソッドを使用して、MetricObservation オブジェクトに値を追加します。	$mo->AddAttribute ("MySql:Global:MySQLVersion", "MyHost:3306", "MySQL version", "5.6.0"); $mo->AddGauge ("MySql:Global:MySQLMemUtil", "MyHost:3306", "MySQL Total CPU utilization percentage", 20.12);
5.	MetricObservation オブジェクトを MetricObservationList オブジェクトに追加します。	$mol->AddObservation($mo);
6.	SubmitObservations API を使用して、MetricObservationList をメトリック データストアに送信します。	$access->SubmitObservations ($mol);

例:

```
$access = oaperlapi::OAAccess->new();
$mol = oaperlapi::MetricObservationList->new();
$interval = time;
$mo = oaperlapi::MetricObservation->new($interval);
$mo->AddAttribute("MySql:Global:MySQLVersion", "MyHost:3306", "MySQL version", "5.6.0");
$mo->AddGauge("MySql:Global:MySQLMemUtil", "MyHost:3306", "MySQL Total CPU utilization percentage", 20.12);
$mol->AddObservation($mo);
$access->SubmitObservations($mol);
```

API を使用してデータをデータストアに送信するための使用例

シナリオ

John は、Web サイト www.abcdefg.com の Web 管理者です。彼は、USA から彼の Web サイトにアクセスするユーザー数を分析しようとしています。John はまた、10 A.M ~ 11 A.M の間に行われた購入数を監視することも望んでいます。

説明

John は、次の値を把握しようとしています。

- USA から 1 時間中にログインしたユーザー数。
- 1 時間に行われた購入数。

出演者

John - Web 管理者

前提条件

ユーザーのログインと購入を監視する時間は 1 時間。

時間間隔	累積ユーザー ログイン数	場所	1 時間の購入数
10:00 ~ 10:30	6000	USA	400
10:30 ~ 11:00	8000	USA	600

注:

パスの接頭辞が `<InstallDir>/nonOV/perl/a/bin` であることを確認します。

次のコマンドを使用して次のスクリプトを実行します。

Windows x64 の場合:

```
perl -I <Installdir>\support -I <Installdir>\nonOV\perl\a\lib\site_
perl\5.16.0\MSWin32-AMD64-multi-thread <perl script>
```

Windows x86 の場合:

```
perl -I <Installdir>\support -I <Installdir>\nonOV\perl\a\lib\site_
perl\5.16.0\MSWin32-x86-multi-thread <perl script>
```

HP-UX PA-RISC の場合:

LD_PRELOAD は、perl スクリプトの実行中に liboaperlapi.sl のパスを設定する必要があります。

例:

```
LD_PRELOAD=/opt/OV/nonOV/perl/a/lib/site_perl/5.16.0/PA-RISC2.0-thread-
multi/liboaperlapi.sl /opt/OV/nonOV/perl/a/bin/perl <perl script>
```

UNIX の場合:

```
perl <perl script>
```

データをデータストアに送信するには、次の Perl スクリプトを実行します。

```
use oaperlapi;

$now_string = localtime;
```

 # ステップ 1: access オブジェクトを作成

```
$access = oaperlapi::OAAccess->new();
```

 # ステップ 2: MetricObservationList オブジェクトを作成

```
$molist = oaperlapi::MetricObservationList->new();
$obsTimestamp = 1415939400;
```

メモ: `$obsTimestamp` は、最初のレコードがデータストアに記録された時刻を表します。

 # ステップ 3: MetricObservation オブジェクトをタイムスタンプ付きで作成

```
$mgbl = oaperlapi::MetricObservation->new($obsTimestamp);
```

 # ステップ 4: MetricObservation オブジェクトに値を追加

```
$mgbl->AddAttribute("Website:Global:Location", "www.abcdefg.com", "Location",
"USA");
$mgbl->AddCounter("Website:Global:UserLogins", "www.abcdefg.com", "Cumulative User
Logins", 6000);
$mgbl->AddGauge("Website:Global:Purchases", "www.abcdefg.com", "User Purchases in
an hour", 400);
```

 # ステップ 5: MetricObservation オブジェクトを MetricobservationList に追加

```
$molist->AddObservation($mgbl);
# 次のタイムスタンプへ移動
$obsTimestamp += 1800;
```

 # ステップ 3、4、5 を繰り返して、次に示すように別の監視を作成します。

```
$mgbl = oaperlapi::MetricObservation->new($obsTimestamp);
$mgbl->AddCounter("Website:Global:UserLogins", "www.abcdefg.com", "Cumulative User
Logins", 8000);
$mgbl->AddGauge("Website:Global:Purchases", "www.abcdefg.com", "User Purchases in
an hour", 600);
$molist->AddObservation($mgbl);
```

 # ステップ 6: データを一括送信

```
$access->SubmitObservations($molist);
```

次のデータがデータストアに記録されます。

```
===
11/14/14 10:00:00|GLOBAL_ID                      |www.abcdefg.com|
11/14/14 10:00:00|LOCATION                        |USA       |
11/14/14 10:00:00|USERLOGINS                      |      6000|
11/14/14 10:00:00|PURCHASES                       |    400.00|
===
11/14/14 10:30:00|GLOBAL_ID                      |www.abcdefg.com|
11/14/14 10:30:00|LOCATION                        |USA       |
11/14/14 10:30:00|USERLOGINS                      |      8000|
11/14/14 10:30:00|PURCHASES                       |    600.00|
```

第13章: データ ソース統合の概要

データ ソース統合 (DSI) 機能により、カスタム データの記録、アラームの定義、Performance Collection Component の **oacore** データ 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できます。

DSI によって記録されるデータは、HP Operations Agent のデータ 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 Performace Manager で表示できます。また、Extract プログラムを使用すると、DSI を使用して記録されたデータを、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できます。

HP Operations Agent 12.00 へのアップグレード

HP Operations Agent 12.00 では、DSI によって収集されたデータは メトリック データストア に記録されます。メトリック データストアに記録されるデータのすべてのクラスに対して、固有のデータベース ファイルが作成されます。ただし、以前のバージョンとの互換性のため、コマンド ラインでは引き続きログファイル引数がサポートされます。

```
sdlcomp <Class specification file> <log file name>
```

DSI コンパイラ **sdlcomp** は、ログ ファイル名を基にファイルを作成しません。代わりにログ ファイル名はデータソース名として使用されます。

> **例:**
>
> `"/tmp/test_log"` または `"C:\test_log"` がログ ファイル引数 (コマンド ラインで) として使用される場合、test_log はデータソース名として使用されます。

HP Operations Agent 11.xx から 12.00 にアップグレードする前に、以下に保存されている `datasources` ファイルに DSI データストアが追加されていることを確認します。

HP-UX/Linux/Solaris の場合:

```
/var/opt/OV/conf/perf
```

Windows の場合:

```
%ovdatadir%\conf\perf
```

以下のエントリを datasources ファイルに追加します。

```
DATASOURCE=<Datasource Name>LOGFILE=<DSI Logfile Path>
```

DSI データソースを `datasources` ファイルに追加した場合にのみ、メトリック データストア 内に DSI データを記録するためのモデルが作成されます。

> **注:** HP Operations Agent 12.00 でサポートされていないオプションはすべて無視され、以下の例に示されるメッセージが表示されます。

例:

```
MAX INDEXES is obsoleted and will be ignored
INDEX BY is obsoleted and will be ignored
CAPACITY is obsoleted and will be ignored
ROLL BY is obsoleted and will be ignored
ACTION is obsoleted and will be ignored
PRECISION is obsoleted and will be ignored
```

DSI の動作

DSI データを メトリック データストア に記録するには、以下の手順を実行します。

1. DSI データを メトリック データストア に記録するために、 を作成します。詳細については、「DSI メトリックをメトリック データストアに記録するためのモデルの作成」を参照してください。
   a. クラス仕様ファイルを作成します。詳細については、「クラス仕様のファイルの作成」(231ページ)を参照してください。
   b. DSI コンパイラ **sdlcomp** を使用してクラス指定ファイルをコンパイルするために、以下のコマンドを実行します。

      ```
      sdlcomp <class specification file> <log file name>
      ```

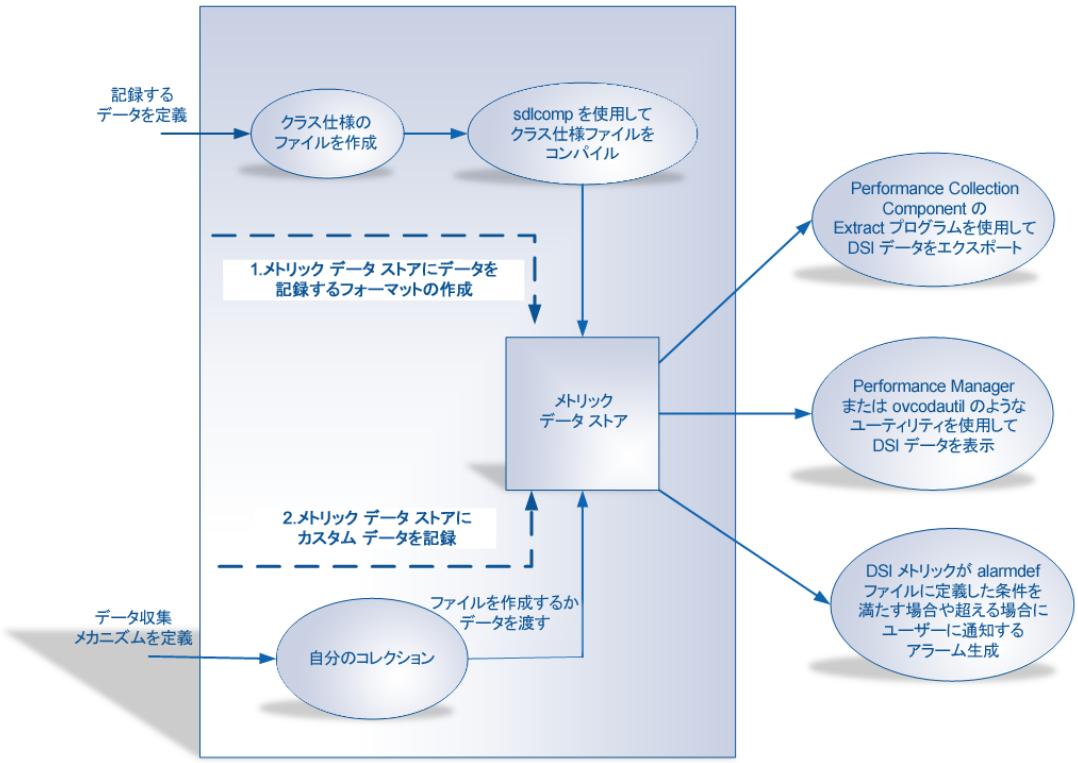
      詳細については、「DSI コンパイラによるクラスの仕様ファイルのコンパイル」(239ページ)を参照してください。

   データ (メタ データ) を記録するためのモデルは、メトリック データストア に保存されます。
2. DSI データを メトリック データストア に記録します。詳細については、「DSI メトリックのメトリック データストアへの記録」(241ページ)を参照してください。

   カスタム データを dsilog に渡します。dsilog はデータを メトリック データストア に記録します。

次の図は、DSI が収集したデータの メトリック データストア への記録を示しています。

図 17 データ ソース統合のプロセス



DSI データをエクスポートするには、Performance Collection Component の **extract** プログラムを使用できます。DSI データは、HP Performance Manager や `ovcodautil` などのユーティリティを使用して表示できます。DSI メトリックが `alarmdef` ファイルに定義した条件を超えたときのアラームを設定できます。

第14章: DSI メトリックをメトリック データストアに記録するためのモデルの作成

DSI メトリックをメトリック データストアに記録する前に、DSI メトリックを記録するためのモデルを作成および記録する必要があります。以下の手順に従って、モデルをメトリック データストアに記録します。

1. クラスの仕様ファイルの作成
2. モデルのメトリック データストアへの記録

クラス仕様のファイルの作成

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モデル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、「クラスの仕様構文」(232ページ)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。

- 受信データ セットに名前と数値 ID を割り当てるクラスの記述。クラスの記述の詳細は、「クラスの記述」(232ページ)を参照してください。
- 記録するメトリックを定義するメトリックの記述。メトリックの記述は名前を定義し、データ項目を記述します。メトリックの記述の詳細は、「メトリックの記述」を参照してください。

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テキスト エディタまたはワード プロセッサを使用します。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。
- メトリック名

クラスの仕様の例を次に示します。

```
CLASS VMSTAT_STATS = 10001
LABEL "VMSTAT data"
RECORDS PER HOUR 120
;

METRICS
RUN_Q_PROCS       = 106
LABEL     "Procs in run q"
;

BLOCKED_PROCS   = 107
```

```
LABEL     "Blocked Processes"
;
```

クラスの仕様ファイルには複数のクラスが含まれます。

DSI コンパイラ **sdlcomp** を実行する場合は、DSI データを記録するモデルが メトリック データストアに記録されます。

クラスの仕様構文

クラス仕様ファイルを作成するには、以下の名称を使用します。

- [ ] (大かっこ) 内に示されている構文は、オプションです。
- 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。
- 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。
- すべてのクラスの記述、メトリックの記述の後にはセミコロンを追加します。
- 文では、大文字と小文字を区別しません。

注: 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。

たとえば、次のようになります。

```
CLASS class_name = class_id_number
[LABEL "class_label_name"]
[RECORDS PER HOUR  number]

;

METRICS
metric_name = metric_id_number
[LABEL "metric_label_name"]
;
```

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"]

[RECORDS PER HOUR number]

;

CLASS

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。

- メトリック名には、最大 20 文字までの英数字を使用できます。クラス名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)。大文字と小文字の区別はありません。
- class_ID_number は、最大 6 文字までの英数字でなければなりません。
- 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** ファイルの詳細については、「parm ファイルの使用」を参照してください。

例:

```
CLASS VMSTAT_STATS = 10001;
```

LABEL

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。
- 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";
```

Records per Hour

RECORDS PER HOUR の設定は、1 時間あたりのデータベース 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、データがデータベース 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。詳細については、「要約方法」を参照してください。

構文

[RECORDS PER HOUR number]

Records per Hour の使用方法

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      --> 1 record/10 minutes
RECORDS PER HOUR 12     --> 1 record/5 minutes
RECORDS PER HOUR 60     --> 1 record/minute
RECORDS PER HOUR 120    --> 1 record/30 seconds
```

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。

例:

```
CLASS VMSTAT_STATS = 10001
LABEL  "VMSTAT data"
RECORDS PER HOUR  6;
```

この例では、レコードは 10 分おきに書き込まれます。

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)
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。

クラス仕様のサンプル

```
CLASS VMSTAT_STATS = 10001
LABEL "VMSTAT data"
RECORDS PER HOUR 120;
METRICS
RUN_Q_PROCS = 106
LABEL "Procs in run q"
;
BLOCKED_PROCS = 107
LABEL "Blocked Processes"
;
SWAPPED_PROCS = 108
LABEL "Swapped Processes"
;
AVG_VIRT_PAGES = 201
LABEL "Avg Virt Mem Pages"
;
FREE_LIST_SIZE = 202
LABEL "Mem Free List Size"
;
PAGE_RECLAIMS = 303
LABEL "Page Reclaims"
;
ADDR_TRANS_FAULTS = 304
LABEL "Addr Trans Faults"
```

```
;
PAGES_PAGED_IN = 305
LABEL "Pages Paged In"
;
PAGES_PAGED_OUT = 306
LABEL "Pages Paged Out"
;
PAGES_FREED = 307
LABEL "Pages Freed/Sec"
;
MEM_SHORTFALL = 308
LABEL "Exp Mem Shortfall"
;
CLOCKED_PAGES = 309
LABEL "Pages Scanned/Sec"
;
DEVICE_INTERRUPTS = 401
LABEL "Device Interrupts"
;
SYSTEM_CALLS = 402
LABEL "System Calls"
;
CONTEXT_SWITCHES = 403
LABEL "Context Switches/Sec"
;
USER_CPU = 501
LABEL "User CPU"
;
SYSTEM_CPU = 502
LABEL "System CPU"
;
IDLE_CPU = 503
LABEL "Idle CPU"
```

;

メトリックの記述

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、最初のメトリック定義の前に 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。

- メトリック名は最大 20 文字まで使用できます。
- 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。

メトリック ID は最大 6 文字まで使用できます。

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。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。

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。

> **注:** 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。

LABEL

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: ラベル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。たとえば、ラベルが `"\"my\" data"` である場合、“my” data と入力します。

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。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 をかけて、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。

たとえば、次のようになります。

```
METRICS
ITEM_1_3 = 11203
LABEL    "TOTAL_ORDERS"
TOTALED;
ITEM_1_5   = 11205
```

```
LABEL   "LINES_PER_ORDER"
SUMMARIZED BY ITEM_1_3;
```

DSI コンパイラによるクラスの仕様ファイルのコンパイル

クラスの仕様ファイルを使用して、DSI コンパイラ **sdlcomp** を実行します。

コンパイラ構文

`sdlcomp <class specification file> <log file name>`

ここで、

`<class specification file>` は、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。

`<log file name>` は、データソース名として使用されます。

DSI コンパイラ **sdlcomp** は、ログ ファイル名に基づいてファイルを作成することはしません。代わりに、ログ ファイル名がデータソース名として使用されます。

例:

"/tmp/test_log" または "C:\test_log" がログファイル セット引数として与えられた場合 (コマンドラインで)、test_log がデータソース名として使用されます。

DSI データを記録するためのモデルがメトリック データストアに記録されます。

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、メトリック データストア にデータソース、クラス、メトリックの記述が追加されます。

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、次のようにデータソース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. すべてのクラスについて、メトリック データストア からデータをエクスポートします。
3. `sdlutil` を実行してデータソースを削除します。詳細については、「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. `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。

第15章: DSI メトリックのメトリック データストアへの記録

DSI メトリックをメトリック データストアに記録するには、DSI データを記録するためのモデルがデータストアに記録されていることを確認します。DSI メトリックが定義条件を超える場合にユーザーに通知するように、アラーム定義ファイル **alarmdef** を変更します。DSI メトリックのアラームの定義の詳細は、「DSI メトリックのアラーム定義」を参照してください。コマンド ラインから収集プロセスを起動します。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** (またはその他のメソッドで **stdin**) に渡します。

構文

`dsilog <sdl-file-name> <data-class-name> [options]`

ここで、

`<sdl-file-name>` は、ルートの自己記述ファイルの名前です。

`<data-class-name>` は、データ クラスの名前です。

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
-c <char>	指定文字を文字列の区切り記号として使用します。
-i <fifo>	指定した `fifo` から入力されることを示します。fifo には、入力ストリームのための別の選択肢を指定します。
-timestamp	入力データ ファイルの最初のカラムとして、タイムスタンプを指定する必要があります。これは EPOCH 形式で指定する必要があります。
-?	構文の説明を表示します。
-vers	印刷バージョン情報を表示します。

注: 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。

問題のあるスクリプトや推奨されるスクリプトの例に関しては、「データ ソースの統合例」を参照してください。

dsilog によるログ処理

dsilog 処理では、ユーザーが独自のプログラムを作成するか、既存のプロセスを使用する必要があります。データを dsilog に渡すとデータをデータストアに記録できます。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 name> <class name>
```

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。

たとえば、次のようになります。

```
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 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。空白がデフォルトの区切り記号です。

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。

sdlutil によるデータの管理

DSI データベース ファイルに記録されているデータを管理するには、sdlutil プログラムを使用します。次の操作を実行できます。

- 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。
- メトリック データストア からデータソース、クラス、メトリック、データを削除します。
- バージョン情報を表示します。

構文

sdlutil <log file name>[option]

変数とオプション	定義
logfile name	データソース名として使用されます。
-rm all	メトリック データストア からデータソース、クラス、メトリック、データを削除します。
-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。

例:

クラス、メトリック、データと共にデータソース test_log を メトリック データストア から削除するには、次のコマンドを実行します。

```
sdlutil /tmp/test_log -rm all
```

第16章: メトリック データストアに記録される DSI データの使用

DSI データをメトリック データストアに記録したら、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 メトリックが定義された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 を使用してアラームを監視します。

DSI メトリックのアラーム定義

Performance Collection Component システム上の `alarmdef` ファイルを使用すると、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。`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。
- 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。詳細については、「パフォーマンス アラーム」の章の「USE 文」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、アラーム定義のカスタマイズの詳細については、「パフォーマンス アラーム」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。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 HP Operations Manager に送信できます。

DSI データのエクスポート

メトリック データストア から DSI データをエクスポートするには、extract プログラムの export 機能を使用します。詳細については、「Extract プログラムの使い方」を参照してください。エクスポート可能な DSI データソースとクラスのリストを表示するには、ovcodautil -obj コマンドを使用します。

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
- 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 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、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。

第17章: データ ソースの統合例

データ ソース統合は強力で柔軟な技術です。この章では、データを収集して メトリック データストア に記録するための DSI の使用例について説明します。

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使用される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。ファイル内の各行の内容は次の行のコメントで説明されます。

```
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、ログファイル パスを指定します。ログファイル パス内のログファイル名は、データソース名として使用されます。

注: ログファイルは作成されません。

以下のコマンドとコンパイラ出力例で、**/tmp/vmstat.spec** はクラス仕様ファイル名で、/tmp/VMSTAT_DATA はログ ファイル パスです。

```
-> sdlcomp /tmp/vmstat.spec /tmp/VMSTAT_DATA
```

この例では、VMSTAT_DATA という名前のデータソースを作成します。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、データソースは作成されません。

クラス VMSTAT_STATS が正常にデータソースに追加されたかどうかを確認するには、以下のコマンドを実行します。

```
ovcodautil -ds VMSTAT_DATA -obj
```

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 データソース内の 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 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

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。例:

```
extract -xp -l /tmp/VMSTAT_DATA -C VMSTAT_STATS
```

DSI データをエクスポートするには、ユーザーがログ ファイルの root または作成者であることが必要です。

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"
;
METRICS
who_wc = 151
label "who wc"
averaged
;
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
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 &
   ```

第V部: トランザクション追跡

HP Operations Agent は、トランザクションのアプリケーションにおける進行状況を追跡できます。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。

トランザクションは、**parm** ファイルでトランザクション ログがオンになった後でのみ、監視されます。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 セグメントに格納されます。

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ユーザーの視点で提供し、サービス レベル協定 (SLA: Service Level Agreement) の管理を支援します。さらに、サービス レベル目標 (SLO: Service Level Objective) が `alarmdef` ファイルに定義されている条件を超えると、アラームを発生させます。

第18章: 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 Agent 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ユーザー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、22 章「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 レベル目標

サービス レベル目標 (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。

23 章「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、23 章「トランザクション追跡の例」を参照してください。

ARM API コールをアプリケーションに組み込む詳細は、『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。詳細情報は、24 章「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 での使用

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 **oacore** 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、それらを開始させます (19 章「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、トランザクション構成ファイル `ttd.conf` を再度読み込みます (19 章「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 (19 章「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 (24 章「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 (20 章「アラーム」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 (19 章「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 (20 章「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(19 章「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 (「プラットフォームごとの C コンパイラ オプションの例」(290ページ) のセクションを参照)。

第19章: 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 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 エージェントの使い方については、『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。

> **注:** 『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、/<InstallDir>/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、わ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 レベル情報がわかるだけではなく、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

注: 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 文字の組み合わ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 **oacore** データ コレクタを起動すると開始します。ディスパッチされた ttd はバックグラウンド モードで実行され、エラーはエラー ファイル /var/opt/perf/status.ttd 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)。

注意: 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、/<InstallDir>/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 EPIPEerrno 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。はじめに、ARM 組み込みアプリケーションを終了させます。次に、(/<InstallDir>/bin/ovpa start または /<InstallDir>/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 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 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要求は、RPC 接続用の ttd オープン ソケット (オープン ファイル) を使用します。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。**oacore** または perfd のプロセスを実行するか GlancePlus を開始すると、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」(268 ページ) 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わからない場合、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 **oacore** 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、**oacore** 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 **oacore** 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 **oacore** 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。たとえば、次のようになります。

```
[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。

たとえば、次のようになります。

```
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」(259ページ)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ほうがより安価です。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 相関係数の詳細については、『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 (**oacore**) はディスク ログ ファイルを生成します。

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 **oacore** 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、**oacore** 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 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 GlancePlus 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 セグメントを作成します。この共有メモリ 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 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

に必ずあるとは限りません。したがって、まだメモリ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 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)。

第20章: 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 レベル目標の管理の開始に必要な情報を提供します。詳細は、「トランザクション追跡の動作」(257ページ) を参照してください。また、使用例については、「トランザクション追跡の例」(278ページ) 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「プラットフォームごとの C コンパイラ オプションの例」(290ページ)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「ARM ライブラリ (libarm)」(286ページ)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 **oacore** 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、**oacore** 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 レベル目標 (SLO) になります。サービス レベル目標は、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します。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 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、「パフォーマンス データの監視 」(274ページ)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 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。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 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 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、「トランザクション追跡の動作」(257ページ)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 **oacore** 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、**oacore** 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 エージェントで表示できる他、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、[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わかります。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。

第21章: 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。

第22章: 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 `/<InstallDir>/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 Agent Dictionary of Operating System Performance Metrics」ファイルを参照してください。

  UNIX/Linux 上の `/<InstallDir>/paperdocs/ovpa/C/` にある `met <platform>.txt`。

  Windows 上の `%ovinstalldir%paperdocs\ovpa\C` にある `met<platform>.txt`。

第23章: 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、「トランザクション追跡の紹介」(252ページ)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、「トランザクション追跡の紹介」(252ページ)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、「トランザクション追跡の動作」(257ページ)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
```

第24章: 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
- **oacore** の組み込み

データ型の使用方法

下の表は、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」セクションの「データ型の定義」の補足です。

表 3: Performance Collection Component でのデータ型の使用方法

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
例 3 NumericID32 String8 NumericID64 Gauge32 String32 Gauge64	NumericID32 NumericID64 Gauge32 Gauge64
例 4 String8 String32	(なし)

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 5	
Counter32	Counter32
Counter64	Counter64
CntrDivr32	CntrDivr32
Gauge32	Gauge32
Gauge64	Gauge64
NumericID32	NumericID32
NumericID64	

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。

第25章: 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、2 章の「トランザクション追跡デーモン (ttd)」(260ページ)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。
/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/arm64	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。

注: 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/lib/	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。

注: 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 コンパイラ オプションの例

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
  ```

注: 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 /<InstallDir>/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、/<InstallDir>/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、/<InstallDir>/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。たとえば、次のようになります。

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」(294ページ)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)。

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_GOOD.
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_ABORT.
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。	ARMConstants.ARM_FAILED.

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 を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、/<InstallDir>/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。

第VI部: トラブルシューティング

ここでは、HP Operations Agent で作業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。

運用監視コンポーネント

- 問題: HPOM が多数 (1024 超) のノードを管理している場合、HPOM と管理対象ノード間で通信の問題が発生する場合がある。また、多数 (1024 超) の管理対象ノードと通信を行う HP Performance Manager に HP Operations Agentがインストールされている場合も、この問題が発生することがある。

  対策:

  この問題を回避するには、管理サーバー (HPOM が 1024 を超えるノードを管理している場合) または HP Performance Manager サーバー (1024 を超えるノードと通信を行う HP Performance Manager サーバーにエージェントがインストールされている場合) に移動し、次の構成手順を実行します。

  root または管理者としてログオンします。

  次のコマンドを実行します。

  32 ビット版 Windows の場合:

  ```
  %ovinstalldir%bin\ovconfchg -ns xpl.net -set SocketPoll true
  ```

  64 ビット版 Windows の場合:

  ```
  %ovinstalldir%bin\win64\ovconfchg -ns xpl.net -set SocketPoll true
  ```

  HP-UX/Linux/Solaris の場合:

  ```
  /opt/OV/bin/ovconfchg -ns xpl.net -set SocketPoll true
  ```

  エージェントを再起動します。

  32 ビット版 Windows の場合:

  ```
  %ovinstalldir%bin\opcagt -stop
  %ovinstalldir%bin\opcagt -start
  ```

  64 ビット版 Windows の場合:

  ```
  %ovinstalldir%bin\win64\opcagt -stop
  %ovinstalldir%bin\win64\opcagt -start
  ```

  HP-UX/Linux/Solaris の場合:

  ```
  /opt/OV/bin/opcagt -stop
  /opt/OV/bin/opcagt -start
  ```

- 問題: Windows Server 2008 ノードで、`opcmsga` プロセスが機能せず、`ovc` コマンドを使用すると `opcmsga` プロセスのステータスが `aborted` と表示される。

対策:

以下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:ポリシーで Perl スクリプトを利用するには、PATH 設定変数に (HP Operations Agent で提供される) Perl があるディレクトリを設定する必要がある。

  対策:

  a. 以下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. 以下のコマンドを実行してエージェントを再起動します。

     i. `ovc -kill`

     ii. `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:

  以下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 2:

  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 3: 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: 特定のメッセージ インターセプタ (msgi) ポリシーを配布し、このポリシーの条件に一致するメッセージがトリガされると、opcmsgi プロセスは、高 CPU 使用率 (4-CPU サーバーで 25 % を消費するため、単一 CPU では 100 % を消費) での応答を停止し、それ以上メッセージを処理しない。

  原因:

  opcmsgi プロセスは、以下のパターンでは、応答を停止し、それ以上メッセージを処理しない。

  ```
  <*><@.variable><*>
  ```

  対策:

<*><@.variable><*> パターンを <*><@.variable><S><*> で置換します。

例:

元のパターン:

```
^\tM_Type:<@.SiSMonType><*>SiS:<
[http://<@.SiSIP>:<@>].SiSURL><*>Monitor:<@.SiSMonName><*>Group:<@.SiSGrpName><*>
Status:<@.SiSMonStatus><*>URL:<@.SiSMonURL><*>Match
Content:<@.SiSMonMatch><*>Monitor Description:DOTCOM_BACKEND:{<@.OMTargetNode>}
{<@.OMMsgGrp>}{<*.OMAppl>}{<*.OMObj>}{<*.OMSvcID>}{<@.SiSEMSCode>}{<*.OMCIs>}
{<*.OMMsgText>}
```

置換後のパターン:

```
^\tM_Type:<@.SiSMonType><S><*>SiS:<
[http://<@.SiSIP>:<@>].SiSURL><S><*>Monitor:<@.SiSMonName><S><*>Group:<@.SiSGrpNa
me><S><*>Status:<@.SiSMonStatus><S><*>URL:<@.SiSMonURL><S><*>Match
Content:<@.SiSMonMatch><S><*>Monitor Description:DOTCOM_BACKEND:
{<@.OMTargetNode>}{<@.OMMsgGrp>}{<*.OMAppl>}{<*.OMObj>}{<*.OMSvcID>}
{<@.SiSEMSCode>}{<*.OMCIs>}{<*.OMMsgText>}
```

例:

元のパターン:

```
^\tM_Type:<@.SiSMonType><*>SiS:<
[http://<@.SiSIP>:<@>].SiSURL><*>Monitor:<@.SiSMonName><*>Group:<@.SiSGrpName><*>
Status:<@.SiSMonStatus><*>URL:<@.SiSMonURL><*>Match
Content:<@.SiSMonMatch><*>Monitor Description:DOTCOM_AUTONOMY:{<@.OMTargetNode>}
{<@.OMMsgGrp>}{<*.OMAppl>}{<*.OMObj>}{<*.OMSvcID>}{<@.SiSEMSCode>}{<*.OMCIs>}
{<*.OMMsgText>}
```

置換後のパターン:

```
^\tM_Type:<@.SiSMonType><S><*>SiS:<
[http://<@.SiSIP>:<@>].SiSURL><S><*>Monitor:<@.SiSMonName><S><*>Group:<@.SiSGrpNa
me><S><*>Status:<@.SiSMonStatus><S><*>URL:<@.SiSMonURL><S><*>Match
Content:<@.SiSMonMatch><S><*>Monitor Description:DOTCOM_AUTONOMY:
{<@.OMTargetNode>}{<@.OMMsgGrp>}{<*.OMAppl>}{<*.OMObj>}{<*.OMSvcID>}
{<@.SiSEMSCode>}{<*.OMCIs>}{<*.OMMsgText>}
```

- 問題:HP Operations Agent では、パターン マッチングで間違った結果が報告される。

  原因:

  HP Operations Agent では、パターン内の OR (|) 演算子の数が 148 を超えると、パターン マッチングで間違った結果が報告される。

  > **注:** パターン マッチングでは、OR (|) 演算子は、2 つの式の文字列をマッチングすることにより、両方の式を区切る特別な文字です。
  >
  > **例:**
  >
  > `[ab|c]d`
  >
  > 上記のパターンでは、OR <|> 演算子は、文字列 **abd** と文字列 **cd** をマッチングします。

  対策:

OR (|) 演算子の数が 148 を超えないことを確認します。

- 問題: ノード上で組み込み Perl スクリプトを使用してスケジュール タスク ポリシーを実行すると、opcmona プロセスが自動的に再起動され、HPOM コンソールに次のメッセージが表示される。

  ```
  (ctrl-208) Component 'opcmona' with pid 6976 exited with exit value '-1073741819'.Restarting component.
  ```

  原因:

  組み込み Perl スクリプトの exit (0) の参照により opcmona が再起動する。

  対策:

  組み込み Perl スクリプトに exit (0) を使用しないでください。

- 問題: Red Hat Enterprise Linux システムでは、ovbbccb がタイム ゾーン ファイルにアクセスできず、日付情報をシステムに設定されている実際のタイム ゾーンではなく、UTC (GMT) 形式で書き込む。この間違ったタイム ゾーン情報が System.txt ファイルに記録される。

  対策:

  正しいタイム ゾーンを確立するには、以下の手順に従います。

  a. 以下のコマンドを実行して、zoneinfo ファイルの場所を確認します。

     ```
     ls /usr/share/zoneinfo/
     ```

  b. 以下のコマンドを実行して、localtime ファイルの場所を確認します。

     ```
     ls -l /etc/localtime
     ```

     注: "TZ" 環境変数が未設定の場合、現地時刻によって現在のタイム ゾーンが決まります。

  c. 以下のディレクトリを作成します。

     ```
     mkdir -p /var/opt/OV/usr/share/lib
     ```

  d. ファイルを 1 つのフォルダから別のフォルダに再帰的にコピーします。

     ```
     cp -rp /usr/share/zoneinfo /var/opt/OV/usr/share/lib
     ```

  e. 以下のコマンドを実行して、/var/opt/OV/ の下に etc という名前のディレクトリを作成します。

     ```
     /var/opt/OV/etc
     ```

  f. localtime ファイルをこのディレクトリにコピーします。

     ```
     cp /etc/localtime /var/opt/OV/etc
     ```

  g. 以下のコマンドを実行してエージェントを再起動します。

     i. ovc -kill

     ii. ovc -start

  注: ovbbccb は、後でエントリを System.txt ファイルに記録する際、現在のタイム ゾーンを使用します。

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、以下の手順に従います。

  a. root ユーザーとしてシステムにログオンします。
  b. 以下のコマンドを実行して、HP Operations Agent を停止します。

     ```
     /opt/OV/bin/opcagt -stop
     ```

  c. 以下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. 以下のコマンドを実行して、midaemon バイナリのページ サイズを 4K に縮小します。

     ```
     chatr +pi 4K /opt/perf/bin/midaemon
     ```

  e. 以下のコマンドを実行して、HP Operations Agent を起動します。

     ```
     /opt/OV/bin/opcagt -start
     ```

- 問題: HP Operations Agent をインストールした後、トレース機構が有効になっていると System.txt ファイルに以下のエラー 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。

- 問題: 以下のエラー メッセージが HPOM コンソールに出力される。

  ```
  CODA:GetDataMatrix returned 76='Method ScopeDataView::CreateViewEntity failed
  ```

  原因: ソースが組み込みパフォーマンス コンポーネントに設定されている Measurement Threshold ポリシーで、PROCESS オブジェクトを SCOPE データ ソースとともに使用している場合に、このエラーが出力される。

  対策: 代わりに、サービス/プロセス監視ポリシーを使用します。

- 問題: HP Performance Manager や HP Service Health Reporter といったデータ分析用製品がエージェントのデータストアからのデータの取得に失敗し、以下のエラーが表示される。

  ```
  Error occurred while retrieving data from the data source
  ```

  原因: エージェントのデータ アクセス ユーティリティは、クライアント プログラムからの単一のクエリに対して、データ クラスのすべてのレコードを読み込む。クエリは、HP Performance Manager といったデータ分析クライアントによってエージェントに送信される。データ クラスに大量のレコードが含まれていると、データ アクセス ユーティリティがクエリの処理に失敗する。

  対策: この問題を回避するには、データ レコードを複数のチャンクで転送するようデータ アクセス ユーティリティを設定します。以下の手順に従ってください。

  a. root または管理者として、エージェント ノードにログオンします。
  b. 次のコマンドを実行します。

Windows の場合:

```
%ovinstalldir%bin\ovconfchg -ns coda -set DATAMATRIX_VERSION 1
```

HP-UX/Linux/Solaris の場合:

```
/opt/OV/bin/ovconfchg -ns coda -set DATAMATRIX_VERSION 1
```

AIX の場合:

```
/usr/lpp/OV/bin/ovconfchg -ns coda -set DATAMATRIX_VERSION 1
```

各クエリに対して、エージェントのデータ アクセス ユーティリティがデータを 5 つのレコードのチャンクに分割し、クライアント プログラムにデータを送信します。データを分割すると、データ転送プロセスのパフォーマンスが向上します。

エージェントが 1 つのチャンクでクライアントに送信できるレコードの数を制御できます。DATAMATRIX_ROWCOUNT 変数 (coda 名前空間で使用可能な変数) により、この数を制御できます (デフォルト値は 5)。

DATAMATRIX_ROWCOUNT 変数の値を減らすと、データストアに大量のデータがある場合に、データ転送率がわずかに向上することがあります。

DATAMATRIX_ROWCOUNT 変数を 0 に設定した場合は、HP Operations Agentはチャンクなしでデータ レコードを送信するデフォルトの動作に戻ります。

ただし、DATAMATRIX_ROWCOUNT 変数のデフォルト設定を変更しないことをお勧めします。

c. エージェントを再起動して設定変更を有効にします。

```
ovc -restart coda
```

RTMA

- 問題: HP-UX 11.11 システムで status.perfd ファイルに次のエラーが出力される。

  ```
  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、以下の手順に従います。

  a. root ユーザーとしてシステムにログオンします。

  b. 以下のコマンドを実行して、HP Operations Agent を停止します。

  ```
  /opt/OV/bin/opcagt -stop
  ```

  c. 以下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. 以下のコマンドを実行して、perfd バイナリのページ サイズを 4K に縮小します。

  ```
  chatr +pi 4K /opt/perf/bin/perfd
  ```

  e. 以下のコマンドを実行して、HP Operations Agent を起動します。

  ```
  /opt/OV/bin/opcagt -start
  ```

HP GlancePlus

問題: HP GlancePlus に、AIX フレーム上でホストされているすべての LPAR インスタンスが表示されない。

原因: HP Operations Agent をインストールした LPAR が、AIX フレーム上でホストされているその他の LPAR と通信できない。

対策: HP Operations Agent をインストールした LPAR が、AIX フレーム上でホストされているその他すべての LPAR と通信できることを確認します。

HP Operations Agent をホストしている LPAR 上で次のコマンドを実行して、その他の LPAR との接続性を確認します。

`xmpeek -l`<hostname>

ここで、<hostname> は LPAR のホスト名です。

hpsensor

- 問題: hpsensor で常駐メモリの高使用率が報告される。

  原因: 割り当て済みの内部 STL データ構造が、削除後にもクリーンアップされない。この問題は主に HPUX プラットフォームで発生する。

  対策: この問題を解決するには、以下の手順に従います。

  a. 管理者としてシステムにログオンします。
  b. 以下のディレクトリに移動します。
     - Windows の場合: `%OvDataDir%hpcs\`
     - UNIX/Linux の場合: `/var/opt/OV/hpcs/`
  c. **hpcs.conf** ファイルを開き、**hpcs.runtime** 名前空間で次の設定変数を **true** に設定します。

     MemMap=<true>

その他

- 問題: `fs_type` メトリックで、autofs システムが NFS (Network File System) として報告される。

  原因: HP Operations Agent 12.00 が実行されている Linux システムでは、autofs システムは、カーネル 2.6 以前のバージョンであれば、NFS として報告される。

  対策: この問題を解決するには、autofs システムがカーネル 3.0 以降のバージョンであることを確認します。
- 問題: **oacore** プロセスが、次のエラー メッセージを `System.txt` ファイルに記録後に停止する。

```
Database is in an inconsistent state.
```

原因: 対応するデータベース ファイルがない管理対象クラス情報が存在する場合、**oacore** プロセスはエラー メッセージを `System.txt` ファイルに記録後に停止する。

注: Global クラスの `System.txt` メッセージの例:

```
0:INF:Thu Aug 20 16:14:28 2015: oacore (21580/139960775325472):Collection
intervals:Process = 60 Global = 300  DataFile Rollover% = 20.
```

```
0:ERR:Thu Aug 20 16:14:28 2015: oacore (21580/139960775325472):Database is in
an inconsistent state.No database found for Class  Scope::Global
```

```
0:INF:Thu Aug 20 16:14:28 2015: oacore (21580/139960775325472):(oacore-84)
oacore. oacore Server stopped.
```

対策: この問題を解決するには、データソースまたはデータソースの特定のクラスを削除する必要があります。

データソースまたはデータソースの特定のクラスを削除するには、`oadbutil.pl` ツールを使用します。

構文:

- `oadbutil.pl -d <datasource name>`
  データソースをそのクラスと共に削除します。

- `oadbutil.pl -d <datasource name> -c <class name>`
  指定したクラスのメタ データおよびレコードのみ削除します (データソースに関する情報は保持されます)。

例:

```
oadbutil.pl –d Scope –c Global
```

ここで、

`<datasource name>` は Scope

`<class name>` は Global

デフォルトでは、データ コレクタは、デフォルトのパフォーマンス メトリック クラスとデータソース (Scope) をメトリック データストアに記録するモデルを作成しません。デフォルトのパフォーマンス メトリック クラスまたはデータソースを削除したら、以下の手順に従ってモデルを再作成します。

a. 以下のコマンドを実行して、**oacore** プロセスを停止します。

   ```
   ovc -stop oacore
   ```

b. 以下のコマンドを実行して、クラスを再作成します。

   ```
   ovconfchg -ns oacore -set UPDATED_MODEL_AVAILABLE true
   ```

c. 以下のコマンドを実行して、**oacore** プロセスを起動します。

```
ovc -start oacore
```

注: カスタム クラス (Perl ベースのスクリプト可能な API または DSI 経由で送信される) が削除されている場合、次回の送信時に自動的に再作成されます。

第VII部: HP Operations Agent の以前のバージョンから 12.00 へのアップグレード

HP Operations Agent バージョン 12.00 では、ログ ファイルに基づくデータストアに代わって、メトリック データストア が使用されます。CODA および scope プロセス (UNIX および Linux 上の scopeux、Windows 上の scopent) は、**oacore** プロセスと呼ばれる 1 つのプロセスに統合されています。**oacore** プロセスは、システム パフォーマンスおよびカスタム データに対する読み取りと書き込みの両方のインターフェイスを提供します。

CODA データベース ファイル、SCOPE ログ ファイル、DSI ログ ファイルに記録されている古いデータは、読み取り専用モードで保持されます。この古いデータには、ovcodautil、extract といったユーティリティ、または HP Performance Manager や HP Reporter といったレポート作成ツールを使用してアクセスできます。

注:

- HP Operations Agent 11.xx からの古いデータは、メトリック データストア に移行できません。
- HPUX IA システムで、HP Operations Agent 11.xx から 12.00 にアップグレードすると、古いデータベース ファイルはバックアップされ、次の場所に保存されます。

  `/var/opt/OV/tmp/BackUp`

  このデータには、Extract、Utility、`ovcodautil` などのツールでアクセスすることはできません。
- Linux または Windows システムでは、HP Operations Agent 12.00 にアップグレードすると、HP Performance Manager や HP Reporter などのツールを使用して、古い論理システム データ (ログファイルに保存されているデータ) にアクセスできなくなります。

以前のバージョンから HP Operations Agent 12.00 へのアップグレード中、DSI ログ ファイルからの既存のすべてのデータ、すべてのクラスとメトリックとともに CODA データストアに記録された Smart Plug-in (SPI) データが メトリック データストア 内に自動的に作成されます。

DSI データソース

HP Operations Agent 11.xx から 12.00 にアップグレードする前に、以下に保存されている `datasources` ファイルに DSI データストアが追加されていることを確認します。

HP-UX/Linux/Solaris の場合:

`var/opt/OV/conf/perf`

Windows の場合:

`%ovdatadir%\conf\perf`

DSI データソースを datasources ファイルに追加するには、以下のコマンドを実行します。

```
DATASOURCE=<Datasource Name>LOGFILE=<DSI Logfile Path>
```

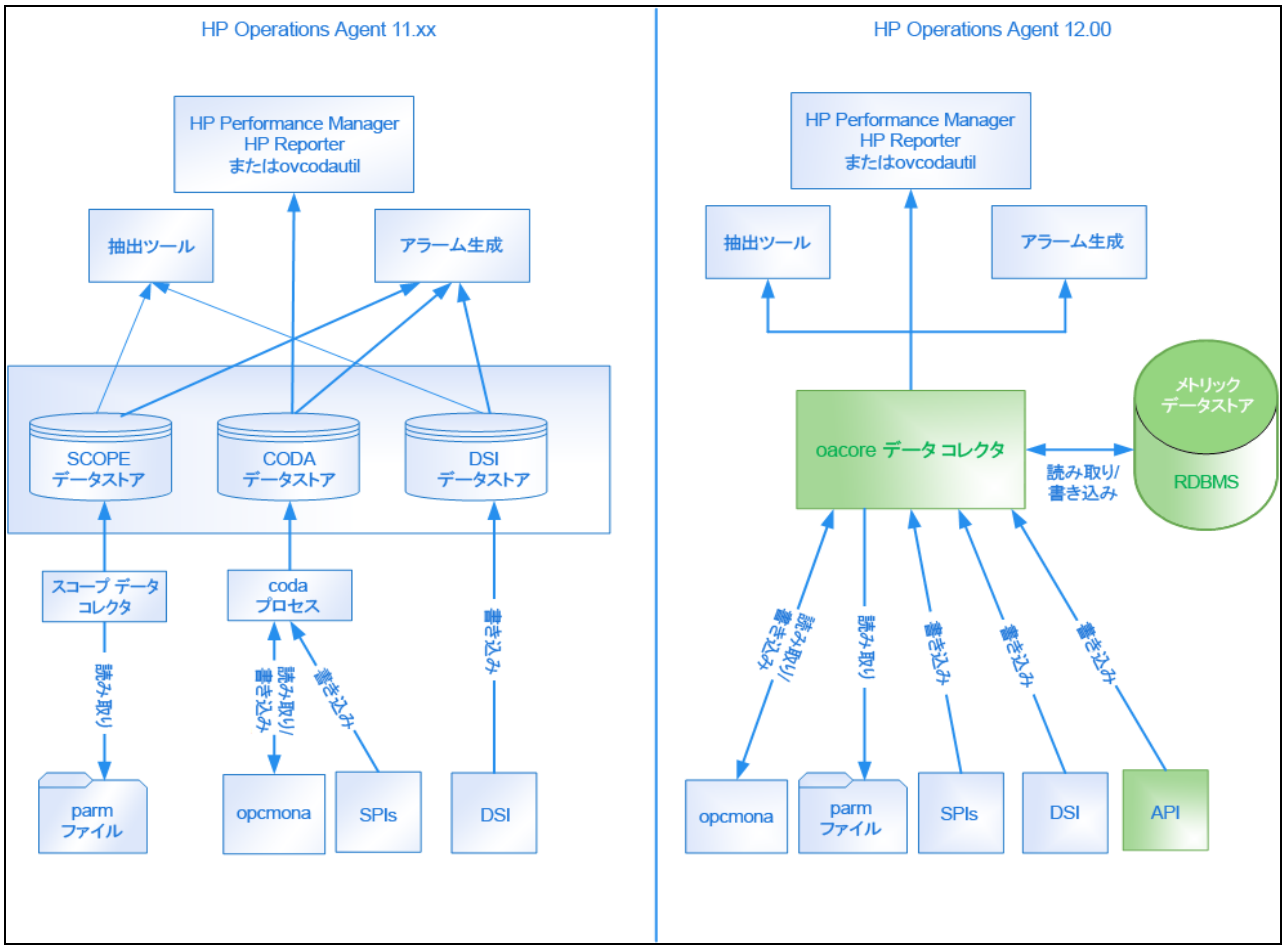
ログファイル名とデータ ソース名が同じであることを確認します。

例:

```
DATASOURCE=VMSTAT LOGFILE=/root/dsi/VMSTAT
```

DSI データソースを datasources ファイルに追加した場合にのみ、メトリック データストア 内に DSI データを記録するためのモデルが作成されます。

HP Operations Agent 12.00 と以前のバージョンの比較



HP Operations Agent 12.00 と以前のバージョンの比較

新機能	HP Operations Agent 11.xx	HP Operations Agent 12.00
データ コレクタ	**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、ログ ファイルに基づくデータストアにデータを記録します。 収集したデータは、parm ファイルに指定したデータ クラスに応じて、次のログ ファイルに記録されます。logglob、logappl、logproc、logpcmd、logdev、logtran、logls、logindx。	**oacore** データ コレクタはシステム全体のパフォーマンスおよび状況データを継続的に収集し、収集したデータを メトリック データストア に記録します。 **注:** CODA および **scope** プロセス (scopeux および scopent) は、**oacore** と呼ばれる 1 つのプロセスに統合されています。**oacore** プロセスは、システム パフォーマンスおよびカスタム データに対する読み取りと書き込みの両方のインターフェイスを提供します。
scope の RUN ファイル	`/var/opt/perf/datafiles/RUN` にある **scope** の RUN ファイルは、**scope** が実行されているかどうかを示します。このファイルは、**scope** (scopent および scopeux) が実行されているときにのみ存在します。 **scope** が異常終了した場合、このファイルは削除されません。**scope** を再開するには、**scope** の RUN ファイルを削除して **scope** を開始する必要があります。	このファイルは提供されません。 **oacore** データ コレクタのステータスを確認するには、「oacore プロセスのステータスの検証」を参照してください。
データストア	監視対象システムから収集したメトリック データは、ログ ファイルに基づくデータストアに記録されます。	監視対象システムから収集したシステム パフォーマンスとヘルス データは メトリック データストア に記録されます。メトリック データストア は RDBMS (Relational Database Management System) に基づく 1 つのデータストアです。 メトリック データは、**parm** ファイルに指定したデータ クラスに基づいて収集されます。メトリック データストアに記録されるデータのすべてのクラスに対して、固有のデータベース ファイルが作成されます。

HP Operations Agent 12.00 と以前のバージョンの比較 (続き)

新機能	HP Operations Agent 11.xx	HP Operations Agent 12.00

プロキシ データソース	さまざまなシステムからの複数のデータベース ファイルをデータソースとしてホストできます。HP Operations Agent 11.xx はこれらのデータソースから同期に読み取ることができます。	HP Operations Agent バージョン 12.00 を使用する場合、一度に 1 つのデータソースだけを読み取り専用モードでホストできます。
		次の手順を実行して、異なるシステムからのデータベース ファイルをデータソースとしてホストします。
		1. 以下のコマンドを実行して、**oacore** プロセスを停止します。
		`ovc -stop oacore`
		2. 既存のデータベース ファイルをバックアップします
		3. 次のコマンドを実行します。
		`ovconfchg -ns oacore.dml -set READ_ONLY_MODE True`
		4. リモート システムからのデータベース ファイルを、`datadir/databases/oa` に置きます。
		5. 以下のコマンドを実行して、**oacore** プロセスを起動します。
		`ovc -start oacore`
		注: プロキシ モードに設定すると、メトリック データストア へのデータ ログ記録は完全に無効になります。
		次の手順を実行して、異なるシステムからのデータベース ファイルのデータソースとしてのホスティングを停止します。
		1. 以下のコマンドを実行して、**oacore** プロセスを停止します。
		`ovc -stop oacore`
		2. 次のコマンドを実行します。
		`ovconfchg -ns oacore.dml -clear READ_ONLY_MODE`
		3. 以下のコマンドを実行して、**oacore** プロセスを起動します。
		`ovc -start oacore`

HP Operations Agent 12.00 と以前のバージョンの比較 (続き)

新機能	HP Operations Agent 11.xx	HP Operations Agent 12.00
rtmd プロセス	RTM コンポーネントにより提供される rtmd プロセスは、ノードからリアルタイム データへアクセスする安全な通信チャネルの確立に役立ちます。	rtmd プロセスは hpsensor プロセスに置き換えられました。rtmd の XPL 設定は以前のバージョンと互換性がなく、HP Operations Agent を 11.xx から 12.00 にアップグレードすると動作しなくなります。hpsensor プロセスには、セキュリティ (SSL) を使用するための同様の XPL 設定があります。 hpsensor プロセスを使用すると、リアルタイムのパフォーマンス メトリックに、セキュアな通信チャネルを通じてローカルまたはリモートでアクセスできます。

Performance Collection Component

HP Operations Agent 12.00 と以前のバージョンの比較

新機能	HP Operations Agent 11.xx	HP Operations Agent 12.00
Performance Collection Component のグラフィカル ユーザー インターフェイス	Performance Collection Component のグラフィカル ユーザー インターフェイスを使用すると、次のようなタスクを実行できます。ログ ファイル データの抽出、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 ファイルのスキャン、ログ ファイルの解析、エクスポート テンプレートの設定、ユーザー オプションの設定、収集パラメータの設定、アラーム定義の設定、およびステータスの確認。	Performance Collection Component のグラフィカル ユーザー インターフェイスは、サポートされていません。
拡張収集ビルダ/マネージャ	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。拡張収集ビルダ/マネージャ (ECBM) を使用して、特定のパフォーマンス カウンタを選択し、データ収集を構築することができます。	この機能はサポートされません。
ovcodautil (記録	ovcodautil -ds SCOPE -o Global -m GBL_CPU_TOTAL_UTIL	ovcodautil -ds SCOPE -o Global -m GBL_CPU_

HP Operations Agent 12.00 と以前のバージョンの比較 (続き)

新機能	HP Operations Agent 11.xx	HP Operations Agent 12.00
されたメトリックの表示に使用されるコマンドラインユーティリティ)	/opt/OV/bin/ovcodautil [options] -h:メトリック ヘッダーを表示	TOTAL_UTIL /opt/OV/bin/ovcodautil [options] -header:メトリック ヘッダーを表示 -h:ヘルプ メッセージを表示

parm ファイル

HP Operations Agent 12.00 と以前のバージョンの比較

新機能	HP Operations Agent 11.xx	HP Operations Agent 12.00
スコープ トランザクション	**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。scope トランザクション 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。 これらの 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。	HP Operations Agent12.00 では、Scope_Get_Process_Metrics と Scope_Get_Global_Metrics は記録されません。
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。	HP Operations Agent バージョン 12.00 では mainttime パラメータがサポートされていません。 size パラメータは、データベース ファイルの最大サイズ

HP Operations Agent 12.00 と以前のバージョンの比較 (続き)

新機能	HP Operations Agent 11.xx	HP Operations Agent 12.00
	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。 **注:** 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。	を (メガバイト単位で) 設定するために使用されます。**parm** ファイルに指定された最大サイズに達した場合、デフォルトのパフォーマンス メトリック クラスを記録するデータベース ファイルはロール オーバーされます。 **parm** ファイル内の size 指定を変更すると、**oacore** の開始時にその変更が検出されます。 **parm** ファイルに size が指定されていない場合、データベース ファイルは最大サイズの 1GB に達したときにロール オーバーされます。ロール オーバーの際には、古い方から 20% のデータがデータベース ファイルから削除されます。
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。 **注:** データ収集中にログ ファイルのロール	HP Operations Agent バージョン 12.00 では Days パラメータがサポートされていません。 size パラメータは、データベース 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。**parm** ファイルに指定された最大サイズに達した場合、デフォルトのパフォーマンス メトリック クラスを記録するデータベース ファイルはロール オーバーされます。 **parm** ファイル内の size 指定を変更すると、**oacore** の開始時にその変更が検出されます。

HP Operations Agent 12.00 と以前のバージョンの比較 (続き)

新機能	HP Operations Agent 11.xx	HP Operations Agent 12.00
	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。	**parm** ファイルに size が指定されていない場合、データベース ファイルは最大サイズの 1GB に達したときにロールオーバーされます。ロールオーバーの際には、古い方から 20% のデータがデータベース ファイルから削除されます。
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、maintime 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週の日に maintweekday の値を設定することをお勧めします。 **注:** 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	HP Operations Agent バージョン 12.00 では Maintweekday パラメータがサポートされていません。 size パラメータは、データベース 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。**parm** ファイルに指定された最大サイズに達した場合、デフォルトのパフォーマンス メトリック クラスを記録するデータベース ファイルはロール オーバーされます。 **parm** ファイル内の size 指定を変更すると、**oacore** の開始時にのみその変更が検出されます。 **parm** ファイルに size が指定されていない場合、データベース ファイルは最大サイズの 1GB に達したときにロールオーバーされます。ロールオーバーの際には、古い方から 20% のデータがデータベース ファイルから削除されます。

HP Operations Agent 12.00 と以前のバージョンの比較 (続き)

新機能	HP Operations Agent 11.xx	HP Operations Agent 12.00
	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 maintime 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。	このパラメータは、Windows と UNIX のプラットフォームで有効です。
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 Agent データストアにプロセス 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。	このパラメータはすべてのプラットフォームでサポートされています。 proccmd パラメータを使用すると、データストアにプロセス コマンドを記録できます。デフォルトでは、このプロセスの値は 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。プロセス コマンドの記録を有効にするには、このパラメータの値を 1 に設定します。 proccmd パラメータの記録は、このパラメータの値が 1 以上の場合にオンになります。記録されるプロセス コマンドの長さは、このパラメータに指定された値に関わりなく、常に 4096 です。

Utility プログラム

HP Operations Agent 12.00 と以前のバージョンの比較

新機能	HP Operations Agent 11.xx	HP Operations Agent 12.00
utility プログラムの実行	HP Operations Agent11.xx では、utility プログラムの実行は、バッチ モード、対話型モードおよびコマンド ライン モードでサポートされます。	HP Operations Agent 12.00 では、utility プログラムの実行はコマンド ライン モードでのみサポートされます。
resize コマンド	サイズ変更コマンド -xr は、使用しているログ 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、このコマンドを使用する必要があります。	HP Operations Agent バージョン 12.00 ではサイズ変更コマンドがサポートされていません。 size パラメータは、データベース 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。**parm** ファイルに指定された最大サイズに達した場合、デフォルトのパフォーマンス メトリック クラスを記録するデータベース ファイルはロールオーバーされます。 **parm** ファイル内の size 指定を変更すると、**oacore** の開始時にその変更が検出されます。 ロール オーバーの際には、**parm** ファイルに指定されたサイズを超える、古い方から 20% のデータがデータベース ファイルから削除されます。
utility のコマンド: start stop exit guide logfile menu sh show	これらのコマンドがサポートされており、以下のように使用できます。 • start - 分析または走査関数の開始日時を指定します。 • stop - 分析または走査関数の終了日時を指定します。 • 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。 • guide - guide コマンドは、ガイド コマンド モードに移行すると	これらのコマンドはサポートされていません。

HP Operations Agent 12.00 と以前のバージョンの比較 (続き)

新機能	HP Operations Agent 11.xx	HP Operations Agent 12.00
	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。 • 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。 • 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。 • 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。 • 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。	

utility 走査レポート

HP Operations Agent 11.xx から 12.00 にアップグレードする場合、utility 走査レポートは、データの各クラスが使用するディスク容量に関する情報、およびログファイル セットに保存されている従来のデータに関する情報を提供します。

例:

utiltiy -xs コマンドを実行すると、以下の図に示すように、クラス要約レポートおよびログ ファイルの内容の要約レポートの両方が生成されます。

クラス要約レポート

```
                              utility -xs
      CLASSNAME                                      RECORDS      STARTTIME             ENDTIME              HH:MM:SS
---------------------------------------------------------------------------------------------------------------------
SCOPE::APPLICATION                                       73 2015/07/21 14:03:39  2015/07/21 15:50:00  1:46:21
SCOPE::TRANSACTION                                       46 2015/07/21 14:03:39  2015/07/21 15:50:00  1:46:21
SCOPE::PROCESS                                            4 2015/07/21 14:03:39  2015/07/21 15:42:00  1:38:21
SCOPE::CPU                                               23 2015/07/21 14:03:39  2015/07/21 15:50:00  1:46:21
SCOPE::FILESYSTEM                                       207 2015/07/21 14:03:39  2015/07/21 15:50:00  1:46:21
SCOPE::NETIF                                             46 2015/07/21 14:03:39  2015/07/21 15:50:00  1:46:21
SCOPE::CORE                                               0 0000/00/00 00:00:00  0000/00/00 00:00:00  00:00:00
SCOPE::GLOBAL                                            23 2015/07/21 14:03:39  2015/07/21 15:50:00  1:46:21
SCOPE::DISK                                               7 2015/07/21 14:03:39  2015/07/21 15:50:00  1:46:21
SCOPE::LVOLUME                                            0 0000/00/00 00:00:00  0000/00/00 00:00:00  00:00:00

**Data is now logged into Perf dataStore.(oa.db)
**Below is the information for older readonly logfile set.

The total time covered was      :        07:47 out of 07:47
Time lost when collector was off:        00:00   0.00%
The scope collector was started :            1 times
```

ログ ファイルの内容の要約レポート

```
            ------Total------  --Each Full Day-- -------Dates-------   Full
Type        Records MegaBytes  Records MegaBytes  Start        Finish   Days
Global            2      0.00    370.0     0.197 07/21/15 to 07/21/15    0.0
Application       6      0.00   1552.1     0.205 07/21/15 to 07/21/15    0.0
Disk              2      0.00    517.4     0.043 07/21/15 to 07/21/15    0.0
NETIF             4      0.00   1034.7     0.087 07/21/15 to 07/21/15    0.0
CPU               2      0.00    517.4     0.060 07/21/15 to 07/21/15    0.0
Filesystem        4      0.00   1034.7     0.132 07/21/15 to 07/21/15    0.0
Tran              4      0.00    576.0     0.323 07/21/15 to 07/21/15    0.0
Overhead                 0.03
            ------- ---------  ------- ---------
TOTAL            24      0.03   5602.3     1.047

The Global      file is now   0.0% full with room for 152.4 more full days
The Application file is now   0.0% full with room for  97.6 more full days
The Device      file is now   0.0% full with room for  61.9 more full days
The Transaction file is now   0.1% full with room for  31.0 more full days
```

ログ ファイルの内容の要約レポートには、古い読み取り専用のログファイル セットに保存されている情報がリストされます。

例: `utiltiy -xs -D` コマンドを実行すると、次のレポートが生成されます。

初期 parm ファイルのグローバル情報

HP Operations Agent 11.xx

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セクションです。

```
04/01/2014 12:38 System ID="  "
SCOPE/UX C.05.00.100 COLLECTION INTERVALS:Process = 60, GLOBAL = 300 Seconds, Log
version=D
Configuration: ia64, O/S Linux 2.6.16.21-#1 SMP Monia64 #1 SMP Monia64 CPUs=1
Logging Global(NoDisk)(NoNETIF) Application Process Transaction
        Device = Disk NETIF FileSys CPU records

Thresholds:CPU=10.00%, Disk=n/a, ProcMem=900.0 MB
Nonew=TRUE, Nokilled=TRUE, Shortlived=FALSE (<1 sec)

Javaarg = true

Parms:Buffer Cache Size = 0KB, NPROC = 0

Memory:Physical = 981 MB, Swap = 1286.4 MB,
Available to users = 853 MB

Application and device data flush frequency = 3600 seconds
01/02/2014 09:22 Data collected on 3 disk devices:
    Disk #0              = "sda"
    Disk #2              = "sdb"
    Disk #3              = "hda"
```

ここで、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、データ コレクタ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、必ずしもその最初のグローバル レコードであるとは限りません。

HP Operations Agent 12.00

このレポートは、HP Operations Agent 11.xx から 12.00 にアップグレードした場合にのみ生成されます。

初期 parm ファイルのアプリケーション定義

HP Operations Agent 11.xx

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。

例:

```
10/29/2013 15:25 Application(1) = "other"
Comment=all processes not in user-defined applications

10/29/2013 15:25 Application(2) = "network"
File=nfs*,biod,automount,amd,*inetd,snmp*,rpc*,llbd,netfmt,portmap
File=rbootd,telnet*,ftp*,*rlogin*,remsh*

File=rcp,nktl*,nvsisr,ttisr,lcsp,gcsp,strmen,strweld,vtdaemon
File=mib*,trapdest*,*web*,xntpd,yp*

File=hp_unixagt,ntl*,pty*

10/29/2013 15:25 Application(3) = "xwindows"
File=X*,xload,xclock,*term,grmd,softmsg*,vue*,ttsession,pexd
File=dt*,xfs,rpc.ttdbserver,Aserver,gdm*,kdm*

File=gnome*,kde*,soffice.bin,netscape*,mozilla*,realplay

10/29/2013 15:25 Application(4) = "memory_management"
File=swapper,vhand,syncer,pageout,fsflush,kswapd,kreclaimd,bdflush
```

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。

HP Operations Agent 12.00

このレポートは、HP Operations Agent 11.xx から 12.00 にアップグレードした場合にのみ生成されます。

parm ファイルのグローバル変更通知

HP Operations Agent 11.xx

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*"
```

HP Operations Agent 12.00

このレポートは生成されません。

parm ファイルのアプリケーション変更内容

HP Operations Agent 11.xx

parm ファイルのアプリケーション変更内容レポートを取得するには、scan コマンドをデフォルトの detail on と共に使用し、データストアにアプリケーション データを記録します。

parm ファイルに何らかの変更を行った場合、データストアにその変更内容を記録するために、**scope** を再起動する必要があります。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
```

メモ: 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。

HP Operations Agent 12.00

このレポートは生成されません。

scope のオフタイム通知

HP Operations Agent 11.xx

このレポートには、scope を再開始した日時、および scope を再開始する前のログ ファイルに最後の有効なデータを記録した日時が示されます。

HP Operations Agent 12.00

このレポートは生成されません。

アプリケーション固有の要約レポート

HP Operations Agent 11.xx

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、または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ことができます。

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ます。これにより、変更前と変更後のアプリケーションの定義を表示できます。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、最後のレポート生成時以降の期間であり、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。

例:

```
 Percent of Total

Application          Records      CPU   Disk  Trans

-------------------- ---------- ------ ------ ------

other                     26258   3.7%   6.6% 100.0%
network                    4649   0.0%   0.4%   0.0%
xwindows                  34571  25.3%   1.9%   0.0%
memory_management          3797   0.0%   0.0%   0.0%
other_user_root           45504  71.0%  91.0%   0.0%

-------------------- ---------- ------ ------ ------

All user applications     77.1%  96.3%  93.4%   0.0%
```

HP Operations Agent 12.00

このレポートは、HP Operations Agent 11.xx から 12.00 にアップグレードした場合にのみ生成されます。

プロセス要約レポート

HP Operations Agent 11.xx

このレポートは、プロセス データが走査された場合、常に出力されます。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見なされたすべての理由と、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:11/08/2013 10:18    to 05/05/2014 16:55
There were 3427.4 hours of process data
Process records were logged for the following reasons:

Log Reason      Records   Percent  Recs/hr
--------------- -------   -------  -------
New Processes         6      0.1%      0.0
Killed Processes      6      0.1%      0.0
CPU Threshold      6956    100.0%      2.0

NOTE: A process may be logged for more than one reason at a time.
Record counts and percentages will not add up to 100% of the process records.
```

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、最後のレポート生成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 データの量を削減するには、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。

HP Operations Agent 12.00

このレポートは生成されません。

走査の開始および終了レポート

HP Operations Agent 11.xx

走査の開始および終了レポートは、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。

例:

```
Scan started on 11/08/2013 10:18
Scan stopped on 05/05/2014 16:55
```

HP Operations Agent 12.00

このレポートは、HP Operations Agent 11.xx から 12.00 にアップグレードした場合にのみ生成されます。

アプリケーションの全体的な要約

HP Operations Agent 11.xx

このレポートを生成するには、HP Operations Agent データストア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。

例:

```
Overall, user defined applications account for
88521 out of 114779 records ( 77.1%)
832.2 out of 863.9 CPU hours ( 96.3%)
819.2 out of 877.2 M disk IOs ( 93.4%)
 0.0 out of 0.0 M trans ( 0.0%)
```

HP Operations Agent 12.00

アプリケーションの全体的な要約レポートは、HP Operations Agent 11.xx から 12.00 にアップグレードした場合にのみ生成されます。このレポートには、CODA データベース ファイル、scope ログ ファイル、DSI ログ ファイルの従来のデータのみが含まれています。

コレクタの適用範囲の要約

HP Operations Agent 11.xx

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。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。

例:

```
The total time covered was :172/13:29:27 out of 178/06:37:00
Time lost when collector was off: 5/17:07:33 3.20%
The scope collector was started : 5 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 の開始時刻および終了時刻を最も近い時刻に決定します(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)。

HP Operations Agent 12.00

このレポートは、HP Operations Agent 11.xx から 12.00 にアップグレードした場合にのみ生成されます。

> **注: oacore** プロセスの開始時刻と終了時刻の詳細は、System.txt ファイルにリストされています。

クラス要約およびログ ファイルの内容の要約

HP Operations Agent 11.xx

クラス要約レポートは、HP Operations Agent 11.xx ではサポートされていません。

ログ ファイルの内容の要約レポートは、有効なデータが走査された場合に出力され、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。

例:

```
            ------Total------  --Each Full Day-- -------Dates-------   Full
Type        Records MegaBytes  Records MegaBytes  Start       Finish   Days
Global        7460      3.97   1440.1     0.766 07/03/2015 to 07/08/2015    5.2
Application   9161      1.21   1768.7     0.233 07/03/2015 to 07/08/2015    5.2
Process      14920      4.45   2880.5     0.858 07/03/2015 to 07/08/2015    5.2
Disk          7451      0.63   1438.5     0.121 07/03/2015 to 07/08/2015    5.2
NETIF        14920      1.25   2880.5     0.242 07/03/2015 to 07/08/2015    5.2
CPU          29840      3.46   5761.1     0.668 07/03/2015 to 07/08/2015    5.2
Filesystem   37300      4.77   7201.3     0.922 07/03/2015 to 07/08/2015    5.2
Tran         14921      8.36   2880.2     1.613 07/03/2015 to 07/08/2015    5.2
Overhead                0.58
            ------- ---------  ------- ---------
TOTAL       135973     28.67  26250.9     5.424

The Global      file is now  13.3% full with room for  33.9 more full days
The Application file is now   6.3% full with room for  80.3 more full days
The Process     file is now  15.0% full with room for  29.7 more full days
The Device      file is now  52.4% full with room for   4.9 more full days
The Transaction file is now  84.2% full with room for   1.0 more full days
```

HP Operations Agent 12.00

クラス要約レポートは、HP Operations Agent 12.00 を新しくインストールした場合のみ生成されます。データの各クラスが使用するディスク容量に関する情報を提供します。

HP Operations Agent 11.xx から 12.00 にアップグレードすると、クラス要約レポートおよびログ ファイルの内容の要約レポートの両方が生成されます。

ログ ファイルの空きスペースの要約

HP Operations Agent 11.xx

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。

例:

```
The Global      file is now  73.9% full with room for   0.3 more full days
The Application file is now  78.2% full with room for  21.8 more full days
The Process     file is now   7.0% full with room for  93.0 more full days
The Device      file is now  96.4% full with room for   3.1 more full days
The Transaction file is now  90.4% full with room for   3.0 more full days
```

追加データを格納できる空き容量は、次の要素に基づいて算出されます。

- 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。

HP Operations Agent 12.00

ログ ファイルの空きスペースの要約レポートは、HP Operations Agent 11.xx から 12.00 にアップグレードした場合にのみ生成されます。このレポートには、CODA データベース ファイル、scope ログ ファイル、DSI ログ ファイルの従来のデータのみが含まれています。

Extract

HP Operations Agent 12.00 と以前のバージョンの比較

新機能	HP Operations Agent 11.xx	HP Operations Agent 12.00
extract プログラムの実行	HP Operations Agent 11.xx では、extract プログラムの実行は対話型モードとコマンド ライン モードでサポートされます。	HP Operations Agent 12.00 では、extract プログラムの実行はコマンド ライン モードでのみサポートされます。
出力ファイル	extract プログラムはエクスポート機能を実行します。このプログラムは、ログ ファイルからデータを読み取り、結果を ASCII、データファイル、バイナリ、WK1 (表計算シート) の各形式にエクスポートします。	Extract プログラムはエクスポート機能を実行します。このプログラムは、データストアからデータを読み取り、結果を ASCII 形式の出力ファイルにエクスポートします。
Host Bus Adapter (HBA)	このメトリック クラスは、HP Operations Agent 11.xx では使用できません。	新しいメトリック クラス Host Bus Adapter (HBA) が追加されました。HBA データをエクスポートするには、次のコマンドを使用します。 • -h - エクスポートする HBA 詳細データを指定します。 • -H - エクスポートする HBA 要約データを指定します。
extract のコマンド: -s -k -ut -v -we -xt -xw -xm -xy	これらのコマンドがサポートされており、以下のように使用できます。 • -s- 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。 • -k - 抹消されたプロセスのみを抽出します。 • -ut - エクスポートされたログ 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。 • -v - 冗長出力レポート フォーマットを生成します。	これらのコマンドはサポートされていません。

HP Operations Agent 12.00 と以前のバージョンの比較 (続き)

新機能	HP Operations Agent 11.xx	HP Operations Agent 12.00
	• -we - 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。 • -xt - データをシステム内部フォーマットに抽出します。 • -xw - 月曜日から日曜日までの 1 週間のデータを抽出します。 • -xm - 暦月のデータを抽出します。 • -xy - 暦年のデータを抽出します。	
ファイル オプション: Append、Purge、New	サポートされています	サポートされていません
エクスポート オプション: TODAY、TOMORROW、TODAY-1 TOMORROW-1、Last、First	サポートされています	Last と First オプションのみは、-b と -e コマンド オプションと共に使用するときにサポートされます。
エクスポートされるデータ形式: ASCII、DATAFILE、WK1 または SPREADSHEET、BINARY	サポートされています	ASCII 形式のみサポートされています
エクスポート テンプレート ファイル: Repthist、Reptall	サポートされています	Reptall テンプレート ファイルのみサポートされています。
レイアウト: 単一、複数	サポートされています	単一の零案とのみがサポートされています。
エクスポート ファイルのタイトル: !date、!time、!logfile、!class、 !collector、!system_id	サポートされています	サポートされていません

メトリック

HP Operations Agent 12.00 と以前のバージョンの比較

新機能	HP Operations Agent 11.xx	HP Operations Agent 12.00
BLANK RECORD_TYPE DATE TIME YEAR DAY DATE_SECONDS INTERVAL STATDATE STATTIME	これらのメトリックはサポートされています。	これらのメトリックは、データの任意のクラスに対してサポートされていません。
トランザクション追跡メトリック	トランザクション追跡メトリックには、監視システムで実行されるすべてのシステム トランザクションに関連するメトリックが含まれます。このメトリック クラスの接頭辞は TTBIN_ または TT_ です。	トランザクション追跡メトリックには、監視システムで実行されるすべてのシステム トランザクションに関連するメトリックが含まれます。このメトリック クラスの接頭辞は TT_ です。 次のメトリックはサポートされなくなりました。 TT_USER_MEASUREMENT_NAME TT_USER_MEASUREMENT_MAX TT_USER_MEASUREMENT_MIN TT_USER_MEASUREMENT_AVG TT_USER_MEASUREMENT_NAME_2 TT_USER_MEASUREMENT_MAX_2 TT_USER_MEASUREMENT_MIN_2 TT_USER_MEASUREMENT_AVG_2 TT_USER_MEASUREMENT_NAME_3 TT_USER_MEASUREMENT_MAX_3

HP Operations Agent 12.00 と以前のバージョンの比較 (続き)

		TT_USER_MEASUREMENT_MIN_3 TT_USER_MEASUREMENT_AVG_3 TT_USER_MEASUREMENT_NAME_4 TT_USER_MEASUREMENT_MAX_4 TT_USER_MEASUREMENT_MIN_4 TT_USER_MEASUREMENT_AVG_4 TTBIN_UPPER_RANGE_10 TTBIN_UPPER_RANGE_1 TTBIN_UPPER_RANGE_2 TTBIN_UPPER_RANGE_3 TTBIN_UPPER_RANGE_4 TTBIN_UPPER_RANGE_5 TTBIN_UPPER_RANGE_6 TTBIN_UPPER_RANGE_7 TTBIN_UPPER_RANGE_8 TTBIN_UPPER_RANGE_9 TTBIN_TRANS_COUNT_10 TTBIN_TRANS_COUNT_1 TTBIN_TRANS_COUNT_2 TTBIN_TRANS_COUNT_3 TTBIN_TRANS_COUNT_4 TTBIN_TRANS_COUNT_5 TTBIN_TRANS_COUNT_6 TTBIN_TRANS_COUNT_7 TTBIN_TRANS_COUNT_8 TTBIN_TRANS_COUNT_9 TT_NUM_BINS
Host Bus Adapter (HBA)	このメトリック クラスは、HP Operations Agent 11.xx では使用できません。	HBA は、他のメトリック クラスと共に記録されるメトリックの新しいクラスです。監視システムで稼働するすべてのホスト バス アダプタに関連するメトリックが含まれます。このメトリック

HP Operations Agent 12.00 と以前のバージョンの比較 (続き)

		クラスの接頭辞は BYHBA_ です。 次の設定メトリックがログ記録されます。 BYHBA_TIME BYHBA_INTERVAL BYHBA_ID BYHBA_NAME BYHBA_DEVNAME BYHBA_DEVNO BYHBA_CLASS BYHBA_DRIVER BYHBA_STATE BYHBA_UTIL BYHBA_THROUGHPUT_UTIL BYHBA_IO BYHBA_READ BYHBA_WRITE BYHBA_IO_RATE BYHBA_READ_RATE BYHBA_WRITE_RATE BYHBA_BYTE_RATE BYHBA_READ_BYTE_RATE BYHBA_WRITE_BYTE_RATE BYHBA_REQUEST_QUEUE BYHBA_BUSY_TIME BYHBA_AVG_WAIT_TIME BYHBA_AVG_SERVICE_TIME BYHBA_TYPE

HP Operations Agent 12.00 と以前のバージョンの比較 (続き)

BYLS メトリック	このメトリックはサポートされています。	BYLS メトリックは、Windows および Linux プラットフォームではサポートされていません。 HP Operations Agent 12.00 は、Xen、KVM、VMware vSphere、Hyper-V、および他の仮想化ドメインから BYLS メトリックを収集しません。

SNMP トラップ インターセプタ

HP Operations Agent 12.00 と以前のバージョンの比較

新機能	HP Operations Agent 11.xx	HP Operations Agent 12.00
opctrapi プロセス	**opctrapi** プロセスは、SNMPv1 および SNMPv2 トラップを傍受するように設定されています。	**opctrapi** プロセスは、SNMPv1、SNMPv2、SNMPv3 トラップおよび通知メッセージを傍受するように設定されています。

SNMP_SESSION_MODE 変数	SNMP_SESSION_MODE 変数を使用すると、SNMP トラップ インターセプタを構成し、外部ソースから発信された SNMP トラップを待ち受けるためにポート 162 にバインドするさまざまなメカニズムを使用することができます。
	• SNMP トラップを待ち受ける Net-SNMP API を使用してポート 162 にバインドする SNMP トラップ インターセプタを構成するには、以下のコマンドを実行します。
	`ovconfchg -ns eaagt set SNMP_SESSION_MODE NETSNMP`
	• Windows ノードのみ。SNMP トラップの待ち受けに Microsoft トラップ サービスを使用する SNMP トラップ インターセプタを構成するには、以下のコマンドを実行します。
	`ovconfchg -ns eaagt set SNMP_SESSION_MODE WIN_SNMP`
	• UNIX/Linux ノードのみ。SNMP トラップの待ち受けに OVSNMP API を使用してポート 162 にバインドする SNMP トラップ インターセプタを構成するには、以下のコマンドを実行します。
		SNMP_SESSION_MODE 変数はサポートされていません。
		HP Operations Agent 12.00 では、NETSNMP モードのみがサポートされています。

HP Operations Agent 12.00 と以前のバージョンの比較 (続き)

新機能	HP Operations Agent 11.xx	HP Operations Agent 12.00

	`ovconfchg -ns eaagt set SNMP_SESSION_MODE NO_TRAPD`
	• SNMP トラップの待ち受けに OVSNMP API を使用してポート 162 にバインドする SNMP トラップ インターセプタを構成するには、以下のコマンドを実行します。
	`ovconfchg -ns eaagt set SNMP_SESSION_MODE NNM_LIBS`
	• SNMP トラップを待ち受けるのに、NNM (7.5x) の PMD デーモンに登録を試みるよう SNMP トラップ インターセプタを構成するには、以下のコマンドを実行します。
	`ovconfchg -ns eaagt set SNMP_SESSION_MODE TRY_BOTH`
	• UNIX/Linux ノードのみ。SNMP トラップを待ち受けるのに、NNM (7.5x) の PMD デーモンに登録するよう SNMP トラップ インターセプタを構成するには、以下のコマンドを実行します。
	`ovconfchg -ns eaagt set SNMP_SESSION_MODE NNM_PMD`	

データ ソース統合

HP Operations Agent 12.00 と以前のバージョンの比較

新機能	HP Operations Agent 11.xx	HP Operations Agent 12.00
データ収集	DSI で収集したデータは、ログ ファイルに格納されます。	DSI で収集したデータは、メトリック データストア に格納されます。データストアに記録されるデータのすべてのクラスに対して、固有のデータベース ファイルが作成されます。 **注:** DSI がカスタム データを収集するためには、**oacore** が実行されている必要があります。 ただし、以前のバージョンとの互換性のため、コマンド ラインでは引き

HP Operations Agent 12.00 と以前のバージョンの比較 (続き)

新機能	HP Operations Agent 11.xx	HP Operations Agent 12.00
		続きログファイル引数がサポートされます。ログファイル名はパスから抽出され、データ ソース名と見なされます。 `sdlcomp <class specication file> <logfile name>` HP Operations Agent 11.xx から 12.00 にアップグレードした場合、メタデータまたはデータ記録のフォーマットは古いログ ファイルから メトリック データストア にコピーされます。 **注:** DSI コンパイラ **sdlcomp** は、ログファイル名を基にファイルを作成しません。代わりにログファイル名はデータソース名として使用されます。 DSI ログ ファイルに記録された古いデータへのアクセスについては、「DSI データソース」を参照します。
データのロールオーバー	`INDEX BY`、`MAX INDEXES` および `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。	`INDEX BY`、`MAX INDEXES`、`ROLL BY` はサポートされていません。 カスタム データを記録するデータベース ファイルの最大サイズは、デフォルトで 1 GB に設定されています。このサイズは設定できません。 メトリック データストアに記録されているデータは、データベース ファイルが最大サイズの 1 GB に達したときに自動的にロール オーバーされます。 ロールオーバーの際には、最も古いデータの 20% が削除されます (最も古いデータベース ファイルは完全に削除されます)。

HP Operations Agent 12.00 と以前のバージョンの比較 (続き)

新機能	HP Operations Agent 11.xx	HP Operations Agent 12.00
ACTION	ACTION パラメータで記述されるコマンドは、ロール オーバー前に実行されます。	このパラメータはサポートされません。
CAPACITY	CAPACITY は、クラスに保存されるレコード数です。	サポートされていません
sdlutil 構文	`sdlutil logfile_set [option]` ここで、logfile_set は、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。 変数とオプション: • -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。 • -rm all	`sdlutil <log file name>[option]` 変数とオプション: • -rm all 指定したデータ ソースのメトリック データストアから、データ ソース、クラス、メトリック、データを削除します。 • -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。

HP Operations Agent 12.00 と以前のバージョンの比較 (続き)

新機能	HP Operations Agent 11.xx	HP Operations Agent 12.00
	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。	
PRECISION	PRECISION はメトリック値の小数点以下の桁数を識別します。	サポートされていません
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。	サポートされていません
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 (クラスの仕様を参照)、クラス内の各メトリックについてデータを作成します。	サポートされていません

よくある質問 (FAQ)

アップグレード後に古いデータが存在していることを、どのように確認したらいいのですか?

以下の手順に従ってください。

1. HP Operations Agent 12.00 にアップグレード後に、root 権限でログオンします。
2. 次のコマンドを実行します。

   ```
   <OvInstallBinDir>ovcodautil -obj
   ```

3. すべてのデータソース、クラスおよびメトリックで構成されるリストが生成されます。

アップグレード後に古いログ ファイルをどのように削除したらいいのですか?

以下の手順に従って、古いログ ファイルを削除します。

1. root 権限で HP Operations Agent 12.00 にログオンします。
2. 以下のコマンドを実行して、oacore プロセスを停止します。

   ```
   ovc -stop oacore
   ```

3. 次のファイルを削除します。

   Windows の場合:

   ```
   <OvDataDir>/conf/perf/datasources
   <OvDataDir>/datafiles/coda*
   <OvDataDir>/datafiles/log*
   ```

   HP-UX/Linux/Solaris の場合:

   ```
   <OvDataDir>/conf/perf/datasources
   <OvDataDir>/datafiles/coda*
   /var/opt/perf/datafiles/log*
   ```

4. 以下のコマンドを実行して、oacore プロセスを起動します。

   ```
   ovc -start oacore
   ```

メトリック データストア は、どのように再作成したらいいのですか?

以下の手順に従って、メトリック データストアを再作成します。

1. 以下のコマンドを実行して、**oacore** プロセスを停止します。

   ```
   ovc -stop oacore
   ```

2. オプション: 既存のデータベース ディレクトリ `<OvDataDir>databases/oa/` をバックアップします。
3. 以下のコマンドを実行して、データストアを削除します。

   Windows の場合:

   ```
   del /F "<OvDataDir>databases\oa\*"
   "<OvInstallBinDir>\sqlite3.exe" "%OvDataDir%databases\oa\oa.db"
   <"%OvDataDir%conf\oa\Model\DMLMetaMetaSchema"
   ```

   HP-UX/Linux/Solaris/AIX の場合:

   ```
   rm -f <OvDataDir>/databases/oa/*
   <OvInstallBinDir>/sqlite3 /var/opt/OV/databases/oa/oa.db
   <OvDataDir>/conf/oa/Model/DMLMetaMetaSchema
   ```

4. 以下のコマンドを実行して、UPDATED_MODEL_AVAILABLE 設定変数を True に設定します。

   ovconfchg -ns oacore -set UPDATED_MODEL_AVAILABLE TRUE

5. 以下のコマンドを実行して、**oacore** プロセスを起動します。

   ```
   ovc -start oacore
   ```

注: 次回の送信時に、カスタム クラス (送信用の API または DSI 経由で送信される) が自動的に再作成されます。

ドキュメントのフィードバックを送信

本ドキュメントについてのご意見、ご感想については、電子メールでドキュメント制作チームまでご連絡ください。このシステムで電子メールクライアントが設定されていれば、このリンクをクリックすることで、以下の情報が件名に記入された電子メールウィンドウが開きます。

Feedback on ユーザー ガイド (Operations Agent 12.00)

本文にご意見、ご感想を記入の上、[送信] をクリックしてください。

電子メールクライアントが利用できない場合は、上記の情報をコピーしてWebメールクライアントの新規メッセージに貼り付け、docfeedback@hp.com 宛にお送りください。

お客様からのご意見、ご感想をお待ちしています。