ソフトウエアエージェント設定

ソフトウエアエージェント設定

Pandora FMS ドキュメント一覧に戻る

ソフトウエアエージェントとは?

OS にインストールされ、監視情報を抽出して Pandora FMS (のデータを処理して DB へ保存する)データサーバへ定期的に送信するための小さなソフトウエアです。

エージェント設定概要

ソフトウェアエージェントの動作 (動作パラメータとモジュール) は、その設定ファイルによって決定されます。

- MS Windows® の場合:

```
%ProgramFiles%\pandora_agent\pandora_agent.conf
```

- GNU/Linux® システムの場合

```
/etc/pandora/pandora_agent.conf
```

エージェントの一般的なパラメータ

パラメータのほとんどは MS Windows® および GNU/Linux® システムで共通です。 一般パラメータを変更した後は、ソフトウェアエージェントを再起動する必要があります。

エージェント設定ファイルのエンコードは、GNU/Linux® システムと MS Windows® システム共に UTF-8 です。ただし、日本語 Windows の場合は Shift_JIS の利用を推奨します。

ソフトウェア エージェントから初めてデータを受信すると、すべての情報がデータベースに保存されます。 連続した情報の送信では (学習モードが有効かどうかに応じて)、XML の次のフィールドのみが更新されます: version timestamp (日付) および os_version (オペレーティングシステムのバージョン)、また、次の設定ファイルパラメータ: gis_exec latitude longitude altitude parent_agent_name timezone_offset address custom_field

server_ip

データを保持する Pandora FMS サーバのホスト名もしくは IP アドレスを設定します。

server_path

エージェントから送られるデータを受け取るサーバのディレクトリパスを設定します。通常は、/var/spool/pandora/data_in です。

temporal

エージェントがサーバにデータを送信する前にローカルで利用するフォルダのパスを設定します。

description

XML でエージェントが送る「説明」を設定します Pandora FMS はエージェントを作成する時に、この「説明」を取り込みます。

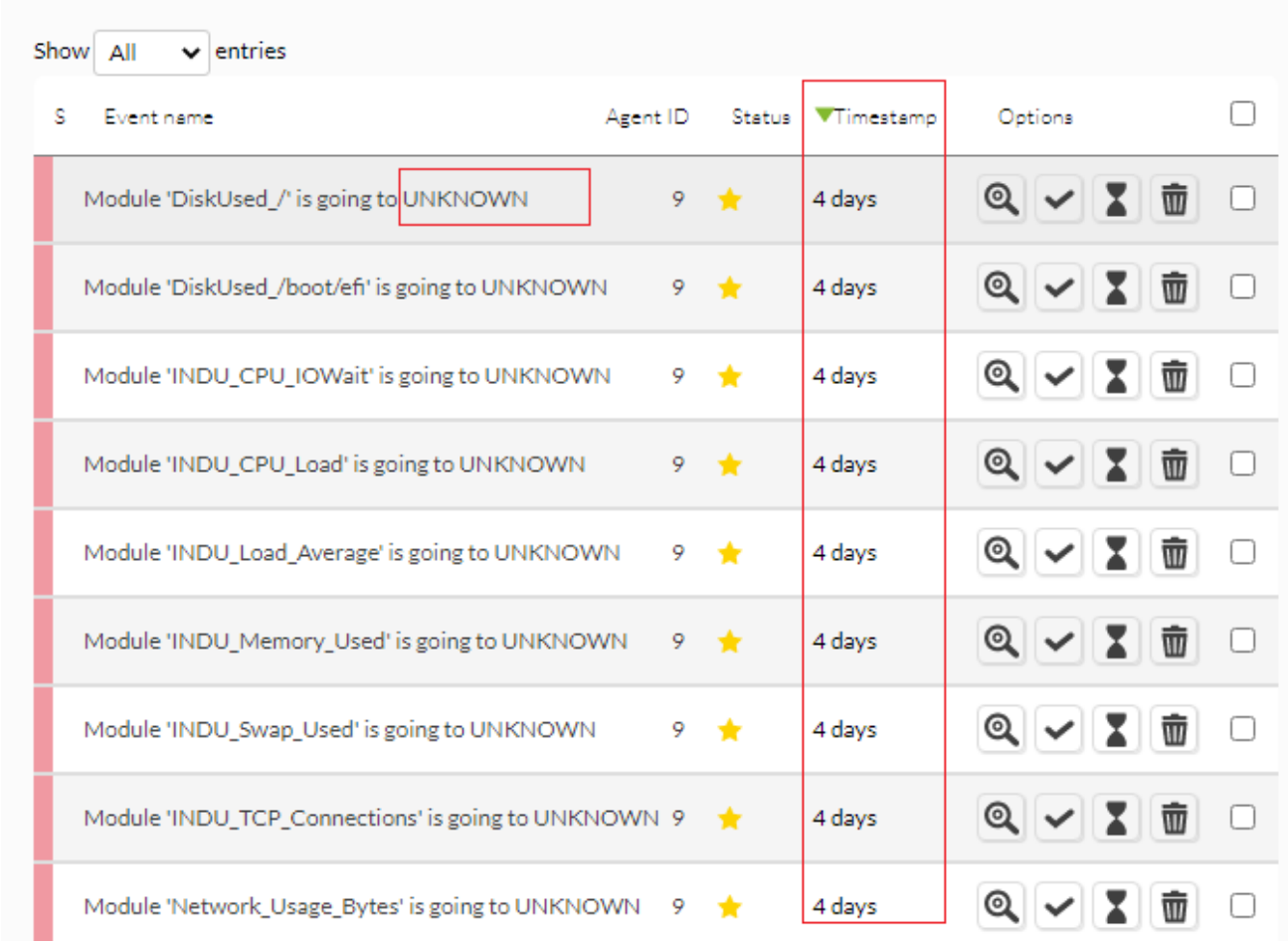
group

エージェントが所属するグループ名を設定します。このパラメータで指定された名前のグループがある場合、サーバが特定のグループにエージェントを作成するようにしていなければ、指定したグループ内にエージェントが作成されます。

temporal_min_size

テンポラリディレクトリのパーティション空き容量がここで指定したサイズ (MB 単位) より小さくなったら、データパケットの生成を停止します。 (デフォルトは 1MB) 何らかの理由でプライマリおよびセカンダリの Pandora FMS サーバへの接続が長時間切れた場合に、ディスクがいっぱいになるのを避けます。

Pandora FMS イベント表示で、原因が何であれ、ソフトウェアエージェントがメインおよびセカンダリ Pandora FMS サーバへのデータ配信を停止したかどうかをタイムリーに知ることができます。この場合、その状態は 不明 になります。必要に応じて、アラート条件の作成も参照してください。



4日間オフラインのこの例では、1メガバイト(または設定した値)の空きディスク容量が残るまで、監視データが保存されます。

上記の理由により、ソフトウェアエージェントが改めてプライマリおよびセカンダリ Pandora FMS サーバに接続するときに、蓄積された量のデータでPandora FMS データサーバの高負荷が発生する可能性があることに注意が必要です。

temporal_max_size

XML バッファの最大サイズ(メガバイト)です。デフォルトは 1024 です。

temporal_min_size も確認してください。

temporal_max_files

XML バッファの最大ファイル数です。デフォルトは 1024 です。

temporal_min_size も確認してください。

logfile

Pandora FMS エージェントのログファイルのパスです。

interval

エージェントの秒単位のデータ収集間隔です。この間隔でエージェントは情報を収集し、Pandora FMS サーバへ送ります。

disable_logfile

このパラメータは、pandora_agent.log へのログ出力を無効化します Windows のみ。

debug

有効化すると、エージェントのデータファイルはテンポラリディレクトリ内に名前を変えて保存され、サーバへ送信したあとも削除されません XMLファイルを開いて内容を確認することができます。

agent_name

カスタム名を設定できます。設定していない場合、エージェント名はマシンのホスト名になります。

agent_name_cmd

外部コマンドを使ってエージェント名を定義したい場合に設定します。これはオプションです。このパラメータを設定した場合 'agent_name' は無視されます。外部コマンドはエージェント名を標準出力に返す必要があります。複数行を返した場合、1行目がエージェント名として扱われます。

agent_alias_cmd

バージョン NG 7 以上

外部コマンドを使ってエージェントの別名を定義したい場合に設定します。このパラメータを設定した場合 'agent_alias' は無視されます。外部コマンドはエージェントの別名を標準出力に返す必要があります。複数行を返した場合、1行目がエージェントの別名として扱われます。

address

これは、ソフトウエアエージェントの IP アドレスです。X.X.X.X というフォーマットの IP アドレス。'localhost' のようなホスト名、または 'auto' を設定できます。IP アドレスまたはホスト名を指定した場合は、そのアドレスがエージェントに追加され、メインのアドレスとなります。'auto' を指定した場合は、ホストから IP アドレスを取得してエージェントに追加されます。

encoding

ISO-8859-15 や UTF-8 のように、システムのエンコーディングの種類を設定します。

server_port

このパラメータは、リモートのサーバの待ち受けポートの設定です。Tentacle のデフォルトは 41121 です。

transfer_mode

このパラメータには、エージェントからサーバへデータを送信するための方法を設定します。デフォルトは Tentacle です。

transfer_timeout

バージョン 6.0 以上

データ転送プログラムの実行タイムアウトを秒単位で指定します。設定されていない場合のデフォルトは 30 です。

server_pwd

Windows の FTP もしくは Tentacle 転送モードのパスワードを設定します。Tentacle モードの場合はパスワードは必須ではありません。このパスワードでサーバでの認証を行います。

server_ssl

Tentacle の転送モードを設定します。SSL による暗号化を有効化(yes)または、無効化(no)できます。Tentacle の暗号化通信については、こちらの章も参照してください。

server_opts

Tentacle の転送モードを設定します。Tentacle クライアントの拡張設定のために渡す、追加パラメータを設定します。

バージョン 3.2 以降のエージェントでは、tentacle がサーバへデータを送信するのに HTTP プロキシをサポートしています。プロキシを利用するには、つぎのような拡張オプションを使います。

```
server_opts "-y user:pass@proxy.inet:8080"
```

この例では、tentacle クライアントがポート 8080、ユーザ “user”、パスワード “pass” でプロキシ 'proxy.inet' に接続します。プロキシサーバが 192.168.1.2 で、ポート番号が 9000、また、認証が無い場合は次のようにします。

```
server_opts "-y 192.168.1.2:9000"
```

delayed_startup

デフォルトでは無効です。 エージェント起動後、処理を開始するまでの待ち時間(秒 または 分)です。MS Windows 以外のソフトウエアエージェントで利用できます。

startup_delay

デフォルトでは無効です。 エージェント起動後、処理を開始するまでの秒単位の待ち時間です。MS Windows のみ。

pandora_nice

このパラメータには、Pandora FMS エージェントプロセスの優先順位を設定します。この設定は、Unix/Linux エージェントにのみ存在します。

autotime

これを有効に設定 (1 に設定) すると、エージェントが送信する時間データを無視し、サーバに接続した時間をサーバのローカル時間で認識するようになります。この設定は、何らかの理由によりエージェントの時刻がおかしかったり、サーバの時刻と大きくずれている場合に利用します。

cron_mode

このパラメータを有効にすると、エージェントをそれ自身の実行の仕組みではなく、Linux の crontab から指定した時刻に実行できるようになります。デフォルトでは無効になっています。

remote_config

(Pandora FMS Enterprise のみ)

リモートエージェント設定を有効化(1)または無効化(0)します。Tentacle 転送モードでのみ利用できます。

xml_buffer

有効化(1)すると、エージェントは接続障害等でサーバに XML ファイルを送信できなかったとき、それをテンポラリディレクトリに保存します。それらは、通信が復旧したときに送られます。

timezone_offset

エージェントではサーバのタイムゾーンとのオフセットを設定できます。これは、エージェントが異なるタイムゾーンのサーバで稼働しているとき、同じ時間に合わせる場合にとても便利です。エージェントは、調整したタイムゾーンでサーバに送信します。

```
# Timezone offset: Difference with the server timezone
timezone_offset 3
```

サーバのタイムゾーンからエージェントのタイムゾーンを差し引いて計算されます。例えば、サーバのタイムゾーンが UTC+1 で、エージェントのタイムゾーンが UTC-5 であれば、タイムゾーンのオフセットは、6 = 1 - (-5) です。

parent_agent_name

ソフトウエアエージェントの親を指定します。Pandora FMS に存在するエージェント名である必要があります。

agent_threads

モジュールを並行して実行するエージェントのスレッド数です。デフォルトではシングルスレッドで、すべての処理完了まで 1つずつモジュールを実行します。これは、UNIX エージェントのみの機能です。

```
# Number of threads to execute modules in parallel
agent_threads 4
```

include

```
include <file>
```

別の設定<ファイル>をインクルードすることができます。このファイルには、メインの設定ファイルに加えて追加のモジュールや収集を含めることができます。これはオプションです。ファイルのアップロードには、ユーザにエージェント書き込み（AW□のプロファイル が必要です。

broker_agent

```
broker_agent < broker_name >
```

ブローカーエージェント機能を有効化します。有効化するにはパラメータのコメントを外し、ブローカーエージェントに割り当てる名前 (< broker_name >) を指定する必要があるだけです。メインの設定ファイルに記載したブローカーエージェントの名前で新たな設定ファイルが生成されます。この設定はメインのエージェントでのみ利用でき、作成された新たなエージェント設定内では利用できません。

pandora_user

```
pandora_user <user>
```

このパラメータはオプションで、システムの特定のユーザ (<user>) でエージェントを実行させるためのものです。指定するユーザは、エージェントの実行権限等を持つ必要があります。

custom_id

外部アプリケーションのための、エージェントのカスタム ID です。

url_address

コンソールでエージェントから開くカスタム URL です。

custom_fieldX_name

システムに設定済のエージェントのカスタムフィールド名です。存在しない場合は無視されます。

例:

```
custom_field1_name Model
```

custom_fieldX_value

前述のパラメータで定義されたカスタムフィールド X の値です。

例:

```
custom_field1_value C1700
```

module_macro

Unix/Linux 用ソフトウエアエージェント

```
module_macro<_macro_name_ > < value >
```

モジュールの定義で利用可能なローカル実行マクロを定義します。これらのマクロは、メタコンソールシステムとローカルモジュールコンポーネントシステムで使用され、コードを直接編集することで難しくなるモジュール定義を「抽象化」し、高度なユーザに値を「入力」できるローカルインターフェイスを提供します。 これらの値は、ローカルプラグインのマクロシステムに比較的似たマクロシステムを使用して、以下のように使用できます。

fieldx で始まるローカル拡張マクロ。

例:

```
module_begin
module_name Particion_opt
module_type generic_data
module_exec df -kh _field1_ | tail -1 |  awk '{ print $5}' | tr -d "%"
module_macro_field1_ /opt
module_end
```

group_password

```
group_password <password>
```

エージェントグループのパスワードです。グループのパスワード保護をしない場合は、コメントアウトした状態にしてください。

ehorus_conf

```
ehorus_conf <path>
```

有効な eHorus エージェント設定ファイルへの絶対パス。エージェントは、eHorus エージェントの識別キーを含む eHorusID という名前のカスタムフィールドを作成します。

transfer_mode_user

```
transfer_mode_user <user>
```

ローカル転送モードでコピーされるファイルのユーザでえす。正しく動作するためには、コンソールのフォルダで、このユーザがリモート設定ファイルを読み書きできる権限が必要です。デフォルトは apache です。

secondary_groups

```
secondary_groups <group name1>, <group name2>, ... <group nameN>
```

エージェントに割り当てるセカンダリグループ名です。複数のセカンダリグループをカンマ区切りで指定できます。サーバ上に存在しないグループを指定した場合はグループは割り当てられませんが、エージェントの作成には影響ありません。

standby

```
standby <1|0>
```

エージェントがスタンバイモード(standby 1)の場合、エージェントは監視を実行せず XML の生成や送信を行いません。この設定は、リモート設定がある Enterprise 版の利用時に意味があります。これにより、エージェントを無効化した場合に動作を停止することができます。

デバッグモードの場合はこの機能を上書きし、エージェントは通常動作をします。

module_absoluteinterval

```
module_absoluteinterval <interval>[s,m,h,d]
```

```
module_absoluteinterval once
```

モジュールの実行間隔を指定しますが、module_interval とは異なります。

- エージェントを再起動するときに、最後の実行日を記憶します。 モジュールは、指定された間隔が経過するまで実行されません。 - 間隔を秒、分、時間、または日単位で指定できます (例: 30s□5m□1h□7d)。 - 間隔値として once を指定することで、モジュールが 1 回だけ実行されるように設定できます。

セカンダリサーバ

データを送信するセカンダリサーバを設定することができます。設定により二種類の動作モードがあります。

- on_error: プライマリサーバにデータを送信出来なかった場合のみ、セカンダリサーバにデータを送信します。
- always: プライマリサーバにデータが送信できるかどうかに関わらず、常にセカンダリサーバにもデータを送信します。

UDP サーバ

UDPは本質的に安全ではないことに注意してください(ただし、応答を必須としないメッセージを送信するには効率的です)。

Pandora FMS エージェントは、 リモートからコマンドを受け取る設定ができます。このサーバは、ユーザが指定した UDP のポートで待ち受けており、アラートが発生したときに何らかのコマンドを実行するなど、リモートシステム (基本的には Pandora FMS) から命令を受け取ることができます。

UDP リモートサーバの設定には、いくつかのオプションがあり、*pandora_agent.conf* にて設定します。

- udp_server:UDP サーバを有効にする場合は 1 を設定します。デフォルトでは無効です。
- udp_server_port: 待ち受けポート番号を設定します。
- udp_server_auth_address: 命令の送信を許可する IPアドレスを設定します。カンマ区切りで複数のアドレスを指定できます。0.0.0.0 に設定すると UDP サーバはすべてのアドレスからのリクエストを受け付けます。

すべてのソースからのコマンド受け入れのために 0.0.0.0 に設定できますが、この方法はお勧めしません。複数の Pandora FMS サーバがある場合、または IPv6 を使用している場合は、カンマで区切って異なる IP アドレスを追加できます。たとえば IPv6:2001:0db8:0000:130F:0000:0000:087C:140B があり、その省略形が 2001:0db8:0:130F::87C:140B の場合は、両方をカンマで区切って指定します。

- process_<name>_start <command>: ユーザ定義プロセスを起動するコマンドを設定します。
- process_<name>_stop <command>: プロセスを停止するコマンドを設定します。
- service_<name> 1: リモートから停止や起動ができるサービス名 <name> を設定します。

設定例:

```
udp_server 1
udp_server_port 4321
udp_server_auth_address 192.168.1.23
process_firefox_start firefox
process_firefox_stop killall firefox
service_messenger 1
```

サーバは、次のコマンドを受け付けます。

```
* **<START|STOP> SERVICE <サービス名>**: サービスの起動や停止を行います。
* **<START|STOP> PROCESS <プロセス名>**: プロセスの起動や停止を行います。
* **REFRESH AGENT <エージェント名>**: エージェントの強制実行および、データの更新を行います。
```

例:

```
STOP SERVICE messenger
START PROCESS firefox
REFRESH AGENT 007
```

サーバの /util/udp_client.pl に Pandora FMS サーバでアラート発生時にプロセスやサービスを起動できる実行コマンドとして利用できるスクリプトがあります。次のような書式で利用します。

```
./udp_client.pl <address> <port> <command>
```

例えば、エージェントを再起動するには次のようにします。

```
./udp_client.pl 192.168.50.30 41122 "REFRESH AGENT"
```

モジュール定義

設定ファイル pandora_agent.conf にてローカルモジュールの実行を定義します。利用可能な全パラメータを以下に説明します。一般的な書式は次の通りです。

```
module_begin
module_name <モジュール名>
module_type generic_data
module_exec <実行するローカルコマンド>
module_end
```

全モジュールで共通の項目

モジュールのフィールド情報はモジュール作成時にのみ反映されます(モジュールデータ、説明、拡張情報を除く)。モジュールがすでに存在する場合は更新されません。

module_begin

モジュール定義の開始を示す、必須項目です。

module_name

モジュールの名前です(必須)。重複した名前は設定できません。

```
module_name <名前>
```

module_type

```
module_type <タイプ>
```

モジュールが返すデータタイプです。いずれか一つを選択することが必須です。指定可能なデータタイプは次の通りです。

- Numerical (generic_data): 小数または整数の数値データです。小数値の場合は整数に切り捨てられます。
- Incremental (generic_data_inc): 前の値と現在の値の差分を間隔の時間で割ったデータです。差分がマイナスの場合は値はリセットされます。これは、差分は再びプラスになること、増分がプラスの値である限りは前のデータとの差が利用されることを意味します。
- Incremental absolute (generic_data_inc_abs): 前の値と現在の値の差分データです。絶対的な差分の値であり、単位時間あたりの差分ではありません。差分がマイナスの場合は値はリセットされます。差分が再びプラスになった場合はベースの値として利用され、そこから差分が計算されます。
- Alphanumeric (generic_data_string): 文字列です。
- Booleans (generic_proc): 正常(1)または異常(0)をとる値です。コンピュータが動いているか、プロセスやサービスが動いているかどうかのチェックに便利です。異常値(0)には障害状態があらかじめ設定されています。それよりも大きい任意の値(1より大きい数字も扱えます)は正常と判断されます。
- Asynchronous Alphanumeric (async_string): (定期実行ではない)非同期の文字列収集に使います。非同期監視は、任意のタイミングで発生するイベントや変更に依存するため、このタイプのモジュールは不明状態にはなりません。
- Async Boolean (async_proc): generic_proc と似ていますが、非同期のデータタイプです。
- Asynchronous Numeric (async_data): generic_data と似ていますが、非同期のデータタイプです。

module_min

```
module_min <値>
```

そのモジュールが返すことを許容されるデータの最小値です。範囲を外れたデータはサーバにより削除されます。

module_max

```
module_max <値>
```

そのモジュールが返すことを許容されるデータの最大値です。範囲を外れたデータはサーバにより削除されます。

module_min_warning

```
module_min_warning <値>
```

モジュールが警告状態になる最小値です。

module_max_warning

```
module_max_warning <値>
```

モジュールが警告状態になる最大値です。

module_min_critical

```
module_min_critical <値>
```

モジュールが障害状態になる最小値です。

module_max_critical

```
module_max_critical <値>
```

モジュールが障害状態になる最大値です。

module_disabled

```
module_disabled <値>
```

モジュールが、有効(0)か無効(1)かを表します。

module_min_ff_event

```
module_min_ff_event <値>
```

連続抑制回数を指定します。連続抑制回数とは、収集データに揺らぎがあるような場合に、それを変化としてとらえないように抑止するものです。

module_each_ff

```
module_each_ff <0|1>
```

有効化(1)すると□module_min_ff_event の代わりに連続抑制回数に個別状態変化(module_min_ff_event_normal, module_min_ff_event_warning および module_min_ff_event_critical)を利用します。

module_min_ff_event_normal

```
module_min_ff_event_normal <値>
```

個別状態変化で、正常状態へ移行する場合の連続抑制回数です。

module_min_ff_event_warning

```
module_min_ff_event_warning <値>
```

個別状態変化で、警告状態へ移行する場合の連続抑制回数です。

module_min_ff_event_critical

```
module_min_ff_event_critical <値>
```

個別状態変化で、障害状態へ移行する場合の連続抑制回数です。

module_ff_timeout

```
module_ff_timeout <秒>
```

指定した秒数が経過したら連続抑制回数のカウンターをリセットします。これは、*module_min_ff_event* に指定した抑制回数の状態変化が *module_ff_timeout* に指定の秒数以内に発生する必要があることを意味します。

module_ff_type

```
module_ff_type <値>
```

これは、連続抑制の高度なオプションで、モジュールの状態を制御します。“ カウンタを保持する ” ことによって、値の代わりに、受け取った値を持つモジュールの状態に応じて、ある状態から

別の状態に渡すカウンタ値をいくつか設定します。

有効(1)か無効(0)かを指定します。

module_ff_event

```
module_ff_event X
```

これは、モジュールの連続抑制実行のしきい値(秒単位)です。

module_description

```
module_description <テキスト>
```

モジュールの任意のコメントです。

module_interval

```
module_interval <間隔倍率>
```

それぞれのモジュールの実行間隔をエージェントの間隔の倍率で設定することができます。例えば、エージェントが 300 (5 分)間隔の設定であった場合に、あるモジュールだけ 15分間隔にしたいときに module_interval 3 を設定します。そのモジュールは、300秒 x 3 = 900秒 (15分) 間隔で実行されます。

ブローカーエージェントで module_interval が動作するようにするには、元のエージェントと同じ間隔に設定する必要があります。 そうしないと、正しく動作しない可能性があります。

module_timeout

```
module_timeout <秒数>
```

モジュールの最大実行時間を秒単位で指定します。実行中にこの時間を超過した場合は、実行が中止されます。

module_postprocess

```
module_postprocess <倍率>
```

モジュールから返される値を何倍するかの値です。データの単位を変換するのに便利です。もし、エージェントが取得した値に 1024 を掛けたい場合は、1024 を設定します。また、1024 で割りたい場合は、1/1024 を意味する 0.000976563 を設定します。

module_save

```
module_save <変数名>
```

このパラメータで定義された名前の変数にモジュールから返された値を保存します。この値はあとから他のモジュールで利用できます。

例:

```
module_begin
module_name echo_1
module_type generic_data
module_exec echo 41121
module_save ECHO_1
module_end
```

“ECHO_1” という変数に、値 41121 を保存します。

```
module_begin
module_name echo_2
module_type generic_data
module_exec echo $ECHO_1
module_end
```

2つ目のモジュールでは“$ECHO_1” という変数の内容を表示します。“ 41121 ” となります。

Windows エージェントでは、変数は $var ではなく %var% で指定します。 以下に例を示します。

```
module_begin
module_name echo_2
module_type generic_data
module_exec echo %ECHO_1%
module_end
```

module_crontab

バージョン 3.2 から、モジュールを指定した日時に実行させるようにすることができます。この設定は、module_crontab にて、crontab ファイルに似た設定を行うことにより実現します。

```
module_crontab <分> <時間> <日> <月> <曜日>
```

指定可能な範囲は次の通りです。

- 分 0-59
- 時間 0-23
- 日 1-31
- 月 1-12
- 曜日 0-6 (0 が日曜です)

期間を指定する - を利用することも可能です。

例えば、あるモジュールを毎週月曜の 12時から 15時の間に実行するには、次のような設定をします。

```
module_begin
module_name crontab_test
module_type generic_data
module_exec script.sh
module_crontab * 12-15 * * 1
module_end
```

コマンドを毎時 10分に実行したい場合は、次のようにします。

```
module_begin
module_name crontab_test3
module_type generic_data
module_exec script.sh
module_crontab 10 * * * *
module_end
```

module_condition

```
module_condition <評価式> <コマンド>
```

バージョン 3.2 から、モジュールが特定の値を返す場合にコマンドを実行させることが可能です。次に示すオプションの一つを定義します。

- > [値]: モジュールの値が指定された値よりも大きい場合にコマンドを実行します。
- < [値]: モジュールの値が指定された値よりも小さい場合にコマンドを実行します。
- = [値]: モジュールの値が指定された値と同じ場合にコマンドを実行します。
- != [値]: モジュールの値が指定された値と異なる場合にコマンドを実行します。
- =~ [正規表現]: モジュールの値が指定された正規表現にマッチする場合にコマンドを実行します。
- (値, 値): モジュールの値が指定された値の範囲の場合にコマンドを実行します。

同一のモジュールに複数の条件を設定することも可能です。以下の例では、モジュールの値が 1 と 3 の間の時に*script_1.sh* が実行されます。また、モジュールの値が 5.5 より大きい時に *script_2.sh* が実行されます。*module_exec* の結果が 2.5 なので、最初の *script_1.sh* のみ実行されます。

```
module_begin
```

```
 module_name condition_test
 module_type generic_data
 module_exec echo 2.5
 module_condition (1, 3) script_1.sh
 module_condition> 5.5 script_2.sh
 module_end
```

例:

```
 module_begin
 module_name MyProcess
 module_type generic_data
 module_exec tasklist | grep MyProcess | wc -l
 module_condition> 2 taskkill /IM MyProcess* /F
 module_end
```

```
 module_begin
 module_name PandoraLogSize
 module_type generic_data
 module_exec ls -la "c:\Archivos de programa\pandora_agent\pandora_agent.log" |
gawk "{ print $5 }"
 module_condition> 10000 del "c:\Archivos de
programa\pandora_agent\pandora_agent.log"
 module_end
```

```
 module_begin
 module_name Service_Spooler
 module_type generic_proc
 module_service Spooler
 module_condition = 0 net start Spooler
 module_end
```

- 注意: Windows プラットホームでは、コマンドの実行にはそれが正しく実行されていることを確認するために cmd.exe /c を利用することをお勧めします。例えば次の通りです。

```
 module_begin
 module_name condition_test
 module_type generic_data
 module_exec echo 5
 module_condition (2, 8) cmd.exe /c script.bat
 module_end
```

module_precondition

```
module_precondition <評価式> <コマンド>
```

事前状態定義にマッチした場合モジュールを実行します。次に示すオプションの一つを定義します。

- > [値]: コマンドの実行結果が指定された値よりも大きい場合にモジュールを実行します。

- < [値]: コマンドの実行結果が指定された値よりも小さい場合にモジュールを実行します。
- = [値]: コマンドの実行結果が指定された値と同じ場合にモジュールを実行します。
- != [値]: コマンドの実行結果が指定された値と異なる場合にモジュールを実行します。
- =~[正規表現]:コマンドの実行結果が指定された正規表現にマッチする場合にモジュールを実行します。
- (値, 値): コマンドの実行結果が指定された値の範囲の場合にモジュールを実行します。

以下の例では□precondition で設定した実行結果が 2 と 8 の間の場合に、モジュール *monitoring_variable.bat* が実行されます。この例では *module_precondition* の実行結果が 5 であるため、値が 2 と 8 の間であり、*monitoring_variable.bat* が実行されます。

```
module_begin
module_name Precondition_test1
module_type generic_data
module_precondition (2, 8) echo 5
module_exec monitoring_variable.bat
module_end
```

module_condition と同様に、複数の事前状態定義を利用することができます。モジュールは、すべての事前状態定義にマッチした場合のみ実行されます。

```
module_begin
module_name Precondition_test2
module_type generic_data
module_precondition (2, 8) echo 5
module_precondition <3 echo 5
module_exec monitoring_variable.bat
module_end
```

- 注意: Windows プラットホームでは、コマンドの実行にはそれが正しく実行されていることを確認するために cmd.exe /c を利用することをお勧めします。例えば次の通りです。

```
module_begin
module_name Precondition_test3
module_type generic_data
module_precondition (2, 8) cmd.exe /c script.bat
module_exec monitoring_variable.bat
module_end
```

module_unit

```
module_unit <単位>
```

これは、モジュールの値に付与する単位です。

例:

```
module_unit %
```

module_group

```
module_group <値>
```

これは、モジュールグループ名**です。**

例:

```
module_group Networking
```

module_custom_id

```
module_custom_id <値>
```

モジュールのカスタム ID **です。**

例:

```
module_custom_id host101
```

module_str_warning

```
module_str_warning <値>
```

文字列**タイプのモジュールで**警告状態**を**定義**する**正規表現**です。**

例:

```
module_str_warning .*NOTICE.*
```

module_str_critical

```
module_str_critical <値>
```

文字列**タイプのモジュールで**障害状態**を**定義**する**正規表現**です。**

例:

```
module_str_critical .*CRITICAL.*
```

module_warning_instructions

```
module_warning_instructions <値>
```

モジュールが警告状態に変化したときのオペレータへの指示です。

例:

```
module_warning_instructions Increase incident priority
```

module_critical_instructions

```
module_critical_instructions <値>
```

モジュールが障害状態に変化したときのオペレータへの指示です。

例:

```
module_critical_instructions Call to sys department
```

module_unknown_instructions

```
module_unknown_instructions <値>
```

モジュールが不明状態に変化したときのオペレータへの指示です。

例:

```
module_unknown_instructions Open incident
```

module_tags

```
module_tags <値>
```

カンマ区切りでモジュールに割り当てるタグを指定します。

例:

module_tags tag1,tag2,tag3

module_warning_inverse

```
module_warning_inverse <値>
```

警告閾値範囲の反転を有効化(1)します。

例:

```
module_warning_inverse 1
```

module_critical_inverse

```
module_critical_inverse <値>
```

障害閾値範囲の反転を有効化(1)します。

例:

```
module_critical_inverse 1
```

module_native_encoding

Win32 のみ

```
module_native_encoding <値>
```

この設定トークンは、module_exec によってコマンドラインから実行されるモジュールにのみ影響します。

Windows では、コマンドラインエンコーディング(OEM)、システムエンコーディング(ANSI)および、UTF-16 の 3つのプロセスのエンコーディングがあります。いずれのエンコーディングも通常の文字を扱うことができますが、アクセントを表したものなどいくつかの文字に違いがあります。このトークンにより、Pandora エージェントが設定ファイル(pandora_agent.conf)で指定したエンコーディングに出力を変換します。

module_native_encoding は、次の 4種類の値を設定できます。

- module_native_encoding OEM: コマンドラインエンコーディングに変換
- module_native_encoding ANSI: システムエンコーディングに変換
- module_native_encoding UTFLE: UTF-16 のリトルエンディアンに変換
- module_native_encoding UTFBE: UTF-16 のビッグエンディアンに変換

module_native_encoding の設定が無い場合は、変換は行われません。

module_quiet

```
module_quiet <値>
```

有効化(1)すると、モジュールは静観モードになります。イベントやアラートを生成しません。

例:

```
module_quiet 1
```

module_ff_interval

```
module_ff_interval <値>
```

収集データに揺らぎがあるような場合に、それを変化としてとらえるべきかを判断する期間のデータ収集間隔です。(秒単位)

例:

```
module_ff_interval 2
```

module_macro

```
module_macro<マクロ> <値>
```

コンソールから、ローカルコンポーネントにのみ適用できます。設定ファイルで直接設定しません。

module_alert_template

```
module_alert_template <テンプレート名>
```

このマクロは、パラメータ名(アラートテンプレート 参照)に対応したモジュールにアラートテンプレートを割り当てます。

例:

```
<module>
<name><![CDATA[CPU usage]]></name>
<type>generic_data</type>
<module_interval>1</module_interval>
<min_critical>91</min_critical>
<max_critical>100</max_critical>
<min_warning>70</min_warning>
<max_warning>90</max_warning>
<alert_template><![CDATA[Critical condition]]></alert_template>
<data><![CDATA[92]]></data>
</module>
```

intensive_interval

高頻度モニタリング間隔です。module_intensive_monitoring を設定したモジュールは、障害状態の場合にこの間隔で実行できます。

module_intensive_condition

高頻度モニタリングのための状態です。高頻度監視モジュールがこのパラメータで設定された値に達した場合、intensive_interval で設定した間隔で実行されます。

module_end

モジュール定義の終わりを表します。必須項目です。

情報を取得するためのディレクティブ

各モジュールでは、これらのタイプのうち 1 つだけ使用できます。

module_exec

```
module_exec <コマンド>
```

一般的な<コマンド>実行行です。 1行で情報を取得するコマンドを指定する必要があります。

GNU/Linux では、コマンドはデフォルトのシェルを使って実行されます。デフォルトのシェルは、/bin/sh のシンボリックリンクによって決まります。通常は、bash ですがUbuntu のシステムでは異なります(この場合 dash)。端末でコマンドをテストしても、エージェントからの実行ではエラーが発生する可能性があります。解決策としては、次のようにコマンドラインの実行で明示的に bash を記載します。

```
module_exec bash -c "<command>"
```

実行結果が '0' 以外を返す場合は、実行エラーであり情報は取り込まれません。

Windows エージェントでのデータ取得のためのディレクティブは他にもあります。以下に示します。

module_service

```
module_service <サービス>
```

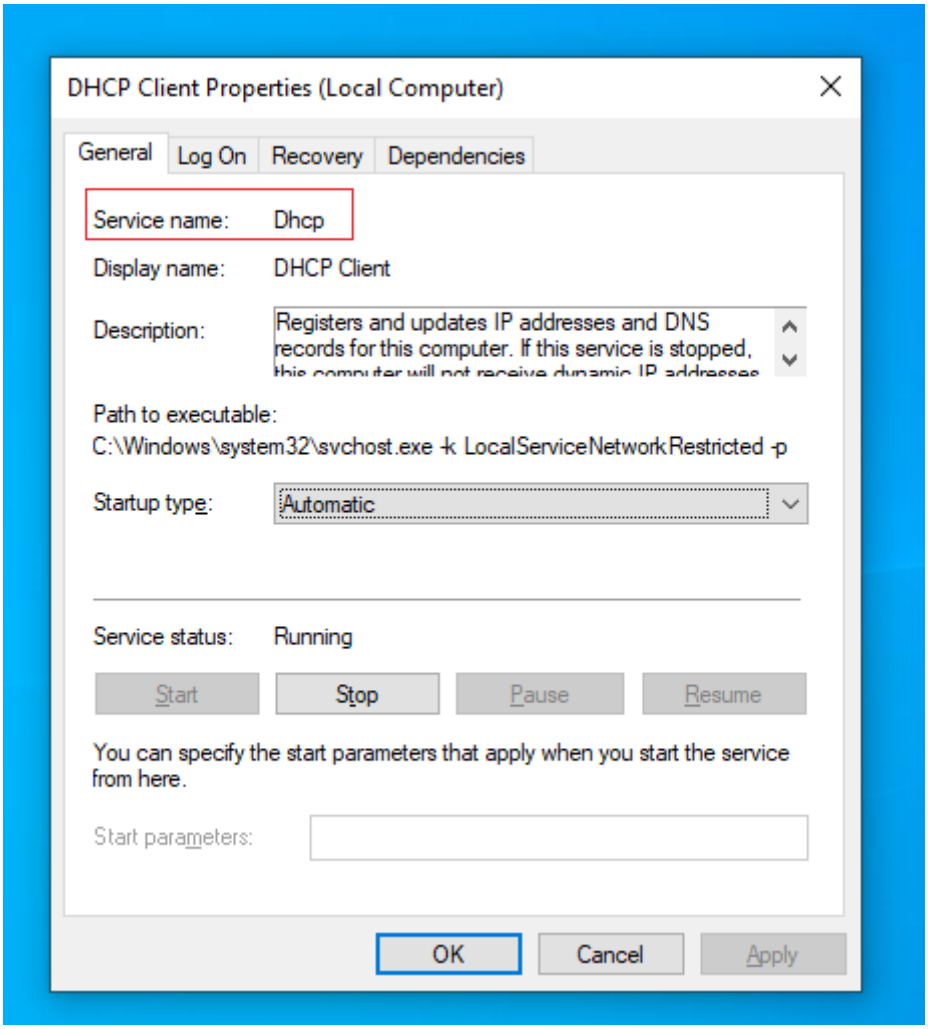
指定したサービスが実行中であるかどうかをチェックします。

MS Windows

サービス名にスペースが含まれる場合は、“ ”でくくるのを忘れないようにしてください。

```
module_begin
module_name Service_Dhcp
module_type generic_proc
module_service Dhcp
module_description Service DHCP Client
module_end
```

サービスは、Windows サービスマネージャに表示される短い名前 (サービス名) で識別されます。



非同期モード

Pandora FMS は通常、(モジュールによって定義される)一定の秒間隔にてチェックを実行します (デフォルトでは、300秒 = 5分)。そのため、チェックの直後にサービスがダウンすると、それがダウンであると認識するのに、さらに 300秒かかります。非同期モジュールでは Pandora に “ 今すぐ ” サービスの障害を通知するようにできます。これは、*非同期*操作モードと呼んでいます。非同期モードにするには、以下のディレクティブを設定してください。

```
module_async yes
```

この機能は、ブローカーエージェントでは利用できません。

Windows Home Edition では、この非同期機能はサポートされていません Pandora エージェントはサービスが動作しているかどうかを定期的に確認します。大量のサービスを監視する場合システムリソースを消費するため、同期モードを利用することをお勧めします。

サービスのウォッチドッグ

サービスがダウンしたときに再起動するためのウォッチドッグモードがあります。サービスを起動するためのパラメータは Windows が認識しているため必要ありません。設定は簡単で、以下に例を示します。

```
module_begin
module_name ServiceSched
module_type generic_proc
module_service Schedule
module_description Service Task scheduler
module_async yes
module_watchdog yes
module_end
```

Unix

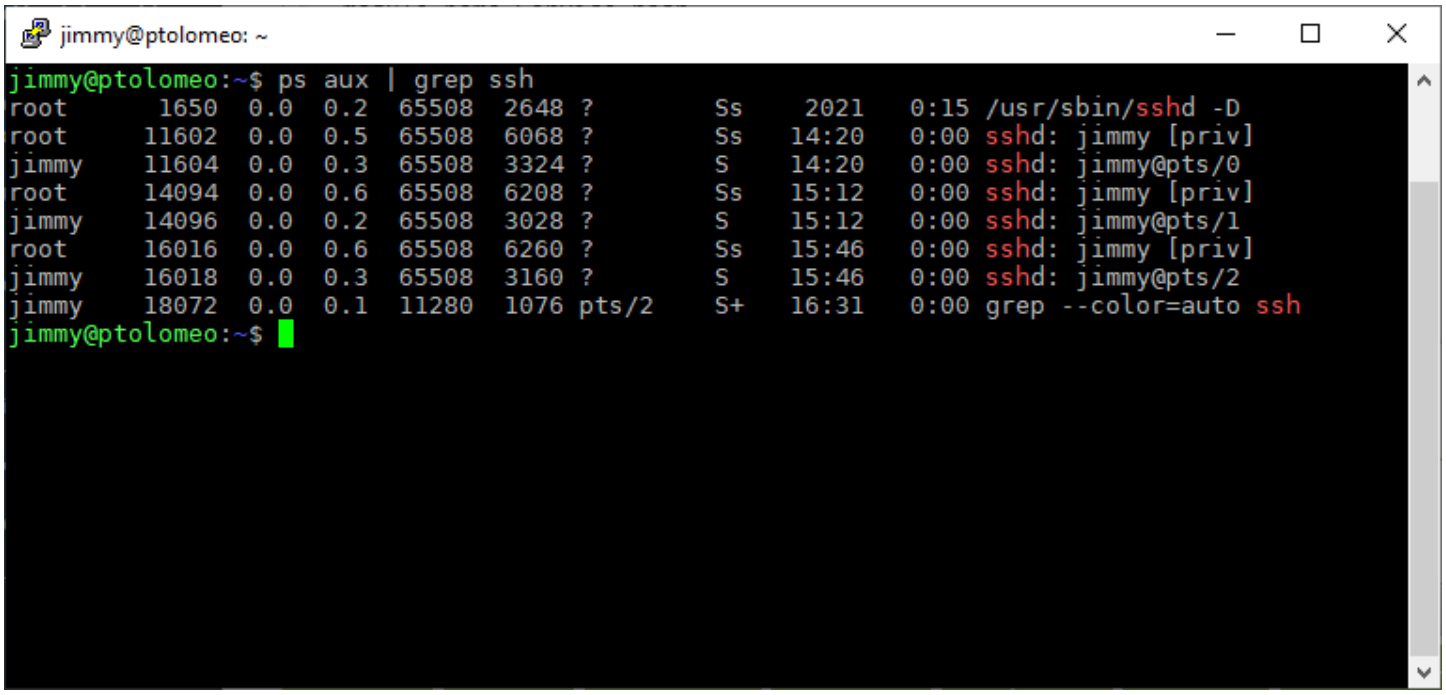
Unix でも Windows と同様に動作します。違いは Unix であることだけで、プロセスおよびサービスは同じ考え方です。例えば bash プロセスが動作しているかを確認するには次のようにします。

```
module_begin
module_name Service_bash
module_type generic_proc
module_service /bin/bash
module_description Process bash running
module_end
```

Unix エージェントでは、ウォッチドッグと非同期検出はできません。

module_service では、ps aux で表示されるコマンドのフルパスを指定する必要があります。以下の例では SSH サービスをチェックします。

```
ps aux | grep ssh
```



Configure it:

```
module_begin
module_name MY_SSHD
module_type generic_proc
module_service /usr/sbin/sshd -D
module_description Is sshd running?
module_end
```

module_proc

```
module_proc <プロセス>
```

指定した名前のプロセスがいるかどうかをチェックします。

Windows

プロセス名にスペースが含まれていても、“ ” は使わないでください。プロセス名に拡張子 .exe がつくかどうかを良

く確認してください。モジュールは、指定した名前で実行
されているプロセスの数を返します。

以下に cmd.exe プロセスをモニタリングする例を示します。

```
module_begin
module_name CMDProcess
module_type generic_proc
module_proc cmd.exe
module_description Process Command line
module_end
```

非同期モード

サービスと同じように、モニタリングしているプロセスにて何らかの原因で障害が発生することがあります。Windows エージェントでは、*module_proc* モジュールで非同期チェックをサポートしています。この場合、プロセスの状態が変化したとき、通常のモニタリング間隔における次のチェックタイミングを待たずにすぐに 通知されます。これにより、プロセスに障害が発生したのとほぼ同時にそれを知ることができます。以下にプロセスの非同期モニタリング設定の例を示します。

```
module_begin
module_name Notepad
module_type generic_proc
module_proc notepad.exe
module_description Notepad
module_async yes
module_end
```

違いは“module_async yes” が設定されている点です。

プロセスのウォッチドッグThis feature is not supported on broker agents.

ウォッチドッグは、ダウンしたプロセスを見つけた場合にそれをすぐに起動させることができます。Pandora FMS の Windows エージェントは、プロセスがダウンしたときにウォッチドッグとして動作させることができます。

プロセスの実行には、いくつかのパラメータが必要です。このモジュールには、いくつかの追加設定オプションがあります。*ウォッチドッグ*モードは、モジュールタイプが*非同期* の場合のみ動作することに注意してください。ウォッチドッグでの module_proc の設定例を見てみます。

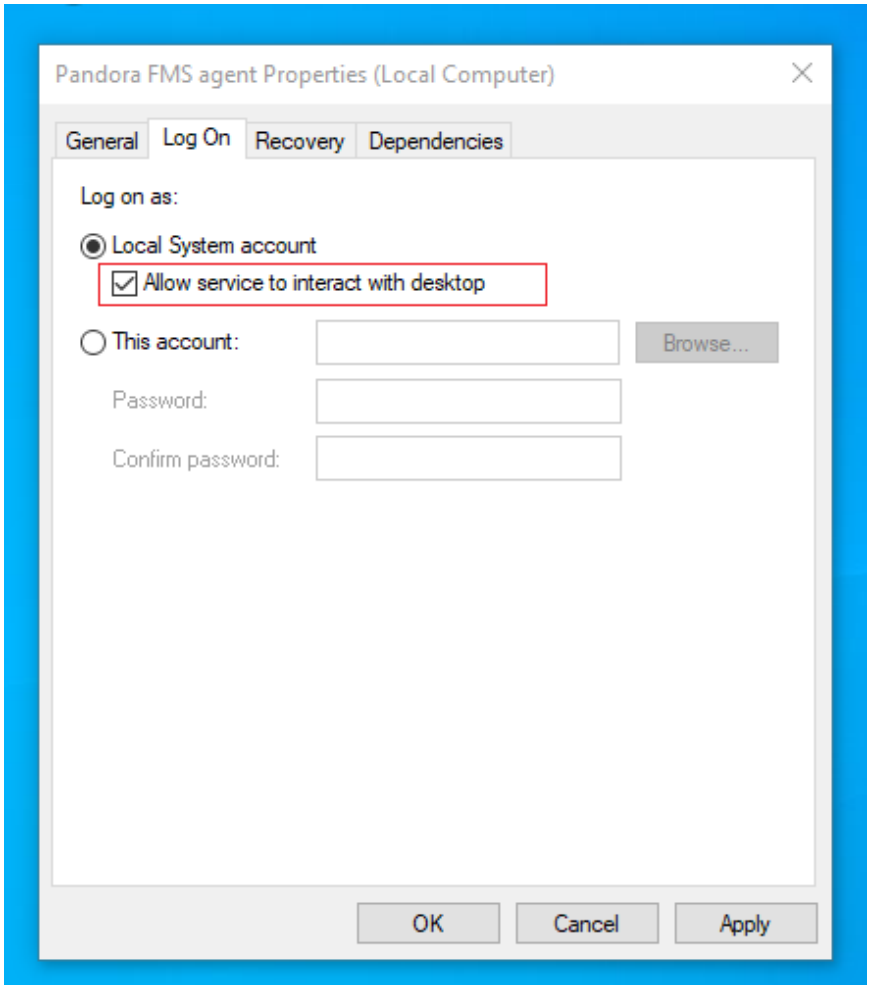
```
module_begin
module_name Notepad
module_type generic_proc
module_proc notepad.exe
module_description Notepad
module_async yes
module_watchdog yes
```

```
module_start_command c:\windows\notepad.exe
module_startdelay 3000
module_retrydelay 2000
module_retries 5
module_end
```

以下に、ウォッチドッグを行う場合の module_proc の追加パラメータを示します。

- module_retries:ウォッチドッグによりプロセスを起動させるリトライ回数を指定します。指定した回数に達した場合、該当モジュールのウォッチドッグは無効になり、ユーザによる復旧(少なくともエージェントの再起動) がされるまでプロセスの起動を行わなくなります。デフォルトでは上限はありません。
- module_startdelay:プロセスのダウンを認識した後、最初にプロセスを起動させるまでの待ち時間をミリ秒単位で指定します。
- module_retrydelay:プロセスダウンを検知した後に起動に失敗した場合の再実行の待ち時間をミリ秒単位で指定します。
- module_user_session: プロセスの起動をどのセッションで行うかを制御します'no' に設定すると、プロセスはサービスセッションで起動し、バックグラウンドで動作します(デフォルト)'yes' に設定すると、プロセスはユーザセッションで起動し、PC のデスクトップに表示されます。

Windows Vista 以前のバージョンではPandora FMS サービスプロパティで、“Interactive access with desktop” を有効にすることにより *module_user_session* を設定することができます。以下のスクリーンショットに示します。



また Pandora FMS は “SYSTEM” アカウントのサービスとして動作するため、実行されるコマンドはこのユーザおよびこのユーザの環境で動作することに注意してください。したがって、特定のプロセスを特定のユーザで実行したい場合は、環境変数の設定やその他事前処理を行う呼び出し用スクリプト (.bat など) を用意し、そのスクリプトをウォッチドッグの起動プログラムとして設定する必要があります。

module_cpuproc <プロセス> (Unix のみ)

特定のプロセスの CPU 使用率を返します。

```
module_begin
module_name myserver_cpu
module_type generic_data
module_cpuproc myserver
module_description Process Command line
module_end
```

Unix

UNIX では、このモジュールは module_service のように動作します。非同期およびウォッチドッグモードはサポートしません。

module_memproc

```
module_memproc <プロセス>
```

Unixのみです。 特定のプロセスが利用しているメモリ量を返します。

```
module_begin
module_name myserver_mem
module_type generic_data
module_memproc myserver
module_description Process Command line
module_end
```

module_freedisk

```
module_freedisk <ドライブ名:>|<ボリューム>
```

このモジュールは、ディスクの空き容量をチェックします。

Windows® の場合

ドライブ名の後に：を記載します。

```
module_begin
module_name freedisk
module_type generic_data
module_freedisk C:
module_end
```

Unix® の場合

/var のようにチェックするボリュームを指定します。

```
module_begin
module_name disk_var
module_type generic_data
module_freedisk /var
module_end
```

module_freepercentdisk

```
module_freepercentdisk <ドライブ名:>|<ボリューム>
```

このモジュールは、ディスクの空きをパーセントで返します。

Windows

ドライブ名: (" : " を忘れないようにしてください) を指定します。

```
module_begin
module_name freepercentdisk
module_type generic_data
module_freepercentdisk C:
module_end
```

Unix

/var などのボリュームを指定します。

```
module_begin
module_name disk_var
module_type generic_data
module_freepercentdisk /var
module_end
```

module_occupiedpercentdisk

```
module_occupiedpercentdisk <ドライブ名:>|<ボリューム>
```

Unix のみです。 このモジュールは、/var 等の Unix ファイルシステムのディスク使用率(%)を返します。

```
module_begin
module_name disk_var
module_type generic_data
module_occupiedpercentdisk /var
module_end
```

module_cpuusage

```
module_cpuusage <cpu id|all>
```

このモジュールは、UNIX および Windows 双方で使えます。指定した CPU 番号の CPU 使用率を返します CPU が 1つしかない場合は番号を指定しないか all を指定します。

次のように、マルチ CPU 環境で全 CPU の平均使用率を得ることができます。

```
module_begin
module_name SystemCPU
module_type generic_data
module_cpuusage all
module_description Average CPU use in systme
module_end
```

1つ目の CPU 使用率を確認するのは次のようにします。

```
module_begin
module_name SystemCPU_1
module_type generic_data
module_cpuusage 1
module_description Average CPU use in system for CPU #1
module_end
```

module_freememory

Windows および Unix 双方で使えます。システム全体の空きメモリ量を取得します。

```
module_begin
module_name FreeMemory
```

```
module_type generic_data
module_freememory
module_description Non-used memory on system
module_end
```

module_freepercentmemory

Windows および Unix 双方で使えます。システム全体の空きメモリ量のパーセンテージを取得します。

```
module_begin
module_name freepercentmemory
module_type generic_data
module_freepercentmemory
module_end
```

module_tcpcheck

Windows のみです。 このモジュールは、指定された IPアドレスおよびポート番号への接続確認を行います。成功すると 1 が返り、失敗すると 0 が返ります。なお、タイムアウトを設定する必要があります。

```
module_begin
module_name tcpcheck
module_type generic_proc
module_tcpcheck www.artica.es
module_port 80
module_timeout 5
module_end
```

module_regexp

Windows のみです。このモジュールは、 正規表現を使ってファイル(ログ)の内容の比較を行います。モニタリングを開始した時点ですでに存在している行については無視します。モジュールが返す値はモジュールタイプにより異なります。

- generic_data_string, async_string: 正規表現にマッチした行全体を返します。
- generic_data: 正規表現にマッチした行数を返します。
- generic_proc: 正規表現にマッチしたら 1、そうでなければ 0 を返します。
- module_noseekeof: 0 がデフォルトです。これを有効にすると、それぞれのモジュールの実行において、ターゲットのファイル更新からは独立して、ファイルの EOF を確認することなくチェック処理が実行されます。これにより、常に検索パターンにマッチするすべての行が XML 出力に展開されます。

```
module_begin
module_name regexp
module_type generic_data_string
```

```
module_regexp C:\WINDOWS\my.log
module_pattern ^\[error\].*
module_noseekeof 1
module_end
```

module_wmiquery

Windows のみです。WMI モジュールは、外部ツールを使わずにローカルで WMI クエリを実行できます。2つのパラメータで設定します。

- module_wmiquery: 利用する WQL クエリを設定します。さまざまなデータを含む複数の行で結果を得ることができます。
- module_wmicolumn:データソースとして利用するカラム名を設定します。

例えば、インストールされているサービス一覧を取得するには次のようにします。

```
module_begin
module_name Services
module_type generic_data_string
module_wmiquery Select Name from Win32_Service
module_wmicolumn Name
module_end
```

現在の CPU 負荷であれば次のようにします。

```
module_begin
module_name CPU_speed
module_type generic_data
module_wmiquery SELECT LoadPercentage FROM Win32_Processor
module_wmicolumn LoadPercentage
module_end
```

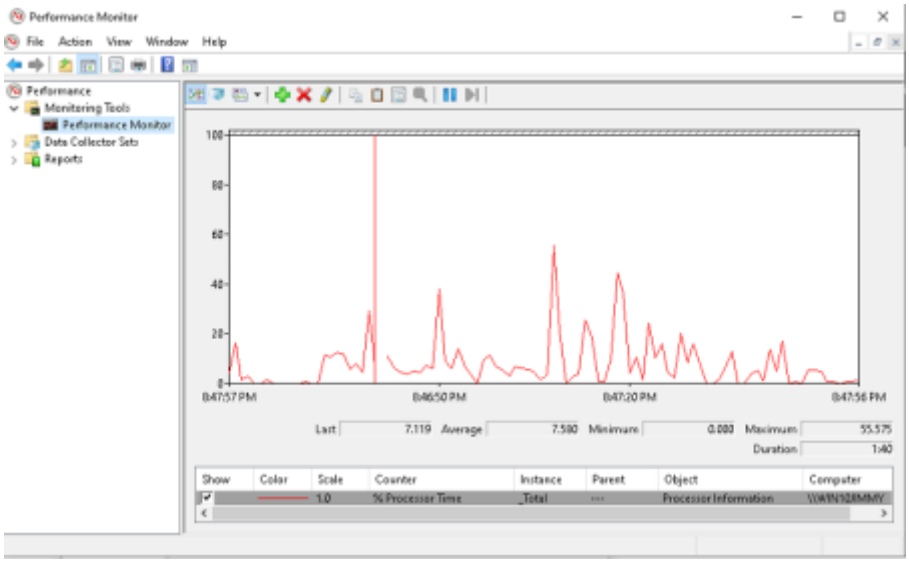
module_perfcounter

Windowsのみです。 PDH インタフェースを通して、パフォーマンスカウンタ (http://msdn.microsoft.com/ja-jp/library/aa373083(v = vs.85).aspx パフォーマンスカウンタドキュメント) からデータを取得します。Windows のライブラリの *pdh.dll* がインストールされている必要があります。もしインストールされていない場合は、Windows パフォーマンス解析ツールをインストールする必要があります (通常はデフォルトでインストールされています)。

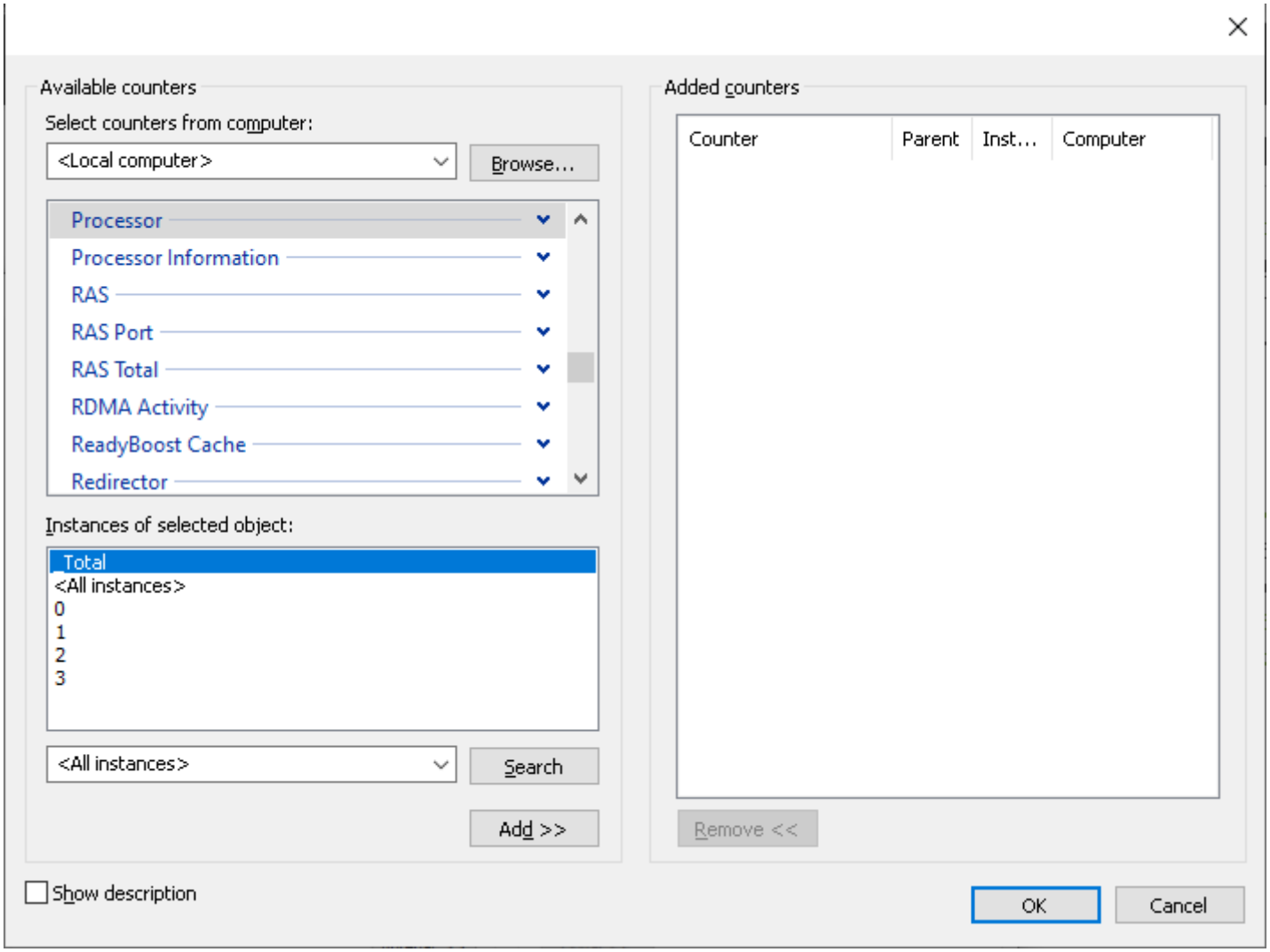
```
module_begin
module_name perfcounter
module_type generic_data
module_perfcounter \Memory\Pages/sec
module_end
```

Windowsパフォーマンスモニタは、モニタリングに利用できる何百ものパラメータを持つ強力なツールです。加えて、各ベンダによる独自のモニタ項目も追加されています。

パフォーマンスツールから、パフォーマンスカウンターを確認できます。



システムツールを用いて新たなパフォーマンスカウンターを追加することができます。その構成には、要素とサブ要素を含む管理構造があります。ここでは、*Processor*, *% of processor time* および *_Total* です。



このようにOS のツールを使うことにより、システムパフォーマンスのさまざまな要素を取り込むことができます。この場合、モジュールの設定は次のようになります。

```
module_begin
module_name Processor_Time
module_type generic_data_inc
module_perfcounter \Processor(_Total)\% of time processor
module_end
```

デフォルトでは、カウンタの生データが表示されます。調整値を取得するには、module_cooked 1 を設定します。

```
module_begin
module_name Disk_E/S_Sec
module_type generic_data
module_cooked 1
module_perfcounter \PhysicalDisk(_Total)\E/S by second
module_end
```

多くのデータは、カウンタ値として返ってきます。そのため、データタイプとしては generic_data_inc を使う必要があります。また、とても大きなスケール (数百万) のデータが返って

くることがありますので、事前処理モジュールで 0.000001 などを設定して値を小さくすると良いです。

module_inventory (廃止)

Windows および Linux/UNIX 双方で、現在この機能はエージェントプラグインによるインベントリで置き換えられています。

(Win32 のみ、Linux/Unix ではエージェントプラグインとして実装されています。)

前述の WMI を利用することにより、このモジュールはソフトウエアおよびハードウエアの違った情報を取得します。

モジュールは、取得した情報の種類を分類し収集します。以下に情報の種類の一覧を示します。

- cpu: システムの CPU 情報を取得します。(プロセッサ名、クロック数、説明)
- CDROM: CD-ROM の情報を取得します。(名称およびドライブ名)
- Video: ビデオカードの情報を取得します。(説明、RAM容量、プロセッサ)
- HDs: ハードディスクの情報を取得します。(モデル、サイズおよび、システムにおける名前)
- NICs: ネットワークコントローラの情報を取得します。(説明、MAC アドレスおよび、IPアドレス)
- Patches: インストール済のパッチ情報を取得します。(ID、説明、コメント)
- Software: MSI パッケージの情報を取得します。(名前およびバージョン)
- RAM: RAM モジュールの情報を取得します。(タグ、容量および名前)
- Service: インストールされたサービスの情報を取得します。最初のカラムに、Pandora FMS がサービスのモニタリングに使う、サービスの短い名前が表示されます。

追加モジュールパラメータ:

- module_interval: このモジュールは、*日単位*で情報収集するための間隔を定義する追加設定です。

このモジュールの利用例を以下に示します。

```
module_begin
module_name Inventory
module_interval 7
module_type generic_data_string
module_inventory RAM Patches Software Services
module_description Inventory
module_end
```

module_logevent

Windows のみです。指定されたパターンに基づいて Windows イベントログから情報を取得し、ソースおよびイベントタイプに従ってフィルタリングする機能を提供します。

このモジュールの一般的な書式は次の通りです。

```
module_begin
module_name MyEvent
module_type async_string
module_logevent
module_source <logName>
module_eventtype <event_type/level>
module_eventcode <event_id>
module_application <source>
module_pattern <text substring to match>
module_description
module_end
```

すでに表示されたものを再度表示するのを防ぐために、エージェントが実行された最終時間より後のイベントのみを対象とします。

module_logevent には、次のパラメータを設定できます。(すべて大文字小文字を区別します)

- module_source: イベントソース (System, Application, Security) を指定します。このフィールドは必須項目です。
- module_eventtype: イベントタイプ (Error, Information 等) を指定します。このフィールドはオプションです。
- module_pattern: 検索するパターン (文字列) を指定します。このフィールドはオプションです。
- module_eventcode: 5112 等のイベント ID 番号です。このフィールドはオプションです。
- module_application: イベント発生元のアプリケーションを指定します。イベントが検索される名前やログファイルを示す module_source と混同しないように注意してください。

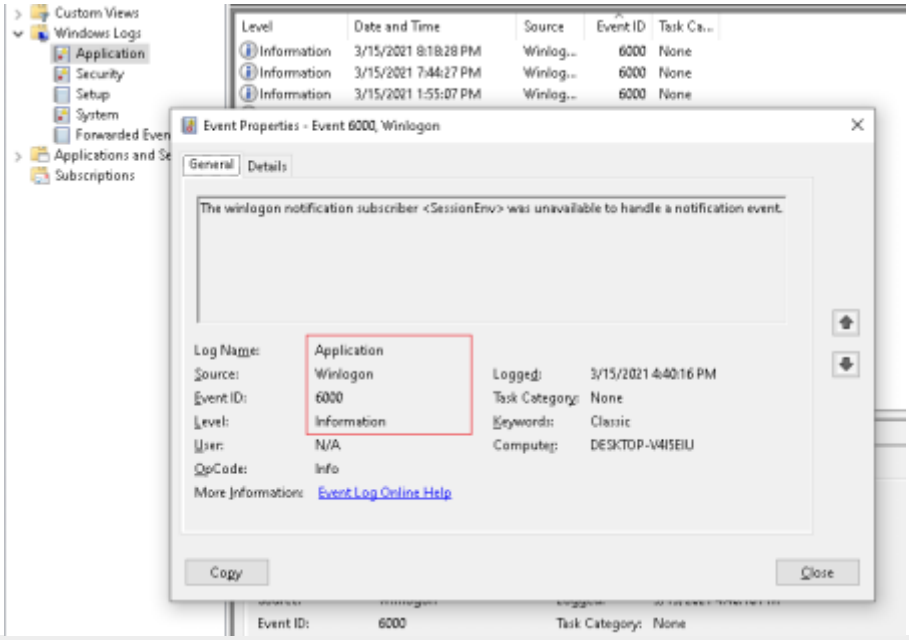
例えばsystem の error に分類される全てのイベントを表示するには、次のように設定します。

```
module_begin
module_name log_events
module_type generic_data_string
module_description System errors
module_logevent
module_source System
module_eventtype error
module_end
```

PandoraAgent という文字を含む全イベントを表示するには次のようにします。

```
module_begin
module_name log_events_pandora
module_type async_string
module_description PandoraAgent related events
module_logevent
module_source System
module_pattern PandoraAgent
module_end
```

その他例として、以下にイベントフィルタリングのスナップショットを示します。




```
module_begin
module_name MyEvent
module_type async_string
module_source Application
module_eventtype Information
module_eventcode 6000
module_application Winlogon
module_pattern unavailable to handle
module_description
module_end
```

Pandora FMS は、ログを収集するシステムではない ということを理解してください。このツールは、これらのクリティカルまたは重要なイベントを選択し、監視するためのシステムです。すべてのイベントを未分類で収集すると、データベースが飽和状態になりシステムパフォーマンスが著しく低下し、長期的に問題をもたらします。Pandora FMS を一般的なイベント収集ツールとして使うべきではありません。

module_logchannel

バージョン NG 715 以上および Windows のみ

Windows ログチャネルに関する情報を取得するモジュールです。*module_logevent* は Windows ログにのみアクセスしますが、これは、チャネルとして設定された他のログファイルからデータを展開することができます。これにより、サービスやアプリケーションログを含めたログ取得が可能になります。

このモジュールの一般的な書式は次の通りです。

```
module_begin
module_name MyEvent
```

```
module_type async_string
module_logchannel
module_source <ログチャネル>
module_eventtype <イベントタイプ/レベル>
module_eventcode <イベントID>
module_application <ソース>
module_pattern <マッチするテキスト文字列>
module_description <説明>
module_end
```

すでに表示されたものを再度表示するのを防ぐために、エージェントが実行された最終時間より後のイベントのみを対象とします。

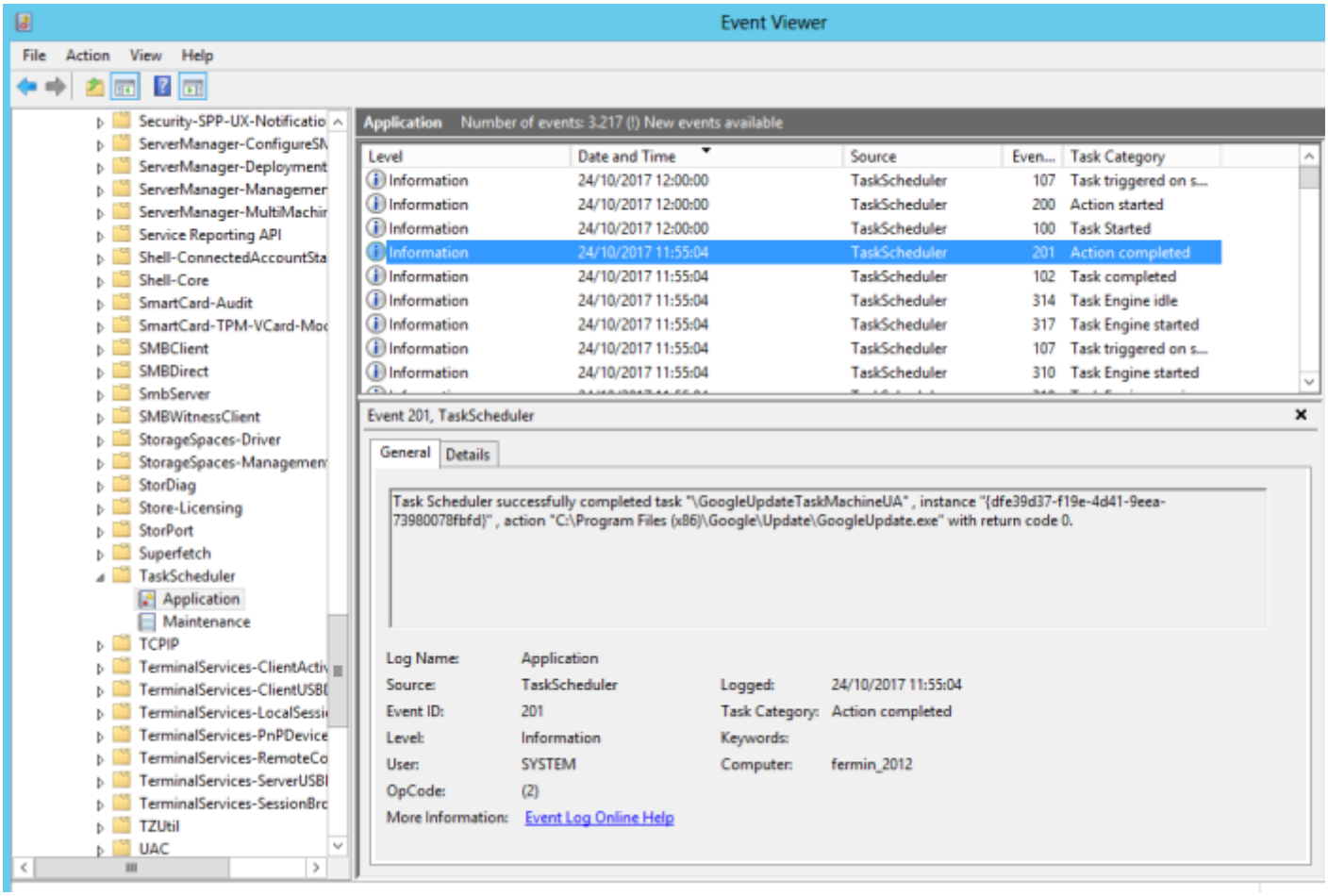
module_logchannel には、次のパラメータを設定できます。(すべて大文字小文字を区別します)

- module_source: イベントチャネルを指定します。wevtutil.exe enum-logs を実行すると、ローカルログチャネルの一覧を取得できます。このフィールドは必須項目です。
- module_eventtype: イベントタイプ (critical, error, warning, info, verbose) を指定します。このフィールドはオプションです。
- module_pattern: 検索するパターン (文字列) を指定します。このフィールドはオプションです。
- module_eventcode: 5112 等のイベント ID 番号です。このフィールドはオプションです。
- module_application: イベント発生元のアプリケーションを指定します。イベントが検索される名前やログファイルを示す module_source と混同しないように注意してください。

例えば、チャネル *Microsoft-Windows-TaskScheduler/Operational* の *information* タイプ、コード *201* 、文字列 *code 0* を含むすべてのイベントを表示するモジュールは次のようになります。

```
module_begin
module_name New logs
module_type async_string
module_logchannel
module_description Successfully completed tasks
module_source Microsoft-Windows-TaskScheduler/Operational
module_eventtype information
module_eventcode 201
module_pattern code 0
module_end
```

このモジュール設定により、Pandora FMS エージェントは次のようなログを収集します。



イベントチャネルの名前を取得するには、チャネルを右クリックして、”プロパティ(properties)”を選択し、パラメータ“フルネーム(Full name)”をコピーします。これは、module_source で必要です。

module_plugin

プラグインから取得したデータを定義するためのパラメータです。これは、特殊なモジュールで、'module_begin' や 'module_type' などの 他の識別子は必要ありません

次のような書式です。

```
module_plugin plugin_filename parameter_1 parameter_2 (...) parameter_X
```

ただし、プラグインに状態や実行間隔などの追加パラメータを設定する場合は、次のように通常の書式を利用します。

```
module_begin
module_plugin plugin_filename parameter_1 parameter_2 (...) parameter_X
module_interval 2
module_condition (0, 1) script.sh
module_end
```

それぞれのプラグインには個別のパラメータが使われます。そのため、それぞれの特定のドキュメントを参照する必要があります。ここでは、エージェントにデフォルトで付属しているプラグインの一つであるファイル内を検索する grep_log で説明します。

```
module_plugin grep_log /var/log/syslog Syslog ssh
```

この例では、プラグインの名前が grep_log で、/var/log/syslog を正規表現 ssh で検索する Syslog というモジュールであることを意味します。

Windows システムの場合の別の例を示します。(バージョン 3.1 以上)

```
module_plugin cscript.exe //B "%ProgramFiles%\Pandora_Agent\util\df_percent.vbs"
```

module_ping

Windows のみ。

```
module_ping <ホスト>
```

このモジュールは、指定したホストに ping を行い、応答があれば 1、そうでなければ 0 を返します。

次のパラメータを設定できます。

- module_ping_count x: 送信する ECHO_REQUEST パケットの数 (1 がデフォルト)
- module_ping_timeout x: 応答を待つタイムアウトミリ秒数 (1000 がデフォルト)
- module_advanced_options: *ping.exe* の拡張オプション

例:

```
module_begin
module_name Ping
module_type generic_proc
module_ping 192.168.1.1
module_ping_count 2
module_ping_timeout 500
module_end
```

module_snmpget

Windows のみ。

このモジュールは snmpget を実行し、その応答を返します。

次のパラメータを設定できます。

- module_snmpversion [1_2c_3]: SNMP バージョン (1 がデフォルト)
- module_snmp_community <community>: SNMP コミュニティ (*public* がデフォルト)
- module_snmp_agent <host>: 対象の SNMP エージェント
- module_snmp_oid <oid>: 対象の OID
- module_advanced_options: *snmpget.exe* の拡張オプション

例:

```
module_begin
module_name SNMP get
module_type generic_data
module_snmpget
module_snmpversion 1
module_snmp_community public
module_snmp_agent 192.168.1.1
module_snmp_oid .1.3.6.1.2.1.2.2.1.1.148
module_end
```

module_wait_timeout

Windows のみ。

```
module_wait_timeout X
```

モジュールの module_exec および module_plugin 出力がチェックされる際のタイムアウト時間。デフォルトの値は 500 ミリ秒です。大量の出力を生成するモジュールで実行が遅い場合は、5 に変更します。それ以外の場合は使用しないことをお勧めします。

module_advanced_options

MS Windows® のみ

```
module_advanced_options <parameter>
```

module_ping および module_snmpget では追加パラメータを使用します。たとえば、ping の場合、512 バイトのパケット サイズを指定できます。

```
module_advanced_options -l 512
```

自動エージェント設定

概要

エージェント設定処理では、エージェントを自動的に設定するための一連のルールを設定できます。

それはこのように動作します：

1. Pandora FMS コンソールまたはPandora FMS メタコンソールで、自動設定の準備をします。
2. Pandora FMS へ接続するエージェントをインストールします。(単一のコンソールの場合はエージェントの接続先は Pandora FMS サーバです。自動プロビジョニングを設定したメタコンソールがある場合は、メタコンソールをサーバとして設定します。)
3. Pandora FMS サーバが、エージェントのデータを含む XML (.data) を初回受信します。
4. 適用される自動設定を決定するためのルールが評価されます。
5. エージェントが新たな設定を受け取り、次回の処理から更新された設定で動作します。

自動エージェント設定の作成/編集

コンソール

自動設定画面へは 設定(Configuration) → 自動エージェント設定管理(Manage agent autoconfiguration) からアクセスします。

メタコンソール

中央管理(Centralized management) → エージェント管理(Agent management) → エージェント自動設定アイコン:

管理ページにアクセスしたら、新たな設定定義の追加(Add new configuration definition) ボタンをクリックすることにより新たな自動設定を作成することができます。自動設定の名前と説明を設定します。

新たな自動設定を作成したら、必要な項目 (ルール(Rules)エージェント自動設定(Agent autoconfiguration)追加アクション(Extra actions)) をクリックすることにより設定フォームを表示できます。

ルール

自動設定が適用されるエージェントを定義するには、まずエージェントを識別するルールを追加します。

自動設定内のルールセクションを展開し、新しいルールの追加(Add new rule) を選択します。 ルールセレクターで一連のオプションを選択して、設定するエージェントを識別できます。

- Server name: マッチするサーバ名
- Group name: マッチするグループ名
- OS: マッチする OS 名 (正規表現)
- Custom field: エージェントが報告してくる、キー/値 のカスタムフィールドマッチ。カスタムフィールド名と値を指定する必要があります。
- IP range: IP の範囲(ネットワーク)にマッチIP/ネットマスク の書式を追加います。
- Script output (> 0): スクリプトの実行結果で、標準出力の結果が 0より大きい場合にルールにマッチしたと評価されます。
- Rules script call: 'arguments' フィールドに次のマクロが使えます。(演算子 AND と OR のどちらかを選

択して、ルールロジックを変更できます)
- _agent_ : エージェント名に置き換えられます。
- _agentalias_ : エージェントの別名に置き換えられます。
- _address_ : エージェントから報告された IP アドレスに置き換えられます。
- _agentgroup_ : エージェントから報告されたグループ名に置き換えられます。
- _agentos_ : エージェントの OS で置き換えられます。

ルールが無い場合は、自動設定は適用されません。全てのエージェントに対して一つの設定でよい場合は、すべての別名にマッチする正規表現 .* を使うことができます。

設定

- エージェントグループ(Agent Group): 変更せずにそのままとするか、強制的に変更します。
- セカンダリグループ(Secondary Group): エージェントにセカンダリグループとして追加するグループを選択します。
- ポリシ(Policies): エージェントがサーバに接続してきたときに適用するポリシーを選択できます。
- 設定ブロック(Configuration Block): エージェントの設定ファイルに追加する設定内容です。

集中管理が有効なメタコンソールに属するノードから自動設定管理を行おうとすると、リードオンリー表示になります。

追加アクション

ここでは、自動設定に追加のアクションを行うことができます。例えば以下のような処理です。

1. 独自イベントを出す (Launch custom event)
2. アラートアクションを実行する (Launch alert action)
3. スクリプトを実行する (Launch script)

システムは、次のマクロをサポートします。

- _agent_ エージェント名に置き換えられます。
- _agentalias_ エージェントの別名に置き換えられます。
- _address_ エージェントが報告した IP アドレスに置き換えられます。
- _agentgroup_ エージェントが報告したグループ名に置き換えられます。
- _togentos_ エージェントの OS に置き換えられます。
- _agentid_ エージェント ID に置き換えられます。

Unix/Linux エージェント

Pandora FMS Unix エージェントの設定

把握しておくべき基本的なファイルとディレクトリは次のとおりです。

- /usr/share/pandora_agent : Pandora FMS エージェントがインストールされる場所です。厳密なシステムポリシーがあり、ここにインストールできないシステムでは、実際のインストールパスからこのパスへのリンクを作成することを推奨します。例: /opt/pandora → /usr/share/pandora_agent
- /etc/pandora/pandora_agent.conf : エージェントのメインの設定ファイルです。ここにデータの収集に使うコマンドの定義がされています。
- /usr/local/bin/pandora_agent : エージェントの実行バイナリです。通常、/usr/bin/pandora_agent にリンクされています。
- /usr/local/bin/tentacle_client : サーバにデータファイルを送信するため Tentacle 実行バイナリです。通常、/usr/bin/tentacle_client にリンクされています。
- /etc/init.d/pandora_agent_daemon : start/stop/restart のためのスクリプトです。
    - AIX システムでは、/etc/rc.pandora_agent_daemon です。
- /var/log/pandora/pandora_agent.log : Pandora FMS エージェントがデバッグモードで動作している時にログが出力されるログファイルです。
- /etc/pandora/plugins : エージェントプラグインを置くディレクトリです。これは /usr/share/pandora_agent/plugins にリンクされています。
- /etc/pandora/collections : エージェントのコレクションを置くディレクトリです。これは /usr/share/pandora_agent/collections にリンクされています。

Unix エージェントの実行

エージェントの起動は以下のようにします。

```
/etc/init.d/pandora_agent_daemon start
```

エージェントの停止は以下のようにします。

```
/etc/init.d/pandora_agent_daemon stop
```

起動スクリプトで Pandora FMS エージェントの起動・停止ができます。デフォルトでは起動後はデーモーンとして動作します。

Unix エージェントでシステム情報を取得する別の方法

設定の章で示した通り、module_exec で特定のコマンドを実行することなく定義済の情報を取得できるモジュールがあります。以下がそれらです。

- module_procmem
- module_freedisk
- module_freepercentdisk
- module_cpuproc
- module_proc
- module_procmem
- module_cpuusage
- module_freememory
- module_freepercentmemory

これらのモジュールの動作は、エージェントの実行ファイル (デフォルトでは /usr/bin/pandora_agent) を直接編集することにより変更できます。Pandora FMS エージェントは一般的に /usr/bin/pandora_agent にあります。

コマンド名で検索することにより、内部コマンドを含むコードを見つけます。システムに合わせて変更する場合は、修正を行います。

```
# Commands to retrieve total memory information in kB
use constant TOTALMEMORY_CMDS => {
   linux => 'cat /proc/meminfo | grep MemTotal: | awk \'{ print $2 }\,
   solaris => 'MEM=`prtconf | grep Memory | awk \'{print $3}\'` bash -c \'echo
$(( 1024 * $MEM ))\,
   hpux => 'swapinfo -t | grep memory | awk \'{print $2}\
};

# Commands to retrieve partition information in kB
use constant PART_CMDS => {
    # total, available, mount point
    linux => 'df -P | awk \'NR> 1 {print $2, $4, $6}\,
    solaris => 'df -k | awk \'NR> 1 {print $2, $4, $6}\,
    hpux => 'df -P | awk \'NR> 1 {print $2, $4, $6}\,
    aix => 'df -kP | awk \'NR> 1 {print $2, $4, $6}\
};
```

情報取得のための定義済の値を変更するには、コマンドを編集するだけですが、以下に注意します。

1. ブロックの終わりに { }; のようにセミコロンがあるがことを確認
2. コマンドが ' ' でくくられているか確認
3. シングルクォーテーションを使う場合は、` ` のような追加の引用符が必要になる場合があります(前の例を参照)。
4. コマンドで使用するシングルクォーテーションの前に \ が付いていることを確認してください。これは \' を意味します。 たとえば、このコマンドは通常次のようになります。

```
df -P | awk 'NR> 1 {print $2, $4, $6}'
```

上記は以下のようになります。

```
df -P | awk \'NR> 1 {print $2, $4, $6}\'
```

Pandora FMS Windows エージェント

Pandora FMS Windows エージェントの設定

Windows エージェントの基本的なパスやディレクトリは、エージェントをインストールした場所になります。デフォルトでは %ProgramFiles% です。

理解しておきたい基本的なファイルは次の通りです。

%ProgramFiles%\pandora_agent

Pandora FMS エージェントの実行ファイルやディレクトリがあるインストール先です。

%ProgramFiles%\pandora_agent\pandora_agent.conf

エージェントの設定ファイルです。実行モジュールおよびエージェントプラグインの設定はここにあります。

%ProgramFiles%\pandora_agent\PandoraAgent.exe

エージェントの実行バイナリです。

%ProgramFiles%\pandora_agent\util\tentacle_client.exe

サーバへファイルを転送するための Tentacle 実行バイナリです。

%ProgramFiles%\pandora_agent\scripts

Pandora FMS エージェントの起動・停止・再起動スクリプトです。

%ProgramFiles%\pandora_agent\pandora_agent.log

Pandora FMS エージェントがデバッグモードで実行されたときに出力されるログファイルです。

%ProgramFiles%\pandora_agent\util

エージェントプラグインを含むディレクトリです。

%ProgramFiles%\pandora_agent\collections

エージェントのコレクションを含むディレクトリです。

ソフトウエアエージェントの自動デプロイ

自動検出システムを通したデプロイの仕組みを使ってソフトウエアエージェントをデプロイすることができます。より詳細は、 こちらを参照してください。

ソフトウエアエージェントの自動アップグレード

ファイルコレクションと pandora_update ツールを使って、ソフトウエアエージェントは自分自身の更新ができます。

pandora_update ツールは、Perl の Digest:MD5 モジュールが必要です。Perl 5.14 からはデフォルトで含まれていますが、古いバージョンでは手動でインストールしておく必要があります。

これは、次のように動作します。

1. エージェントが、例えば次のようなファイルコレクションの incoming ディレクトリに新たなバイナリを受信します。

Windows の例:

```
c:\program files\pandora_agent\collections\fc_1\PandoraAgent.exe
```

Linux の例:

```
/etc/pandora/collections/fc_1/pandora_agent
```

2. エージェントは、pandora_update プラグインを実行します。このプラグインは、コレクションの短い名前をパラメータとして受け取ります(この例では*fc_1*)。コレクションディレクトリ内のエージェントバイナリをスキャンし、現在動作しているものと比較します。違いがあれば、pandora_update がエージェントを停止し、バイナリを置き換え、新しいバイナリを使ってエージェントを再起動します。

異なるアーキテクチャのアップデートを行うには、それぞれ異なるコレクションを用意する必要があります。例えば、32bit および 64bit の Windows エージェントをアップデートする必要がある場合は、2つのコレクションを作成し、それぞれに対応した *PandoraAgent.exe* バイナリを含める必要があります。

3. Pandora_update はまた、async_string モジュールを使うことによって、次の実行タイミングでユーザに通知を出せるように、エージェントのアップデートプロセスに関するアップデートイベントログを出力します。

以下がアップデートプロセスで使うモジュールです。間隔は長めに設定します。

Unix 通常インストール

```
 module_begin
 module_name Pandora_Update
 module_type async_string
 module_interval 20
 module_exec nohup /etc/pandora/plugins/pandora_update fc_1 2> /dev/null && tail
-1 nohup.out 2> /dev/null
 module_description Module to check new version of pandora agent and update
itself
```

```
 module_end
```

Unix カスタムインストール

```
 module_begin
 module_name Pandora_Update
 module_type async_string
 module_interval 20
 module_exec nohup /var/opt/PandoraFMS/etc/pandora/plugins/pandora_update fc_1
/var/opt/PandoraFMS 2> /dev/null && tail -1 nohup.out 2> /dev/null
 module_description Module to check new version of pandora agent and update
itself
 module_end
```

pandora_update コマンドの 2つ目のパラメータ
は、Pandora FMS のインストールパスです。このパラメー
タは、デフォルトのパスにインストールした際は不要です。

Windows

```
 module_begin
 module_name Pandora_Update
 module_type async_string
 module_interval 20
 module_exec pandora_update.exe fc_1
 module_description Module to check new version of pandora agent and update
itself
 module_end
```

XML ファイルからのエージェント・モジュールの自動作成/学習モード

エージェントは、コンソールから 3つの動作モードを設定できます。

- 学習モード(Learning mode): ソフトウエアエージェントから新たなモジュールを含む XML を受け取ると、それらが自動的に作成されます。これがデフォルト動作です。
- 通常モード(Normal mode): コンソールに存在しない新たなモジュールを XML で受け取っても作成しません。
- 自動無効化モード(Self-disabled mode): 学習モードに似ていますが、すべてのモジュールが不明状態になった場合にエージェントを自動的に無効化します。新たな情報を受け取ったときに改めて自動化します。

エージェント作成時に XML からロードされるデータ

XML を受け取ったときにエージェントの作成とともに自動的に取り込まれるデータは次の通りです。

- エージェント名
- エージェントのIPアドレス

- エージェントの説明
- エージェントの親
- タイムゾーンオフセット
- グループ
- OS
- エージェントの実行間隔
- エージェントのバージョン
- カスタムフィールド
- カスタムID
- URL アドレス
- エージェントモード: 学習、通常、自動無効化

XML を受け取ったときに修正されるデータ (学習モード有効時)

- エージェントの IP アドレス
- エージェントの親
- OSバージョン
- エージェントのバージョン
- タイムゾーン
- カスタムフィールド

GIS データは(GIS が有効の場合)、学習モードが有効である
か無効であるかに関わらず、常に更新されます。

加えて、学習モードが有効の場合XML ファイルで受け取った新たなモジュールは、Pandora で作成されます。

作成時にモジュールに追加されるデータ

初回の各モジュールの XML 受信時に反映されるデータは次の通りです。

- 名前
- タイプ
- 説明
- フィルタの最大・最小値
- 保存倍率
- モジュール実行間隔
- 障害の最大・最小値
- 警告の最大・最小値
- 無効化モジュール
- 単位
- モジュールグループ
- カスタムID
- 文字列の警告・障害
- 障害時手順
- 警告時手順
- 不明時手順

- タグ
- 障害反転モード
- 警告反転モード
- 静観モード
- 連続抑制回数
- アラートテンプレート
- Crontab

モジュールがすでに存在する場合に反映されるデータ

すでに存在するモージュールのデータを XML で受け取った場合は、モジュールのデータに加えて説明および拡張情報のみ更新されます。

Pandora FMS ドキュメント一覧に戻る