CA Spectrum®

ウォッチ ユーザ ガイド

リリース 9.3

CA Technologies 製品リファレンス

このドキュメントでは、CA Spectrum ® について説明します。

CA への連絡先

テクニカル サポートの詳細については、弊社テクニカル サポートの Web サイト（http://www.ca.com/jp/support/）をご覧ください。

目次

第 1 章: ウォッチの操作

このセクションには、以下のトピックが含まれています。

ウォッチについて

*ウォッチ*とは、モデル属性にしきい値を追加するメカニズムです。ウォッチを使用すると、ルータなどのネットワーク エレメントを詳細に監視できます。また、ウォッチはネットワーク分析の他の CA Spectrum ツールで使用できる現在のデータも提供します。

ウォッチ管理は次の 2 つの主要なコンポーネントから構成されています。監視機能を実行する SpectroSERVER 内のインテリジェンス回路と OneClick 管理インターフェースです。 ウォッチ管理は、モデルの［情報］タブにある［しきい値とウォッチ］ビューで使用できます。

任意のタイプの属性にウォッチを動的に適用し、しきい値を基準に属性を監視することで、イベントおよびアラームを生成することができます。ウォッチのプロパティを別のウォッチにコピーすることもできます。 このため、1 番目情報を維持しながら、2 番目のウォッチに新しい情報を追加できます。

ウォッチは、任意のモデル タイプの内部属性および外部属性に設定できます。 また、1 つの属性に対して複数のウォッチを設定できます。 たとえば、デバイスに 2 つのパケット レートしきい値ウォッチを設定できます。 値が 10,000 を超えると、1 つのしきい値によってイエロー アラームが生成され、値が 15,000 を超えると、もう 1 つのしきい値によってレッド アラームが生成されます。

注: ウォッチをセットアップする前に CA Spectrum 管理についてよく理解しておくことをお勧めします。 詳細については、「データベース管理ガイド」、「管理者ガイド」、および「SpectroSERVER パフォーマンス管理ガイド」を参照してください。 また、OneClick インターフェースを理解しておくと役に立ちます。

ウォッチ機能を使用すると、以下のタスクを実行できます。

- 任意のタイプの属性にウォッチを動的に適用できます。
- しきい値と比較して属性を監視し、イベントとアラームを生成します。
- ウォッチ違反が発生したとき、またはウォッチがリセットされたときに、スクリプトを実行します。
- ウォッチのレポートを生成できます。

注: CA Spectrum のあるリリースで作成したウォッチは、それ以降のリリースにマイグレートできます。

[しきい値とウォッチ]サブビュー

OneClick でウォッチを作成、設定、および管理できます。 [しきい値とウォッチ] サブビューにあるテーブルからウォッチを表示し設定します。

注: モデルの [情報] タブから [しきい値とウォッチ] サブビューにアクセスできます。

ウォッチ テーブルには、そのモデルに定義された各ウォッチの情報が表示されます。 ウォッチ ステータス列には、ウォッチの状態が以下の色コードで表示されます。

灰色

ウォッチが非アクティブであることを示します。 ウォッチはアクティブではないので、現在は実行中ではありません。

青

ウォッチの初期状態を示します。 ウォッチはアクティブ化されていますが、まだ初回の実行がされていません。

緑

ウォッチはアクティブ化され、違反なしに実行されています。

イエロー

ウォッチのしきい値違反がありました。

レッド

ウォッチが評価に失敗しました。 原因はテキストに示されています。

ツールバー ボタンでは、以下を実行できます。

- アクティブ化
- 非アクティブ化
- 作成
- 編集
- コピー
- 削除
- ウォッチ情報の表示
- ウォッチ情報の印刷
- ウォッチ テーブルのエクスポート

ウォッチの作成

ネットワーク モデルの選択した属性を監視するウォッチを作成できます。

次の手順に従ってください:

1. モデルの[情報]タブをクリックし、[しきい値とウォッチ]サブビューを展開します。
2. ［ウォッチ］サブビューを展開します。

   ［ウォッチ］テーブルが表示されます。
3. ［ウォッチ］サブビューのツールバーにある  （ウォッチの新規作成）をクリックします。

   ［ウォッチの作成］ダイアログ ボックスが表示されます。

   注:［式］タブはデフォルトで選択されています。
4. 新しいウォッチの名前を入力し、［データ タイプ］リストからデータ タイプを選択します。

   注: このデータ タイプは、ウォッチ宛先属性です。 ウォッチの式は、このデータ タイプとして評価される必要があります。
5. 演算子、関数、およびモデル タイプ属性を組み合わせてウォッチの式を構築します。
   - 式領域の左のボタンをクリックし、ウォッチ式のカーソルの場所に演算子または関数を挿入します。
   - ［属性］ボタンを選択し、必要な属性を選択して挿入します。

     注: ウォッチをアクティブにするには、PollingStatus 属性を True に設定します。

6. ［プロパティ］タブをクリックし、以下の設定を指定します。

デフォルトでアクティブにする

ウォッチを継承するすべてのモデルで、ウォッチをデフォルトでアクティブにするかどうかを指定します。 このオプションが設定されていない場合は、必要なモデルについてウォッチを手動でアクティブにします。

重要: ポーリング対象のウォッチに［デフォルトでアクティブにする］を設定すると、SpectroSERVER のパフォーマンスに悪影響を及ぼす場合があります。

オンデマンドで評価

ウォッチの属性が読み取られた場合のみ、ウォッチの式が評価されます。

変更時に評価

ウォッチの式内のいずれかの属性が変更された場合にウォッチの式が評価されます。 この属性にはメモリ フラグまたはデータベース フラグを設定する必要があります。

ポーリングにより評価

各ポーリング後にウォッチが評価されます。

ポーリング間隔

ポーリングの頻度を秒単位で指定します。 このフィールドは、［ポーリングにより評価］を選択した場合のみ有効になります。

注: ポーリング間隔が 0 に設定されている場合、ウォッチはアクティブにはなりません。

ウォッチを作成するモデル タイプ: *model type*

ウォッチが作成されるモデル タイプを指定します。 異なる親モデル タイプを選択するときは、［参照］をクリックします。

注: ウォッチは、デフォルトの設定では、選択したモデルのモデル タイプに対して作成されます。 ただし、他の派生されたモデル タイプにウォッチを継承させるために、ウォッチの別の親モデル タイプを選択できます。

継承可能にする

選択したモデル タイプから派生したすべてのモデル タイプのモデルで、ウォッチを使用できるようにします。

7. ウォッチにしきい値を設定するには、［しきい値］タブをクリックし、［しきい値を設定］チェックボックスをオンにします。

   ［比較］、［通知］、［スクリプト］オプションが有効になります。

8. 以下の［比較］オプションを指定します。 ［比較］オプションでは、ウォッチによる通知が行われる規制の範囲または制限を決定するしきい値を指定できます。

   値が次の場合はしきい値違反

   しきい値違反で使用される演算子を指定します。 たとえば、「次の値より大きい」を選択します。 サンプルがしきい値属性内の値を超える値を返した場合、しきい値は違反されたと見なされます。 しきい値違反の値には、定数を指定するか、［属性］ボタンをクリックして属性を選択します。

   値が次の場合にしきい値をリセット

   しきい値ステータスが［違反］から［正常］にリセットされるタイミングを決定します。しきい値違反後の各サンプルについては、比較プリミティブを逆方向に使用して、リセット値あウォッチの式と比較されます。 つまり、比較プリミティブが >= （「次の値以上」）の場合、サンプルがこのリセット値より小さい値を返すとすぐにステータスは「正常」にリセットされます。 しきい値の比較が == または != に設定されている場合は、方向に関係なく、値がウォッチ式より大きい場合でも小さい場合でも違反であると考慮されるため、リセット値は適用できません。 このような場合はこのオプションが無効になりますが、それ以外の場合、このエントリは必須です。 しきい値のリセット値には、定数を指定するか、［属性］ボタンをクリックして属性を選択します。

9. 以下の［通知］オプションを指定します。 ［通知］オプションでは、しきい値違反のときに送信される通知の種類を指定できます。

   通知なし

   しきい値違反が発生しても通知は生成されません。

   イベントを生成

   ウォッチ違反が発生したりリセットされたときに CA Spectrum のイベント管理システムによって生成されるイベントを指定します。 このオプションを選択すると、以下に示す、イベント コードのフィールドが有効になります。

   - しきい値に違反した場合 - しきい値違反が発生したときに生成されるイベントを指定します。

- しきい値がリセットした場合 - しきい値がリセットされたときに生成されるイベントを指定します。
- 非アクティブ化/全インスタンス リセットの場合 - このウォッチのすべてのインスタンスがリセットされるか、違反のウォッチが非アクティブ化されたときに生成されるイベントを指定します。

これらのイベントを追加設定して、アラームを生成またはクリアし、イベント ルールおよびプロシージャが実行されるようにできます。 すべてのフィールドでイベントが必要とされるとは限りません。有効なイベント コードが入力されたフィールドに対してのみイベントは生成されます。

アラームを生成

ウォッチのしきい値違反が発生すると、すぐにアラームが生成されます。 汎用イベント（0x480004）も生成されアラームに関連付けられます。ウォッチがリセットされアラームがクリアされると、汎用リセット イベント（0x480005）が生成されます。このオプションを選択すると、以下のフィールドが有効になります。

- 重大度 - アラーム重大度の［マイナー］、［メジャー］、または［重大］を指定します。
- 説明 - アラームの原因説明を指定します。デフォルトでは汎用的な説明が追加されます。［参照］ボタンを使用すると、別の原因説明を選択したり、新しいものを作成できます。新規で作成された説明は、OneClick サーバの $INSTALL/custom/Events/CsPCause ディレクトリに保存されます。複数の OneClick サーバがある場合は、ファイルを手動でコピーします。
- ユーザがクリア可 -- 設定されると、ユーザがウォッチ アラームを手動でクリアできます。
- ユーザがアラームをクリアした場合にウォッチをリセット - アラームを手動でクリアしたとき、ウォッチがリセット値に到達していない場合でもウォッチはリセットされます。これによって、これ以降に違反があるとアラームは再度生成されます。このオプションは［ユーザがクリア可］が選択されている場合のみ使用できます。

10. 以下のスクリプト オプションを指定できます。 スクリプト オプションでは、ウォッチのしきい値違反が発生したりクリアされたときに実行するスクリプトを指定できます。

    しきい値が次の場合にスクリプトを実行

    スクリプトの実行を可能にします。 スクリプトの実行をトリガするしきい値条件を選択します。

    インスタンスごとに実行

    これが設定されていれば、ウォッチ式にリスト属性が含まれる場合、しきい値の状態に合致する各インスタンスごとにスクリプトが実行されます。

    スクリプト ファイル

    実行する SpectroSERVER 上のスクリプト ファイルを指定します。 ディレクトリ パスを指定しない場合、デフォルトのディレクトリである <*$SPECROOT*>/SS-Tools/SwScript が使用されます。 ファイル <*$SPECROOT*>/SS/.watchrc に sw_script_path を設定すると、デフォルトのディレクトリを変更できます。

    注: スクリプトは SpectroSERVER を起動したユーザが実行します。 したがって、そのユーザにはスクリプトにアクセスして実行する権限が必要です。 ウォッチの作成時または修正時に権限の問題が検出された場合は、エラー メッセージが表示されます。 後で権限が変更され、スクリプトの実行時に問題が検出されると、イベントが生成されます。

11. ［ランドスケープ］ボタンをクリックし、ウォッチの宛先ランドスケープを指定します。 OneClick 環境では、単一の分散サーバ環境の複数のランドスケープにウォッチを分散できます。 デフォルトではすべてのランドスケープに新しいウォッチが作成されます。

    ランドスケープのリストが表示されたダイアログ ボックスが開きます。

12. ウォッチを作成しないランドスケープを右に移動します。

    注: ウォッチの作成時にランドスケープが使用できない場合は、そのランドスケープに後でウォッチを追加できます。 ［ウォッチの編集］ダイアログ ボックスを使用して、［ランドスケープ］をクリックします。

13. ［OK］をクリックします。

    これでウォッチが作成されました。

詳細情報:

属性 (P. 40)

アラーム原因の説明の作成および編集

しきい値違反の際に生成されるアラームの原因の説明を作成および編集できます。しきい値違反の通知は CA Spectrum のイベントおよびアラーム機能を通じて配信されます。これらの機能では、外部スクリプト インターフェースを使用して、電話、ポケベル、または電子メールによる通知の配信をサポートします。

次の手順に従ってください:

1. ウォッチを選択して （編集）をクリックします。

   ［ウォッチの編集］ダイアログ ボックスにウォッチが開きます。
2. ［しきい値］タブを選択し、［アラームを生成］を選択します。

   アラーム オプションが表示されます。
3. ［参照］をクリックします。

   ［アラーム原因コードの選択］ダイアログ ボックスが表示されます。アラームの詳細が下部に表示されます。
4. （作成）をクリックして新しいアラーム原因を作成するか、リストからアラーム原因を選択して （編集）をクリックして編集を行います。

   ダイアログ ボックスが表示され、アラーム原因の作成または編集ができるようになります。

   注: ［原因コード］フィールドは読み取り専用です。
5. 必要に応じて、フィールドに入力し、［OK］をクリックします。

   アラーム原因の説明が作成または編集されます。

アラームのコピー

互いにごくわずかしか違いのない複数のアラーム原因説明を作成するには、［コピー］機能を使用します。

次の手順に従ってください:

1. ［ウォッチ］テーブルからウォッチを選択し、  （編集）をクリックします。

   ［ウォッチの編集］ダイアログ ボックスにウォッチが開きます。
2. ［しきい値］タブで［アラームを生成］を選択し、［参照］をクリックします。

   ［アラーム原因コードの選択］ダイアログ ボックスが表示されます。
3. リストからアラーム原因の説明を選択し、  （コピー）をクリックします。

   ［アラーム原因のコピー］ダイアログ ボックスが表示されます。
4. 原因コードを入力します。
5. （オプション）他のアラーム プロパティを編集します。
6. ［OK］をクリックします。

   これで新しいアラーム原因の説明が作成されました。

アラームのスクリプト例

以下のサンプルは、通知として作成できる UNIX スクリプト ファイルの例です。 アラームがしきい値のウォッチと共に設定またはクリアされるたびに、実行のためのスクリプトを指定できます。 この例示的なスクリプトは、*sw_alrm_script* という名前であり、CA Spectrum のインストール時に SS-Tools/SwScript ディレクトリに配置されます。

```
#!/bin/sh
############################################################
This is an example script that can used by users to perform important tasks upon
threshold violation of a watch (Watch Status is #VIOLATED) or when an already violated
watch becomes normal (Watch Status #is NORMAL).
#
# In each of the above cases, the user-specified script is executed (specified during
watch creation) and provides twenty arguments. The arguments in ascending order are:
#
# 1) DATE                       - The date on which the watch was violated.
# 2) TIME                       - The time at which the watch was violated.
# 3) MTYPE_NAME                 - The model type of the watch.
# 4) MODEL_HANDLE               - The modelhandle of the model.
# 5) MODEL_NAME                 - The model of the watch.
# 6) INSTANCE                   - The instance (port, board, etc) for which the watch
is either violated or reset.
# 7) ALARM_ID                   - The ID of the alarm.
# 8) CONDITION                  - The CONDITION of the alarm.
# 9) CAUSE_CODE                 - The cause code of the alarm.
# 10) REPAIR_PERSON             - The repair person assigned to troubleshoot.

# 11) ALARM_STATUS              - The status of the alarm.
# 12) SCRIPT_TYPE               - This is set to "VIOLATED" if the script is getting
executed upon violation and it is set to # “NORMAL” if the script is getting executed
upon reset.
# 13) WATCH_NAME        -  The name of the watch for which the script is executed.
# 14) WATCH_CREATOR             - The name of the creator of the watch.
# 15) WATCH_SRC         - The formula for the watch expression.
# 16) WATCH_SRC_VAL             - The value of the watch expression formula.
# 17) WATCH_REF         - The formula for threshold reference.
# 18) WATCH_REF_VAL             - The value of the threshold reference.
# 19) WATCH_RES         - The formula for threshold reset.
# 20) WATCH_RES_VAL             - The value of the threshold reset.
################################################################
```

```
## Save the argument values into variables that could be used later.
## Remember that DATE is the first argument and WATCH_RES_VAL is the last argument.
# IMPORTANT NOTE:
## Pay particular attention to the variable, “SCRIPT_TYPE”. If the watch has a violated
status, SCRIPT_TYPE is set to “VIOLATED”. If a watch has normal status, SCRIPT_TYPE
is set to “NORMAL”. You can be use this to decide what to do when the watch is violated
and when it is normal.
while [ "$#" -ne 0 ]
do
case "$#"
        in
        20) DATE=`echo "$1"`;;
        19) TIME=`echo "$1"`;;
        18) MTYPE_NAME=`echo "$1"`;;
        17) MODEL_HANDLE=`echo "$1"`;;
        16) MODEL_NAME=`echo "$1"`;;
        15) INSTANCE=`echo "$1"`;;
        14) ALARM_ID=`echo "$1"`;;
        13) CONDITION=`echo "$1"`;;
        12) CAUSE_CODE=`echo "$1"`;;
        11) REPAIR_PERSON=`echo "$1"`;;
        10) ALARM_STATUS=`echo "$1"`;;
        9) SCRIPT_TYPE=`echo "$1"`;;
        8) WATCH_NAME=`echo "$1"`;;
        7) WATCH_CREATOR=`echo "$1"`;;
        6) WATCH_SRC=`echo "$1"`;;
        5) WATCH_SRC_VAL=`echo "$1"`;;
        4) WATCH_REF=`echo "$1"`;;
        3) WATCH_REF_VAL=`echo "$1"`;;
        2) WATCH_RES=`echo "$1"`;;
        1) WATCH_RES_VAL=`echo "$1"`;;
                esac
                shift
done
```

```
## Compose a message that can be used, for example, to send mail.
## The composed message can be used for any other purpose as well.
message()
{
echo "Watch Status Notification"
        echo ""
        echo "Watch Status is $1"
        echo ""
echo "Date:                       $DATE"
echo "Time:                       $TIME"
echo "MType Name:                 $MTYPE_NAME"
echo "Model Handle:               $MODEL_HANDLE"
echo "Model Name:                 $MODEL_NAME"
echo "Instance:                   $INSTANCE"
echo "Alarm ID:          $ALARM_ID"
echo "Condition:         $CONDITION"
echo "Cause Code:                 $CAUSE_CODE"
echo "Repair Person:              $REPAIR_PERSON"
echo "Alarm Status:               $ALARM_STATUS"
echo "Watch Name:                 $WATCH_NAME"
echo "Watch Creator:              $WATCH_CREATOR"
echo "Watch Expression:  $WATCH_SRC"
echo "Watch Expression Value:     $WATCH_SRC_VAL"
echo "Watch Reference:            $WATCH_REF"
echo "Watch Reference Value:      $WATCH_REF_VAL"
echo "Watch Reset:                $WATCH_RES"
echo "Watch Reset Value: $WATCH_RES_VAL"}
## Mail the composed message to the interested users.
## <USER LIST> is the list of mail recipients.
message "$SCRIPT_TYPE" | mail <USER LIST>
```

ウォッチのアクティブ化または非アクティブ化

ウォッチはどのように作成されたかによって、デフォルトでアクティブまたは非アクティブになっています。

システムに追加できるウォッチの数およびフォームに制限がありません。この柔軟性のため、注意をしないとシステム リソースが枯渇してパフォーマンスが低下してしまう可能性があります。 単一のウォッチが最小限のプラットフォーム リソースしか消費せず、ネットワーク トラフィックをほとんど生成しなくても、不要なウォッチは非アクティブにするか、削除することを推奨します。 また、デバイス属性の大きなリストを参照するウォッチをセットアップする場合は注意してください。

単一モデルのウォッチをアクティブ化または非アクティブ化するには、［ウォッチ］テーブルからウォッチを選択し  （ウォッチのアクティブ化）または  （ウォッチの非アクティブ化）をクリックします。 またウォッチを複数選択すると、一度にアクティブ化および非アクティブ化できます。

属性エディタを使用すると、複数のモデルの複数のウォッチをアクティブ化および非アクティブ化できます。 ウォッチ属性を選択し、属性エディタの右側のパネルに移動すると、［ウォッチのアクティブ化］および［ウォッチの非アクティブ化］オプションが有効になります。

注: 属性エディタの詳細については、「IT インフラストラクチャのモデリング/管理 - 管理者ガイド」を参照してください。

コマンド ラインからのウォッチのアクティブ化および非アクティブ化

OneClick コンソール以外に、コマンド ラインからも、ウォッチをアクティブ化および非アクティブ化してデータ プロファイルを表示することができます。

以下のコマンドでは、そのモデル タイプのウォッチ データが表示されます。

```
show watch [mh=model_handle] [lh=landscape_handle]
```

以下のコマンドでは、ウォッチ ステータスが更新されます。

```
update action=action_code [watch=watch_id] [mh=modelhandle]
```

注: 引数 [watch=watch_id] はアクティブ化 (0x00480003) および非アクティブ化（0x00480004）のアクションのみに使用します。

ウォッチの編集

ウォッチを編集して、その式、プロパティ、しきい値を変更することができます。

次の手順に従ってください:

1. モデルの[情報]タブをクリックし、[しきい値とウォッチ]サブビューを展開します。
2. ［ウォッチ］サブビューを展開します。
3. ［ウォッチ］テーブルでウォッチを選択し、  （編集）をクリックします。

   ［ウォッチの編集］ダイアログ ボックスにウォッチが開きます。
4. ウォッチ式、プロパティ、およびしきい値を必要に応じて変更し、［OK］をクリックします。

   注: ウォッチ式、プロパティ、およびしきい値の詳細については、「ウォッチの作成」を参照してください。

   これでウォッチが編集されました。

継承可能なウォッチの編集

継承可能なウォッチを削除する場合、選択したモデル タイプのウォッチ定義はデフォルトでは修正されます。 ウォッチに別の親モデル タイプを選択できます。 ［プロパティ］タブの［モデル タイプ］セクションにある［参照］ボタンを使用します。

ウォッチの元のモデル タイプから派生したモデル タイプでウォッチ定義が変更された場合、そのタイプおよびそれからさらに派生したタイプのモデルにのみ変更が適用されます。 ウォッチの派生元のモデル タイプは変更されません。

たとえば、Gen_IF_Port にて元々作成され、継承可能であるウォッチがあるとします。 Serial_IF_Port は Gen_IF_Port から派生されているので、Serial_IF_Port モデルはそのウォッチを継承します。 ウォッチ定義が Gen_IF_Port モデル タイプで変更された場合、その変更は Serial_IF_Port モデルに伝達されます。 ただし、ウォッチ定義が Serial_IF_Port で変更された場合、新バージョンは Gen_IF_Port での定義を上書きします。 Serial_IF_Port モデルのみが影響を受け、Gen_IF_Port モデルは元のウォッチ定義を保持します。 継承ルールの結果として、派生されたモデル タイプの場合、継承可能なウォッチを異なる方法で再定義または上書きすることができます。

ウォッチのコピー

互いにごくわずかしか違いのない複数のウォッチを作成するには、［コピー］機能を使用します。

次の手順に従ってください:

1. ［ウォッチ］テーブルからウォッチを選択し、  （ウォッチのコピー）をクリックします。

   ウォッチが開きます。

2. 新しいウォッチの名前を入力し、必要に応じてほかの情報を変更して［OK］をクリックします。

   ウォッチはコピーされ、新しいウォッチが作成されます。

ウォッチの削除

関連付けられたしきい値の監視が必要でなくなった場合は、ウォッチを削除できます。

次の手順に従ってください:

1. ［ウォッチ］テーブルでウォッチを選択し、✕（削除）をクリックします。

   ウォッチが複数のランドスケープに存在する場合は、ダイアログ ボックスが表示されます。左側にウォッチが存在しているランドスケープが表示されます。それらのランドスケープからウォッチが削除されます。

2. （オプション）ランドスケープ上のウォッチを保護するには、そのランドスケープを右側に移動します。

3. ［OK］をクリックします。

   左側にリスト表示されたランドスケープからウォッチが削除されます。

ウォッチ情報の表示

ウォッチ情報を表示してウォッチの説明を確認できます。 また、表示された情報は印刷したり、.text または .html ファイルにエクスポートすることができます。

次の手順に従ってください:

1. ［ウォッチ］テーブルのウォッチを選択し、（情報）をクリックします。

   ［ウォッチ レポート］ダイアログ ボックスにウォッチ情報が表示されます。

2. （オプション）［印刷］をクリックすると、ウォッチ情報を印刷できます。
3. （オプション）［エクスポート］をクリックすると、情報をテキストまたは HTML ファイルに保存できます。

   これでウォッチは印刷またはエクスポートされます。

複数のウォッチのレポートの生成

OneClick 管理ページからは、複数のウォッチのレポートを生成できます。

次の手順に従ってください:

1. ブラウザで OneClick ホームページを開き、［管理］タブをクリックします。

   ［管理ページ］が開きます。

2. ［ウォッチ レポート］リンクをクリックします。

   ［ウォッチ レポートの生成］ダイアログ ボックスが表示されます。［ランドスケープの選択］リストおよび［モデル タイプの選択］リストが表示されます。

3. レポートを生成するランドスケープおよびモデル タイプを選択し、［レポートの生成］をクリックします。

   レポートが生成されます。

第 2 章: イベント管理

このセクションには、以下のトピックが含まれています。

イベント

CA Spectrum の高度なイベント処理を説明するために、以下のシナリオを検討します。 テストを目的として、子の数が多い状況を一時的に許可します。 子の数が多い状況が複数分の期間持続された場合にのみアラームを生成し、ウォッチがリセットされたときにはそのようなアラームをクリアします。 EventPair イベント ルールを使用することで、CA Spectrum は以下のタスクを実行できます。

- ChildLimit ウォッチの違反が発生した場合に、初期イベントを生成します。
- 初期イベント後 60 秒以内にリセット イベントが生成されない場合は、アラームを抑制します。
- このウォッチのすべてのインスタンスがリセットまたは非アクティブにされた場合は、別のイベントを生成します。

注: イベント ルールおよびイベントの作成の詳細については、「*Event Configuration User Guide*」を参照してください。

イベントの設定

以下の手順は、イベント生成の例です。 前のトピックの例に基づくと、最初のイベントは「初期の子の数超過」（0xffffffff）で、2 番目のイベントは「リセット」（0xfffffffe）です。 子の数のしきい値に違反し、その後 60 秒以内に子の数のしきい値がリセットされない場合は、「子の数超過」アラーム（イベント/アラーム 0xfffffffd）が生成されます。

次の手順に従ってください:

1. ［ツール］メニューから［ユーティリティ］-［イベント設定］を選択します。

   ［ナビゲーション］、［コンテンツ］、および［詳細］パネルと共に［イベント設定］ページが表示されます。

2. ナビゲーション パネルの  (イベントの作成) をクリックします。

   ［イベント コード］にデフォルト値が入力された状態で［イベントの作成］ダイアログ ボックスが表示されます。

3. (オプション) イベント コードを編集してイベント メッセージを入力し［OK］をクリックします。

   イベントが生成され、詳細が右の［コンテンツ］および［詳細］パネルに表示されます。

4. 初期の「子の数超過」イベント（0xffffffff）を作成し、保存します。

   このイベントによって、アラームおよび「リセット」イベント (0xfffffffe) を条件付きで生成する EventPair ルールがトリガされます。ウォッチは、「リセット」イベントを使用して、子の数の状況の持続性を評価すると共に、リセットされたときに EventPair ルールによって生成されたアラームがあればクリアします。

5. 子の数超過の状況が持続している場合は、イベント「0xfffffffd」と通知を生成するために使用される対応するアラームとを作成および保存します。

6. 初期の「子の数超過」イベント（0xffffffff）用のイベント ペア ルールを作成および保存します。これは、60 秒以内に「リセット」イベント（0xfffffffe）を受信しない場合に「0xfffffffd」イベントおよびアラームを生成します。

   重要: イベント ルールを使用したイベントの処理中に SpectroSERVER が更新されると、処理は完了しません。この機能の起動中に処理されているすべてのイベント ルールはフラッシュされます。

7. ［ファイル］-［すべて保存］を選択します。

   新しいイベントが保存されます。

8. 編集のためにウォッチを開きます。

9. ［しきい値］タブの［通知］の下で［イベントの生成］を選択します。

   イベント コード フィールドが表示されます。

10. ［しきい値に違反した場合］フィールドに、作成済みの初期の「子の数超過」イベント（0xffffffff）を入力します。
11. ［しきい値がリセットした場合］フィールドに、作成済みの、アラームをクリアするための「リセット」イベント（0xfffffffe）を入力し、［OK］をクリックします。

    ウォッチにイベントが関連付けられます。

    LAN_802_3 コンテナに 5 以上のモデルが含まれる場合は、初期の「子の数超過」イベントが生成され、イベント ルールが評価されます。 アラームは、初期イベントの発生からリセット イベントを 60 秒以内に受信しないときのみ生成されます。 ウォッチがリセットされた場合、ウォッチによって生成された「子の数超過」アラームがあれば、クリアされます。

この例の EventDisp エントリは以下のとおりです （必要な場合のみ）。

```
# make sure child count too high (0xffffffff) is
# reset (0xfffffffe) within 60 seconds
# otherwise generate alarming event (0xfffffffd)0xffffffff E 50 R Aprisma.EventPair,
0xfffffffe, 0xfffffffd, 60

# generate an alarm on persistent child count too high0xfffffffd E 50 A 1,0xfffffffd

# if child count drops below threshold, clear any existing alarms0xfffffffe E 50 C
0xfffffffd
```

イベント コード

イベント コードは、イベントごとに CA Spectrum イベント ログに生成されます。

注: ウォッチの作成には、イベントは生成されません。

以下のリストには、イベント コードおよびその説明が含まれます。

0x0048000a

モデルでウォッチをアクティブ化できませんでした

0x00480000

ウォッチが修正されました

0x00480001

ウォッチが破棄されました

0x00480002

ウォッチがアクティブ化されました

0x00480003

ウォッチが非アクティブ化されました

0x00480004

ウォッチのしきい値違反が発生しました

0x00480005

ウォッチのしきい値ステータスがリセットされました（変数リセット値を使用）

0x00480006

ウォッチのインスタンス違反が発生しました

0x00480007

ウォッチの 1 つのインスタンスのしきい値ステータスがリセッされました（変数リセット値を使用）

0x00480010

想定される原因のメッセージが無効です

0x00480012

ログ記録の間隔で競合が発生しました

0x00480013

スクリプトの実行エラーが発生しました（しきい値ウォッチのみ）

0x00480014

モデルのウォッチのしきい値がリセットされました（理由も示されます）

イベント形式ファイル

（<*$SPECROOT*>/SG-Support/CsEvFormat にある）既存のイベント フォーマット ファイル Event00480004 および Event00480005 をテンプレートとして使用して、独自のウォッチ関連のイベント フォーマット ファイルを作成できます。

注: 詳細については、「Event Configuration User Guide」を参照してください。

イベント変数

ウォッチのしきい値違反イベント（0x00480004）およびリセット イベント（0x00480005）には変数がバインドされます。 これは、アラームがすぐに生成されるウォッチおよびイベントが生成されるウォッチの両方に該当します。 クリア イベントには、ウォッチ名を含む ID 番号 2 の変数が 1 つバインドされます。

以下のテーブルに、バインドされる変数およびその内容について示します。

バインドされる変数の ID	内容
2	ウォッチ名
4	しきい値参照値
5	しきい値リセット値（リセット イベントのみ）
6	比較文字列値
7	計算されたウォッチの値
8	ウォッチにインスタンスがある場合はインスタンス オブジェクト ID（OID）、これ以外の場合は空
9	内部。該当のイベント メッセージを完了させるための文字列。
10	内部。該当のイベント メッセージを完了させるための文字列。
11	内部。該当のイベント メッセージを完了させるための文字列。

第 3 章: ウォッチ式の構成ルール

このセクションには、以下のトピックが含まれています。

ウォッチ式

ウォッチ式ではウォッチの監視対象を定義できます。 ウォッチ式は、単純に属性の名前にしたり、複雑な式にしたりすることができます。ウォッチ式には、プリミティブと呼ばれる算術記号およびブール演算子で結びつけられた属性および定数値が含まれます。

式は、プリミティブの以下の優先順位に従って左から右の方向に評価されます。

優先順位	プリミティブ
1	.# .
2	( )
3	DELTA ( ) COUNTER_DELTA ( )
4	!
5	| &
6	* /
7	+ -
8	= = != >= <= < >

詳細情報:

属性 (P. 40)

プリミティブ

プリミティブは、通常、左辺と右辺のある式で使用されます。たとえば、A+B で、+ は、左辺 A と右辺 B の関係を定義しているプリミティブです。カッコは複数の構成要素がある式で評価の順序を示すために使用されます。

使用可能なプリミティブは以下のとおりです。

- +
- -
- *
- /
- ==
- !=
- >
- <
- >=
- <=
- ! (NOT)
- & (AND)
- ,
- |
- (
-)
- TRUE
- FALSE
- TIME
- DELTA
- MIN
- MAX
- INTEGER

- REAL
- UNSIGNED
- COUNTER_DELTA
- UNSIGNED64

プリミティブは機能および用途によってグループ分けできます。 以下のプリミティブは、テキスト文字列または数値式の間でのみ使用できます。

- = = 等しい
- >= 次の値以上
- <= 次の値以下
- != 次の値と等しくない
- > より大きい
- < より小さい

注: これらのプリミティブでは、オブジェクト ID、IP アドレス、およびブール値の属性タイプはサポートしていません。

以下のプリミティブは、数値式のみで使用できます。

- + 加算
- - 減算
- * 乗算
- / 除算

注: これらのプリミティブでは、テキスト文字列、オブジェクト ID、IP アドレス、およびブール値の属性タイプはサポートしていません。 サポートされているタイプは、以下の順序で評価されます。

- 実数
- タイム ティック、日付、ゲージ、カウンタ
- 列挙、整数

2 つのエレメントのレベルが異なる場合、高い方のレベルに合わせて評価されます。 たとえば、5 + 5 は整数として評価され、5 + 5.1 は実数として評価されます。

注: CA Spectrum のモデル タイプでは、符号なし長整数を保存するためにカウンタ属性タイプが使用されます。 たとえば、「テキスト文字列 + UNSIGNED(5)」のように、符号なしのプリミティブを不正に式で使用した場合、カウンタ タイプのエラー メッセージが返されます。

以下のプリミティブは、ブール式のみで使用できます。

- ! 論理否定
- & AND
- | OR

以下のプリミティブは、記号ではなく単語で構成されます。

TIME

1970 年 1 月 1 日 00:00 グリニッジ標準時以降の現在の時刻を数値で示します。

DELTA

サンプリングのたびに属性への変更が計算されます。 たとえば、整数属性 int1 上の 30 秒間隔でのウォッチでは、以下の DELTA 値が生成されることがあります。

間隔	値
0	100, 0
30	1000 900
60	1000 0
90	-1000 -2000
120	2000 3000

DELTA プリミティブは、以下の属性タイプをサポートしています。

- 整数
- 列挙値
- 実数

- 短整数
- タイムティック
- 日付
- ゲージ
- カウンタ

値が小さくなる可能性のないもの（タイムティックおよびカウンタなど）に関しては、COUNTER_DELTA を使用します。

DELTA 式で属性のインスタンスを示す方法は以下のとおりです。

- DELTA (If_In_Octets.2)
- DELTA (If_Out_Octets.#)
- DELTA (TIME) もサポートされています。

注: DELTA (TIME) とは、ウォッチによる現在および前回の評価の差です。これは内部属性を使用する計算でのみ有用です。 外部属性を使用するレートの計算を生成するには、たとえば DELTA (XflDELTA (Sys_Up_Time) などのようにデバイスの時間を使用します。この場合、Sys_Up_Time はデバイスの時間カウンタです。

COUNTER_DELTA

増分される符号なし整数であると想定されている属性の変化を計算します。 この値はゼロにリセットされる場合もありますが、このプリミティブの結果は常に正の符号なし整数です。 負の属性値は符号なしの大きな値として扱われます。

COUNTER_DELTA プリミティブは、DELTA と同じ属性タイプをサポートするので、COUNTER_DELTA 式では DELTA での説明と同様の方法でインスタンスを指定できます。COUNTER_DELTA (TIME) もサポートされています。

ATTR

プリミティブに続くカッコ内の 16 進数値が属性識別子であることを示します。以下の例のように名前の重複がある場合、このプリミティブで属性を一意に識別できます。

ATTR (<*attr_id*>)
ATTR (<*attr_id*>).<*instance id*>
ATTR (<*attr_id*>).#

<attr_id> は、「0x」または「0X」の後に 1 ～ 8 の 16 進数値が続く値です。

TRUE

ブール定数 True を定義します。

FALSE

ブール定数 False を定義します。

MAX

たとえば (attribute x+1, attribute y-5) など、かっこ内のカンマで区切られた 2 つの式のうちの大きい方を指定します。この式は、オペランド、属性、またはプリミティブの任意の組み合わせで、数値（テキストでない文字列、オクテット、ブール値）を返します。

MIN

たとえば (attribute x+1, attribute y-5) など、かっこ内のカンマで区切られた 2 つの式のうちの小さい方を指定します。この式は、オペランド、属性、またはプリミティブの任意の組み合わせで、数値（テキストでない文字列、オクテット、ブール値）を返します。

以下のプリミティブ（ポップアップ選択メニューには表示されません）を使用して、ウォッチによって監視しているリスト属性のインスタンスを指定できます。

.

後に続く入力値によってリスト属性の特定のインスタンスを識別します。

たとえば、If_In_Octets.2 は If_In_Octets 属性の 2 番目のインスタンスを意味します。IP アドレス テーブルの場合、指定子は 1 桁の数字ではなくアドレス全体である場合があります。

ウォッチがアクティブ化されているモデルのどれにも、属性の指定したインスタンスが存在しない場合、ウォッチは失敗します。 一部のモデルにはインスタンスがあり、その他にはない場合、存在するモデルではウォッチは成功し、その他では失敗します。

.#

ウォッチの詳細ビューに表示される現在のインスタンス指定子の値によって監視対象のインスタンスが決まることを示します。

たとえば、インスタンス指定子の値が［すべて］の場合、If_In_Octets.# という式は、If_In_Octets 属性のすべてのインスタンスにウォッチを適用します。インスタンス指定子の値が［範囲］（1-3）の場合は、インスタンス、1、2、3 のみが監視されます。

インスタンスの範囲が指定されている場合、その範囲内のインスタンスの 1 つがモデルのいずれかになければ、ウォッチは失敗します。 逆に、範囲内のすべてのインスタンスが存在するすべてのモデルでは、ウォッチが成功します。

データ タイプ

以下のテーブルは、さまざまなタイプの属性に割り当てることができるデータ タイプ値を示します。

注: CA Spectrum では、データ タイプが「オクテット文字列」の属性に対するウォッチはサポートしていませんが、この属性はテキスト文字列データ タイプを使用して監視できます。

式結果タイプ	使用可能な宛先属性タイプ
ブール値	任意の数値タイプ

式結果タイプ	使用可能な宛先属性タイプ
テキスト文字列	テキスト文字列
整数	整数、列挙、実数
列挙	整数、列挙、実数
実数	整数、列挙、実数
日付	実数、日付、タイム ティック、カウンタ、ゲージ、COUNTER64
タイム ティック	実数、日付、タイム ティック、カウンタ、ゲージ、COUNTER64
カウンタ	実数、日付、タイム ティック、カウンタ、ゲージ、COUNTER64
ゲージ	実数、日付、タイム ティック、カウンタ、ゲージ、COUNTER64
オブジェクト ID	オブジェクト ID
IP アドレス	IP アドレス
COUNTER64	COUNTER64、実数

たとえば、Int1 という名前の整数属性では、Int1 = 50.5（キャストが必要）という式は許可されますが、 Int1 = 「文字列」は許可されません。

これらのルールは、ブール値およびテキスト文字列の属性に値を割り当てるときに最も柔軟です。いずれの式も 0 および 1 に評価し、ブール値属性に記述することができます。 たとえば、ブール属性として Bool1 が与えられた場合、式 Bool1 = 500 * 50 + 450 は、Bool1 に TRUE を代入します。 式 Bool1 = (500 * 50 + 450) * 0 は、Bool1 に FALSE を代入します。 同様に、すべての式はテキスト文字列として評価できます。 テキスト文字列属性として str1 がある場合、str1 = 500 + 50 の式では str1 に文字列 "550" が割り当てられます。

注: テキスト文字列属性タイプでは、引用符で囲まれたテキスト文字列、および引用符に囲まれていない算術式を使用できます。たとえば、500 + 50 は、文字列 "550" になります。

定数

式フィールドには、属性値および演算子と同様に定数を直接に入力できます。 式に、以下のタイプの定数を入力できます。

- 符号なしの整数。1 以上の桁数の正の値。

- 実数。ゼロかそれ以上の連続した数字の後に小数点が続き、その後に 1 つ以上の連続した数字が続く実数。たとえば、.7、1.7、または 23.24（7 ではない）。
- 符号付きの整数。1 桁以上の正または負の値。
- 二重引用符によって囲まれた文字の任意のシーケンスによって表されるテキスト文字列。 たとえば、"a string"、 "5.25"、または " "。

キャスト演算子

キャストでは、1 つのデータ タイプが別のデータ タイプに変換されます。キャストを行う際、キャストを行う値の一部が失われるリスクがあります。変換前のデータ タイプの範囲が、変換後のデータ タイプの範囲と適合するときにはキャストは不要です。

たとえば、「-1.79769 e+308 <= -2,147,483,648」および 2,147,483,647 <= 1.79769 e+308」であるため、キャストを行わなくても実数に整数を割り当てることができます。 しかし、カウンタに整数を割り当てる場合は範囲が重ならないので、キャストを行う必要があります。

キャストは、使用しているデータ タイプで範囲のオーバーラップがある場合に正しく動作します。たとえば、整数 5 をカウンタにキャストすると期待どおりに動作します。 しかし、負の数は「カウンタ」の範囲内に収まりません。したがって、「カウンタ」に -5 をキャストすると、コンピュータが数を表現する方法によって、4,294,967,291 という符号なし（正）の結果がもたらされます。 逆に、整数に 2,147,483,647 より大きいカウンタ値をキャストすると、負の数が生成されます。

以下の演算子を使用すると、かっこで囲まれた式の結果を、選択したデータ タイプにキャストできます。

演算子	キャストの結果
UNSIGNED	通常 27 は符号付きの整数として処理されますが、「UNSIGNED (27)」は符号なし整数 27 と解釈されます。 式内のこの演算子の後の負の数は、符号なしと解釈されます。 例： UNSIGNED (-5) は 5 と解釈されます。 -5 の数は符号なしで 4294967291 の値となります。
INTEGER	式内のこの演算子に続く任意の実数は、最も近い整数に切り上げられるか切り下げられます。例: INTEGER (2.4) は 2 と解釈され、INTEGER (2.6) は 3 と解釈されます。

演算子	キャストの結果
REAL	例： REAL (3) は 3.0 と解釈されます。

リテラル数値のデータ タイプ

ウォッチ式の数値リテラルは、データ タイプを判別するために一連の範囲と比較されます。 以下のテーブル内のタイプ列は数値リテラルの実際のデータ タイプであり、その列セルにリスト表示された他のタイプと直接交換可能です。

以下の表は、リテラル数値のデータ タイプがどのように決定されるかを示します。

最小	最大	タイプ
-2,147,483,648	2,147,483,647	整数、ブール値*、列挙
0	4,294,967,295	カウンタ、日付**、ゲージ、タイムティック
0	18,446,744,073,709,551,615	COUNTER64
-1.79769 e+308	1.79769 e+308	実数

* ブール値データ タイプには、定数の TRUE および FALSE も使用できます。ゼロでないすべての値は TRUE と同等です。

** 1970 年 1 月 1 日 00:00:00 UTC(0) 以降の秒数です。

属性

ウォッチ式の数式に入力する属性は、常に文字から始まり、文字、数字、およびアンダースコア文字（_）の組み合わせから構成される必要があります。 プリミティブと同一の名前（TRUE、FALSE、TIME、DELTA、COUNTER_DELTA）を持つ属性は、一重引用符で囲みます。 また ATTR プリミティブを使用して属性 ID で属性を指定できます。

ウォッチ式 (P. 31)は、単一の属性で構成することもできます。複数の属性を使用する式を作成する場合は、ボタン パレットから演算子を選択し、別の属性を選択するという具合に作成していきます。

式でリスト属性を指定した場合は、インスタンス情報を含める必要があります。インスタンスを指定するには、'.' とその後に続けるインスタンス ID 番号（たとえば、**ifInOctets.2**）を追加します。 または、'.#'（たとえば、ifInOctets.#）を追加し、［インスタンス］フィールドで［すべて］または［範囲］のいずれかを選択します。 ［範囲］を指定した場合、［開始］および［終了］フィールドに最小と最大の ID を指定します。 リスト属性は［属性セレクタ］ダイアログ ボックスの［タイプ］列で「 [] 」を使用して示されます。

ウォッチ式の一部として入力する属性のほかに、ウォッチ情報および宛先属性を保存する場合に必要な属性を動的に作成することもできます。

インスタンス識別子

ウォッチ式の数式でリスト属性を使用している場合は、インスタンス識別子を使用して特定のオブジェクトを指定できます。

リスト属性では、オブジェクトを完全に指定するためにインスタンス ID を必要とします。 ウォッチ式では、特定のインスタンス、インスタンス範囲、属性のすべてのインスタンスを指定できます。OID が構築されると、最大 2 つの異なるサブ OID（標準の SpectroSERVER OID メカニズムまたはウォッチ インスタンス指定子）が使用されます。 SpectroSERVER は OID 接頭辞および OID 参照（オプション）から OID を作成します。［式］ページでウォッチ式にインスタンスを指定すると、以下のように OID に追加されます。

```
OID = OID 接頭辞 + OID 参照 + ウォッチ インスタンス
```

インスタンス指定子を使用して範囲を指定した場合、その範囲の各インスタンスの結果が計算されます。 指定した範囲が実際のインスタンスより広範囲である場合、実際のインスタンスのみの結果が計算されます。 つまり、範囲として 1 ～ 10 を指定し、インスタンスが 3 つしか存在しない場合は、3 つの結果が計算されます。

たとえば、ルータの If_In_Octet にウォッチを設定します。 インスタンス指定子として 1 ～ 3 の範囲を提供します。 CA Spectrum データベースによって使用される If_In_Octets の OID 接頭辞は 1.3.6.1.2.1.2.2.1.10 なので、以下の OID が割り当てられます。

- OID = 1.3.6.1.2.1.2.2.1.10 + .1
- OID = 1.3.6.1.2.1.2.2.1.10 + .2
- OID = 1.3.6.1.2.1.2.2.1.10 + .3

インスタンス指定子に「すべて」を使用した場合、If_In_Octets オブジェクトのすべてのインスタンスが動的に決定されます。 また、ウォッチ式内で特定のインスタンスを指定したり、インスタンス指定子「範囲」または「すべて」を使用して 1 つのオブジェクトの特定のインスタンスを複数指定できます。 オブジェクトのインスタンスを指定するために Instance _of プリミティブ（"."）を使用すると、そのオブジェクトのインスタンス識別子は無視されます。

たとえば、以下の式では、Instance_of プリミティブ（とその後の値）を使用して、If_In_Errors オブジェクトのインスタンス番号 3 が指定されています。また、.# プリミティブは、現在のインスタンス指定子を If_In_Octets オブジェクトに使用することを示しています。

```
If_In_Errors.3 / If_In_Octets.#
```

現在のインスタンス指定子は「すべて」であり、If_In_Errors と If_In_Octets は両方ともリスト属性で、その OID はそれぞれ 1.3.6.1.2.1.2.2.1.14 と 1.3.6.1.2.1.2.2.1.10 であると仮定します。 各インスタンスの分割処理は、以下の OID を使用して実行されます。

```
If_In_Errors OID = 1.3.6.1.2.1.2.2.1.14 + .3
If_In_Octets OID = 1.3.6.1.2.1.2.2.1.10 + .1

If_In_Errors OID = 1.3.6.1.2.1.2.2.1.14 + .3
If_In_Octets OID = 1.3.6.1.2.1.2.2.1.10 + .2

If_In_Errors OID = 1.3.6.1.2.1.2.2.1.14 + .3
If_In_Octets OID = 1.3.6.1.2.1.2.2.1.10 + .n
```

If_In_Errors 属性 OID のインスタンス指定子の代わりに、Instance_of プリミティブが使用されている点に注意してください。 「すべて」が選択されているために各属性ですべてのインスタンスを使用するのではなく、Instance_of プリミティブによって特定のインスタンスが指定されています。 ただし、特定のウォッチでは、インスタンス指定子によってインスタンス数が決定されます。

モデルの INSTANCE_ID およびインスタンス指定子は、ウォッチ式の各リスト属性に適用されます。 ウォッチ式に異なるインスタンスを使用する属性が含まれる場合、ウォッチを追加するときにエラーが返されます。

たとえば、インスタンスとしてボード番号を使用する属性が含まれる 1 つの数式と、インスタンスとしてボード ポートを使用する別の数式を入力した場合は、エラーが返されます。

ウォッチ定義属性

モデルにウォッチを作成した場合、そのウォッチの詳細を保存するための属性が作成されます。 属性はコンピュータで生成された名前を持っています。この名前は、モデルが属するモデル タイプ下のウォッチ定義グループにあるアクティブなデータベース開発者 ID（登録されたものかデフォルト）を使用して作成されます。

ウォッチ宛先属性

モデルにウォッチを作成した場合、ウォッチと同じ名前およびデータ タイプで宛先属性も作成されます。 この属性は、他のモデル タイプ属性と似ています。 属性が読み取られると、ウォッチ式が自動的に評価されます。 評価の結果は読み取り操作の結果になります。

しきい値の参照およびリセットの互換性

しきい値のウォッチでは、しきい値およびリセット式のタイプが、ウォッチ式と互換性のあるタイプである必要があります。

以下の表に、結果タイプと対応する使用可能なタイプを示します。

結果タイプ	使用可能なタイプ
ブール値	ブール値
テキスト文字列	テキスト文字列
整数	整数、列挙、実数
列挙	整数、列挙、実数
実数	実数
日付	実数、日付、タイム ティック、カウンタ、ゲージ、COUNTER64
タイム ティック	実数、日付、タイム ティック、カウンタ、ゲージ、COUNTER64
カウンタ	実数、日付、タイム ティック、カウンタ、ゲージ、COUNTER64
ゲージ	実数、日付、タイム ティック、カウンタ、ゲージ、COUNTER64
オブジェクト ID	オブジェクト ID
IP アドレス	IP_ADDR
COUNTER64	COUNTER64、実数

付録 A: ウォッチのタイプの例

このセクションには、以下のトピックが含まれています。

ウォッチ シナリオ 1

以下のパラメータは、Hub_CSI_IRBM にウォッチを確立します。このウォッチは、60 秒ごとに、転送されたフレームの総数および受信された衝突の総数を確認します。しきい値 100 万を超えた場合は、「衝突数過多」というメッセージを伴なうマイナー アラームが生成されます。 しきい値リセット値は、衝突の総数が 500,000 に下がった場合に、しきい値ステータスが違反から「正常」にリセットされる（およびアラームはクリアされる）ことを示します。

このシナリオのウォッチは、以下のように作成します。

1. カウンタ タイプの非アクティブ ウォッチを作成し、HubColls_v という名前を付けます。 式に衝突の総数として 1000000 を割り当てます。
2. カウンタ タイプの非アクティブ ウォッチを作成し、HubColls_r という名前を付けます。 式に衝突の総数として 500000 を割り当てます。
3. カウンタ タイプのアクティブなウォッチを作成し、HubColls と名付け、以下のウォッチ例のとおり値を割り当てます。

例: ウォッチ

最初のウォッチは、以下のパラメータから構成されます。

- 名前： HubColls_v
- データ タイプ： カウンタ
- 式：
  - 式： 1000000
- インスタンス： なし
- プロパティ
  - デフォルトでアクティブにする： オフ
- しきい値： なし

2 番目のウォッチは、以下のパラメータから構成されます。

- 名前： HubColls_r
- データ タイプ： カウンタ
- 式：
  - 式： 500000
- インスタンス： なし
- プロパティ
  - デフォルトでアクティブにする： オフ
- しきい値： なし

3 番目のウォッチは、以下のパラメータから構成されます。

- 名前： HubColls
- データ タイプ： カウンタ

- 式
  - 式： Hub_Trans_Coll+Hub_Rec_Colls
- インスタンス： なし
- プロパティ
  - デフォルトでアクティブにする： オン
  - 評価： ポーリングにより評価
  - ポーリング間隔： 00:01:00
- しきい値
  - しきい値を設定（オン）- 値が次の場合はしきい値違反： > HubColls_v (HubColls_v Inactive というカウンタ ウォッチを作成します。式は 1000000 です。)
  - 値が次の場合にしきい値をリセット： HubColls_r（HubColls_r Inactive という名前のカウンタ ウォッチを作成します。式は 500000 です。）
  - 属性、HubColls_v、および HubColls_r は、HubColls ウォッチの作成前に作成しておく必要があります。
  - アラームを生成（オン）- 重大度： マイナー、アラーム説明： CollsExceeded（「衝突が多すぎます。」というメッセージで CollsExceeded という新しいアラームを作成します。）

ウォッチ シナリオ 2

ネットワーク管理者が、サーバ上で CD-ROM ドライブを除いたディスク使用率のレベルを監視したいと考えています。CD-ROM ドライブでは、通常、CD がセットされていなければ、使用率は 0% と示されますが、フル状態の CD がセットされていると 100% に近い使用率で示されます。 ただし、このような情報は、管理者にとっては有用ではありません。

管理者はディスク使用率をチェックするために、デバイス タイプを OID として返す RFC2790App を使用したいと考えています。 各 OID は、それぞれ異なるデバイス タイプにマップされます。 この MIB 内のストレージ タイプと照合して確認する必要がある式はいずれも、その OID を比較に使用する必要があります。 （文字列を OID と比較できます）。

以下の表は、RFC2790App のデバイス タイプと OID マッピングを示します。

デバイス タイプ	OID
hrStorage	1.3.6.1.2.1.25.2
hrStorageTypes	1.3.6.1.2.1.25.2.1
hrStorageOther	1.3.6.1.2.1.25.2.1.1
hrStorageRam	1.3.6.1.2.1.25.2.1.2
hrStorageVirtualMemory	1.3.6.1.2.1.25.2.1.3
hrStorageFixedDisk	1.3.6.1.2.1.25.2.1.4
hrStorageRemovableDisk	1.3.6.1.2.1.25.2.1.5
hrStorageFloppyDisk	1.3.6.1.2.1.25.2.1.6
hrStorageCompactDisk	1.3.6.1.2.1.25.2.1.7
hrStorageRamDisk	1.3.6.1.2.1.25.2.1.8
hrStorageFlashMemory	1.3.6.1.2.1.25.2.1.9
hrStorageNetworkDisk	1.3.6.1.2.1.25.2.1.10

管理者は RFC2790App モデル用にウォッチを 2 つ作成する必要があります。1 番目のウォッチでは、ブール値を設定します。このブール値は、監視対象から CD-ROM ドライブ（OID 1.3.6.1.2.1.25.2.1.7）を除外するために 2 番目のウォッチで使用されます。 2 番目のウォッチは、ホスト上のほかのタイプのストレージ デバイスのディスク使用率（パーセントで表示）を監視します。 この例では、使用率が 90% を超えるとアラームが送信されます。

例: ウォッチ

最初のウォッチは、以下のパラメータから構成されます。

- 名前： isNotCDROM
- データ タイプ： ブール
- 式： 式： hrStorageType.# != 1.3.6.1.2.1.25.2.1.7
- インスタンス： すべて

- プロパティ
  - デフォルトでアクティブにする： オン
  - 評価： オンデマンドで評価
- しきい値： なし

2 番目のウォッチは、以下のパラメータから構成されます。

- 名前： PctDiskUsed
- データ タイプ： 実数
- 式
  - 式： REAL(hrStorageUsed.#)/REAL(hrStorageSize.#) * 100 * REAL(isNotCDROM.#)
  - インスタンス： すべて
- プロパティ
  - デフォルトでアクティブにする： オン
  - 評価： ポーリングにより評価
  - ポーリング間隔： 30 秒
- しきい値
  - 値が次の場合はしきい値違反 >= <*許容レベルのディスク使用率を設定*>
- アラーム
  - アラーム重大度： マイナー
  - アラーム説明： DiskUtilAlarm
  - ユーザがクリア可能なアラーム
  - ユーザがアラームをクリアした場合、ウォッチをリセットしない
- スクリプト： なし

変更時に評価のウォッチ

以下に「変更時に評価」（EoC）ウォッチの例を示します。これらのウォッチでは、ビューがいつ編集されたかを知ることができます。結果として、管理者は、LAN が編集された時期と、新しいモデルが LAN に追加された時期を正確に表示する検索を実行できます。 両方の属性とも変更時に評価されます。これらの属性は通常あまり頻繁には変更されません。

例: 変更時に評価のウォッチ

最初のウォッチは、以下のパラメータから構成されます。

- 名前： Child_Count_Watch
- データ タイプ： 整数
- 式
  - 式： Child_Count
  - インスタンス： なし
- プロパティ
  - デフォルトでアクティブにする： オン
  - 評価： 変更時に評価
  - ポーリング間隔： なし
- しきい値： なし

2 番目のウォッチは、以下のパラメータから構成されます。

- 名前： Edit_Count
- データ タイプ： 整数
- 式
  - 式： EDIT_COUNT
  - インスタンス： なし
- プロパティ
  - デフォルトでアクティブにする： オン
  - 評価： 変更時に評価
- しきい値： なし

ポーリングしきい値のウォッチ

以下の例では、「ポーリングしきい値」ウォッチを作成して、LAN_802_3 内の子の数が 5 を超えるごとにマイナー アラームを生成します。子の数が 5 より少なくなった場合、ウォッチはアラームをクリアします。

まず属性ウォッチ ChildLimit_v （任意の名前）を作成して、式として 「5」を入力します。 ウォッチを作成することは、ウォッチの詳細を定義することになります。 次に、Child_Limit という名前のポーリングしきい値ウォッチを作成します。

例: ポーリングしきい値ウォッチ

最初のウォッチは、以下のパラメータから構成されます。

- 名前： ChildLimit_v
- データ タイプ： カウンタ
- 式： 5
- プロパティ： なし
- しきい値： なし

2 番目のウォッチは、以下のパラメータから構成されます。

- 名前： ChildLimit
- データ タイプ： 整数
- 式
  - 式： Child_Count
- インスタンス： なし
- プロパティ
  - デフォルトでアクティブにする： オン
  - 評価： ポーリングにより評価
  - ポーリング間隔： 00:01:00
- しきい値
  - しきい値を設定（オン）
  - 値が次の場合はしきい値違反： > ChildLimit_v

- 値が次の場合にしきい値をリセット： <= ChildLimit_v
- アラームを生成（オン）
- 重大度： マイナー
- アラーム説明： ExceedChildLimit （「子は 5 つまで許可されます。」というメッセージで ExceedChildLimit という名前の新しいアラームを作成します。）

オンデマンド ウォッチ シナリオ 1

オンデマンド ウォッチは、そのウォッチから情報が要求されるたびに評価されます。

以下は、ポーリングしきい値ウォッチと共に動作する単純なオンデマンド ウォッチの例で、Cisco ルータ（モデル タイプ： Rtr_CiscoIGS）でメモリが不足しそうになるとアラームを生成します。属性 freeMem は 1500 未満で、2500 を超えるとクリアされます。

例: ポーリングしきい値ウォッチと共に動作する単純なオンデマンド ウォッチ

最初のウォッチは、以下のパラメータから構成されます。

- 名前： MemLow
- データ タイプ： 整数
- 式
  - 式： 1500
  - インスタンス： なし
- プロパティ
  - デフォルトでアクティブにする： オフ
  - 評価： オンデマンドで評価
- しきい値： なし

2 番目のウォッチは、以下のパラメータから構成されます。

- 名前： MemHigh
- データ タイプ： 整数

- 式
  - 式： 2500
    - インスタンス： なし
- プロパティ
  - デフォルトでアクティブにする： オフ
  - 評価： オンデマンドで評価
- しきい値： なし

3 番目のウォッチは、以下のパラメータから構成されます。

- 名前： Mem_Good
- データ タイプ： 整数
- 式
  - 式： freeMem
  - インスタンス： なし
- プロパティ
  - デフォルトでアクティブにする： オン
  - 評価： ポーリングにより評価
  - ポーリング間隔： 1 分
- しきい値
  - 値が次の場合はしきい値違反： <= MemLow
  - 値が次の場合にしきい値をリセット： > MemHigh
- アラーム
  - アラーム重大度： メジャー
  - アラーム説明： Rtr+Memory
- スクリプト： なし

オンデマンド ウォッチ シナリオ 2

オンデマンド ウォッチは、そのウォッチから情報が要求されるたびに評価されます。

以下のウォッチは、Cisco ルータ（モデル タイプ： Rtr_ Cisco）のパフォーマンスが徐々に低下するのに対し、色付きのアラーム状態でネットワーク管理者に示すよう設計されています。 マイナー アラームの値は、50 ～ 59 です。メジャー アラームの値は、60 ～ 69 です。重大アラームの値は 70 を超えています。

例: オンデマンド ウォッチ

最初のウォッチは、以下のパラメータから構成されます。

- 名前： True_Ref
- データ タイプ： ブール
- 式
  - 式： 1
  - インスタンス： なし
- プロパティ
  - デフォルトでアクティブにする： オフ
  - 評価： オンデマンドで評価
- しきい値： なし

2 番目のウォッチは、以下のパラメータから構成されます。

- 名前： Minor_Busy
- データ タイプ： ブール
- 式
  - 式： ((busyPer >= 50) & (busyPer <60))
  - インスタンス： なし
- プロパティ
  - デフォルトでアクティブにする： オン
  - 評価： ポーリングにより評価
  - ポーリング間隔： 1 分

- しきい値
  - 値が次の場合はしきい値違反： > True_Ref
  - 値が次の場合にしきい値をリセット： <= True_Ref
- アラーム：
  - アラーム重大度： マイナー
  - アラーム説明： RtrBusy1
  - スクリプト： なし

3 番目のウォッチは、以下のパラメータから構成されます。

- 名前： Major_Busy
- データ タイプ： ブール
- 式
  - 式： ((busyPer >= 60) & (busyPer < 70))
  - インスタンス： なし
- プロパティ
  - デフォルトでアクティブにする： オン
  - 評価： ポーリングにより評価
  - ポーリング間隔： 1 分
- しきい値
  - しきい値： 値が次の場合はしきい値違反： == True_Ref
  - 値が次の場合にしきい値をリセット： != True_Ref
- アラーム
  - アラーム重大度： メジャー
  - アラーム説明： RtrBusy 2
  - スクリプト： なし

4 番目のウォッチは、以下のパラメータから構成されます。

- 名前： Critical_Busy
- データ タイプ： ブール

- 式
  - 式： (busyPer >= 70)
  - インスタンス： なし
- プロパティ
  - デフォルトでアクティブにする： オン
  - 評価： ポーリングにより評価
  - ポーリング間隔： 60
- しきい値
  - 値が次の場合はしきい値違反： == True_Ref.
  - 値が次の場合にしきい値をリセット： ! = True_Ref.
- アラーム：
  - アラーム重大度： 重大
  - アラーム説明： RtrBusy3
  - スクリプト： なし

オンデマンド ウォッチ シナリオ 3

オンデマンド ウォッチは、そのウォッチから情報が要求されるたびに評価されます。

このシナリオでは、フレーム リレーで障害が発生し、ダイアル バックアップ回線がアクティブ化されるたびに、ネットワーク管理者がアラームを受け取るようにするとします。 ダイアル バックアップは ISDN インターフェース タイプ 21（基本 ISDN サービス）で行われます。 以下のウォッチは、いずれかの ISDN ポートでトラフィックが受信されるかどうかを確認します。トラフィックが流れている場合は、Cisco ルータ（モデル タイプ：Rtr_Cisco）にマイナー アラームが付きます。

例: オンデマンド ウォッチ

最初のウォッチは、以下のパラメータから構成されます。

- 名前： True_Ref
- データ タイプ： ブール

- 式
  - 式： 1
  - インスタンス： なし
- プロパティ
  - デフォルトでアクティブにする： オフ
  - 評価： オンデマンドで評価
- しきい値： なし

2 番目のウォッチは、以下のパラメータから構成されます。

- 名前： ISDN_Backup
- データ タイプ： ブール
- 式
  - 式： ((COUNTER_DELTA (ifInOctets.#) > 0) & (ifType.# == 21))
  - インスタンス： すべて
- プロパティ
  - デフォルトでアクティブにする： オン
  - 評価： ポーリングにより評価
  - ポーリング間隔： 1 分
- しきい値
  - 値が次の場合はしきい値違反： == True_Ref
  - 値が次の場合にしきい値をリセット： != True_Ref
- アラーム：
  - アラーム重大度： マイナー
  - アラーム説明： Backup_Active
- スクリプト： なし

オンデマンド ウォッチ シナリオ 4

オンデマンド ウォッチは、そのウォッチから情報が要求されるたびに評価されます。

このシナリオでは、ウォッチが ISDN インターフェース モデルの ifOutOctets 値を監視し、この値が増加したかどうか、ゼロ以外であるかどうかを確認します。 この値はインターフェースがアクティブであるかどうかを示します。 要件に合うようにポーリング、ログ記録、アラーム重大度、およびアラーム テキストを変更できます。 2 番目のウォッチでは True または False（1/0）の識別子を使用します。1 はインターフェースが ifOutOctets を送信していることを示し、0 は送信していないことを示します。

例: オンデマンド ウォッチ

最初のウォッチは、以下のパラメータから構成されます。

- 名前： isISDN
- データ タイプ： ブール
- 式
  - 式： (X_ifType == 21)
  - インスタンス： なし
- プロパティ
  - デフォルトでアクティブにする： オン
  - 評価： オンデマンドで評価
  - 継承可能： オフ
- しきい値
  - しきい値： なし

2 番目のウォッチは、以下のパラメータから構成されます。

- 名前： ISDN_Up
- データ タイプ： 整数

- 式
  - 式： MAX(0, MIN(1, INTEGER(((COUNTER_DELTA (X_OutOctets) * INTEGER(isISDN)) > 0))))
  - インスタンス： なし
- プロパティ
  - デフォルトでアクティブにする： オン
  - 評価： ポーリングにより評価
  - ポーリング間隔： 0+00:05:00
  - 継承可能： オフ
- しきい値
  - 値が次の場合はしきい値違反： == 1
  - 値が次の場合にしきい値をリセット： != 1
- アラーム
  - アラーム重大度： マイナー
  - アラーム説明： ErrorTholdAlarm
  - ユーザがクリア可能なアラーム
  - ユーザがアラームをクリアした場合、ウォッチをリセットしない
- スクリプト： なし

使用効率およびテスト用ウォッチ シナリオ 1

ここで説明するウォッチの例は、使用効率およびテスト用です。

このシナリオでは、2 つのウォッチを作成します。1 番目は WatchLoad_v という名前の属性を作成し、式の値を .75 にします。2 番目は CPULoadRate 属性を使用します。 以下の例では、たとえば「CPU 負荷が 75% を超過しました。」などのアラーム説明を WatchLoad_Alarm という名前で作成します。

例: 使用効率およびテスト用のウォッチ

最初のウォッチは、以下のパラメータから構成されます。

- 名前： WatchLoad_v
- データ タイプ： 実数

- 式
  - 式： .75
  - インスタンス： なし
- プロパティ
  - デフォルトでアクティブにする： オフ
  - 評価： オンデマンドで評価
- しきい値： なし

2 番目のウォッチは、以下のパラメータから構成されます。

- 名前： CPU_Load
- データ タイプ： 実数
- 式
  - 式： CPULoadRate
  - インスタンス： なし
- プロパティ
  - デフォルトでアクティブにする： オン
  - 評価： ポーリングにより評価
  - ポーリング間隔： 1 分
- しきい値
  - 値が次の場合はしきい値違反： > WatchLoad_v
  - 値が次の場合にしきい値をリセット： < = WatchLoad_v
- アラーム
  - アラーム重大度： マイナー
  - アラーム説明： WatchLoad_Alarm
  - スクリプト： なし

使用効率およびテスト用のウォッチ シナリオ 2

ここで説明するウォッチの例は、使用効率およびテスト用です。

この例では、コンテナ モデルの「複合状態」属性が 4 を超過したときにマイナー アラームを生成する EoC ウォッチを作成します。 まず conditionCheck_ref という参照属性を作成し、それを ConditionCheck というウォッチの属性で使用します。 次に、［アラーム説明］ダイアログ ボックスを使用して、「ConditionCheckAlarm」（または別の適切な文字列）という名前のアラーム説明を作成します。

参照属性には以下のパラメータがあります。

- 名前： ConditionCheck_ref
- データ タイプ： 整数
- 式
  - 式： 4
  - インスタンス： なし
- プロパティ： なし
- しきい値： なし

例: 使用効率およびテスト用のウォッチ

ウォッチは、以下のパラメータで構成されています。

- 名前： ConditionCheck
- データ タイプ： 整数
- 式
  - 式： Composite_Condition
  - インスタンス： なし
- プロパティ
  - デフォルトでアクティブにする： オン
  - 評価： 変更時に評価

- しきい値
  - 値が次の場合はしきい値違反： > ConditionCheck_ref
  - 値が次の場合にしきい値をリセット： <= ConditionCheck_ref
- アラーム：
  - アラーム重大度： マイナー
  - アラーム説明： ConditionCheckAlarm
- スクリプト： なし

使用効率およびテスト用のウォッチ シナリオ 3

ここで説明するウォッチの例は、使用効率およびテスト用です。

この例では、負荷の値が 90 分を超えて 10% 未満であった場合に、アラームを作成します。 このウォッチを設定するには、必要な外部属性を使用するようにウォッチ式を変更し、しきい値を連続して実行されるポーリング回数に設定します。このポーリング回数を求めるには、対象のポーリング期間をポーリング間隔で割り算します。

例: 使用効率およびテスト用のウォッチ

最初のウォッチは、以下のパラメータから構成されます。

- 名前： Watch_Load_Under_10_Pct
- データ タイプ： 整数
- 式
  - 式： MAX (0, MIN (1, (<負荷が 10% 未満のときのテスト式を入力>)))
  - インスタンス： なし
- プロパティ
  - デフォルトでアクティブにする： オフ
  - 評価： オンデマンドで評価
- しきい値： なし

2 番目のウォッチは、以下のパラメータから構成されます。

- 名前： Watch_TimeTicker_LoadUnder10Pct
- データ タイプ： 整数
- 式
  - 式： (Watch_TimeTicker_LoadUnder10Pct + 1) * Watch_Load_Under_10_Pct
  - インスタンス： なし
- プロパティ
  - デフォルトでアクティブにする： オフ
  - 評価： オンデマンドで評価
- しきい値： 連続して実行される必要なポーリング回数

使用率およびテスト用のウォッチ シナリオ 4

ここで説明するウォッチの例は、使用効率およびテスト用です。

1 番目のウォッチは、CPU の参照使用率（この例では 80%）が維持されているかどうかを監視します。

2 番目のウォッチは、CPU 使用率が長期間特定のレベル（80%）であるときにアラームをトリガします。 このウォッチでは、しきい値（3）にポーリング間隔（5 分）を乗算してこの期間を計算します。したがって、CPU 使用率が 15 分間 80% を越えると、このウォッチはしきい値違反となり、アラームがトリガされます。これらの値は、要件に合わせて調整できます。

例: 使用効率およびテスト用のウォッチ

最初のウォッチは、以下のパラメータから構成されます。

- 名前： CPU_Duration_Over_80
- データ タイプ： 整数
- 式
  - 式： MAX(0, MIN(1, INTEGER((INTEGER(cpqHoCpuUtilMin.#) >= 80))))
  - インスタンス： すべて

- プロパティ
  - デフォルトでアクティブにする： オン
  - 評価： オンデマンドで評価
  - 継承可能： オフ
- しきい値： なし

2 番目のウォッチは、以下のパラメータから構成されます。

- 名前： CPU_Time_Duration
- データ タイプ： 整数
- 式
  - 式： ((CPU_Time_Duration.# + 1) * CPU_Duration_Over_80.#)
  - インスタンス： すべて
- プロパティ
  - デフォルトでアクティブにする： オン
  - 評価： ポーリングにより評価
  - ポーリング間隔： 0 + 00:05:00
  - 継承可能： オフ
- しきい値
  - 値が次の場合はしきい値違反： >= 3
  - 値が次の場合にしきい値をリセット： < 3
- アラーム
  - アラーム重大度： マイナー
  - アラーム説明： ErrorTholdAlarm
  - ユーザがクリア可能なアラーム
  - ユーザがアラームをクリアした場合、ウォッチをリセットしない
  - スクリプト： なし