PhoenixDS

ПРОГРАММНАЯ ПЛАТФОРМА

Руководство администратора

Программный модуль

PhoenixDS CmpEvnt

Аннотация

Программный модуль "PhoenixDS CmpEvnt" (далее – ПМ PhoenixDS CmpEvnt), расширяющий базовый функционал Программной платформы "PhoenixDS" (далее – ПП PhoenixDS), позволяет настраивать сценарии дополнительной обработки данных, полученных от различных источников, с применением нестандартной логики. Может использоваться для агрегирования данных, а также для обработки сторонних данных при интеграции систем.

В настоящем Руководстве администратора приведен порядок настройки объектов, посредством которых реализуется нестандартная обработка данных.

Перед изучением настоящего документа рекомендуется освоить основные принципы функционирования ПП PhoenixDS, изложенные в базовом документе "ПП PhoenixDS. Руководство администратора".

ДИНАМИЧЕСКИЕ СИСТЕМЫ

Россия, 101066, Москва, ул. Бауманская, д.53
тел.: +7 (495) 228-1100
www.dynasystems.ru
info@dynasystems.ru

Служба технической поддержки:
support@dynasystems.ru

Содержание

1. ВВЕДЕНИЕ

1.1. Назначение ПМ PhoenixDS CmpEvnt

ПМ **PhoenixDS CmpEvnt** позволяет настраивать сценарии дополнительной обработки данных, полученных от различных источников, с применением нестандартной логики. Может использоваться для агрегирования данных, а также для обработки сторонних данных при интеграции систем.

2. НАСТРОЙКА И ИСПОЛЬЗОВАНИЕ ПМ PHOENIXDS CMPEVNT

2.1. Устройство комплексной логики (Devices::Analysis::Logix)

Устройство комплексной логики (**Complex Logic**) предназначено для дополнительной обработки данных, полученных от различных источников, с применением нестандартной логики. Может использоваться для агрегирования данных, а также для обработки сторонних данных при интеграции систем.

Устройство комплексной логики состоит из одного целевого объекта с произвольным набором источников данных, настроенных администратором. Значение каждого источника данных вычисляется на основании кода, введенного администратором при создании устройства. Поэтому для конфигурирования устройства комплексной логики администратор должен обладать базовым уровнем знания языка Perl или Си-подобных языков.

Для получения данных с устройств комплексной логики в ПК **PhoenixDS Monitoring** необходимо запустить системный процесс `kollector-logix`.

Параметры шаблона объекта Devices::Analysis::Logix

Отображаемое имя | `display-name`

Имя объекта, отображаемое в дереве конфигурации. Должно быть уникальным в пределах одного уровня конфигурации (например, в рамках одной папки).

Описание | `long_desc`

Развернутое описание объекта. Может содержать HTML-теги и пробелы.

Логические выражения

Программный код на языке Perl для вычисления значений источников данных.

Создание устройства

Общий порядок действий для создания нового устройства комплексной логики:

1. в дереве конфигурации поместить курсор на нужную папку, в контекстном меню выбрать пункт **Добавить устройство** и в появившемся списке выбрать **Complex Logic**;
2. ввести параметры устройства **Отображаемое имя** и **Описание**;
3. нажать кнопку  под областью **Логические выражения**, в появившемся окне ввести **Имя логического выражения** и нажать кнопку **OK**; в результате будет создана вкладка с введенным именем;
4. на созданной вкладке ввести код на языке Perl для вычисления значения источника данных. Подробности см. ниже.
5. если для устройства комплексной логики требуется настроить несколько источников данных, то повторить п.п.2-3,
6. нажать кнопку **Сохранить**.

Код для вычисления значения источника данных

При разработке кода на вкладках удобно пользоваться возможностями вставки фрагментов кода. Для этого сначала в текстовом поле вкладки поместить курсор в позицию для вставки фрагмента кода, а затем воспользоваться одной из сервисных возможностей:

- вставка функции – нажать кнопку  и выбрать из списка функцию:
  - ds() – значения (значение) источников данных, существующих в дереве конфигурации;
  - alert() – значение/статус сигнала или множество значений/статусов сигналов;
  - min() – минимальное значение;
  - max() – возвращает максимальное значение;.
  - avg() – возвращает среднее арифметическое значение;
  - worst() – возвращает наихудший статус сигнала;

best() – возвращает наилучший статус сигнала;

havail() – агрегированный статус сигнала.

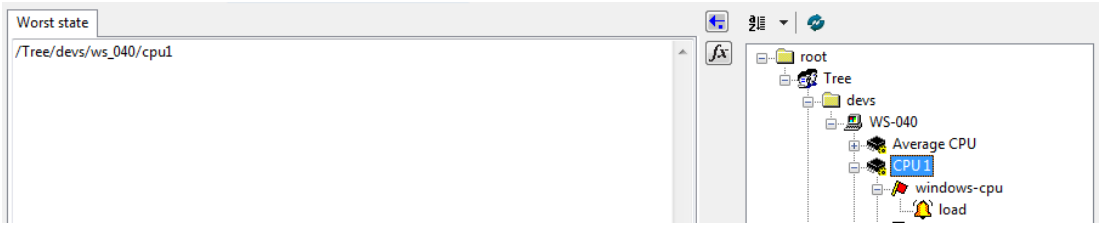
В результате в текстовое поле будет вставлена заготовка кода для вызова функции. Останется отредактировать ее аргументы.

- вставка пути на объект дерева конфигурации или функций `alert()` и `ds()` с заданным аргументом:

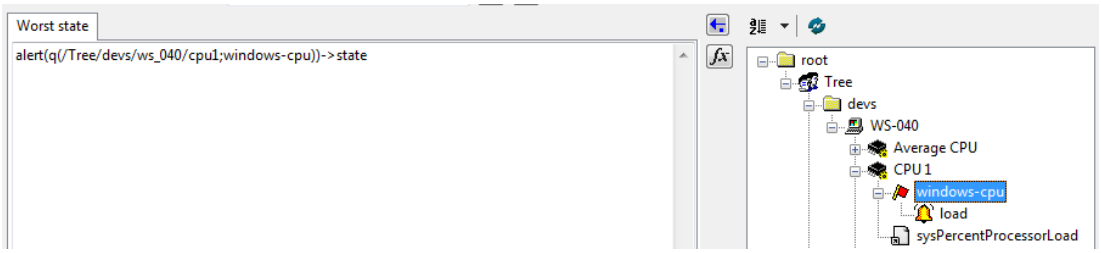
  1. в дереве конфигурации (справа от вкладки с текстом кода) поместить курсор на нужный узел (в дереве отображаются, в том числе, источники данных целевых объектов);
  2. нажать кнопку  . В результате, в зависимости от выбранного узла, в текстовое поле на вкладке добавляется путь или функция с заданным аргументом.

     Примеры.

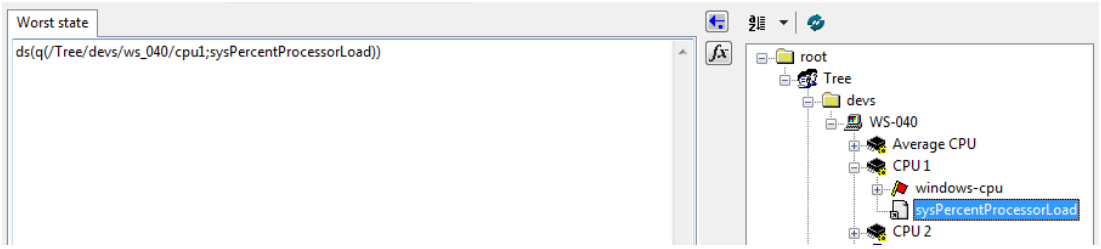
     Если в дереве конфигурации выбрано устройство, то в текстовое поле вставляется путь к объекту:



     Если в дереве конфигурации выбран набор сигналов, то в текстовое поле вставляется код вызова функции `alert` с указанием в качестве аргумента выбранного набора сигналов:



     Если в дереве конфигурации выбран источник данных, то в текстовое поле вставляется код вызова функции `ds` с указанием в качестве аргумента выбранного источника данных:



Удаление источника данных с устройства комплексной логики

Для удаления источника данных с устройства следует:

1. в окне редактирования параметров устройства перейти на вкладку с названием источника данных, который нужно удалить;
2. нажать кнопку  под областью **Логические выражения**;
3. в появившемся окне для подтверждения действия нажать кнопку **Да**.

2.2. Функции

В настоящем разделе описаны функции, которые могут использоваться в коде вычисления значения источника данных для устройства комплексной логики.

2.2.1. Захват значения источника данных (ds)

Функция `ds` возвращает поступившие по шине ESB от коллекторов значения (или одно значение) от источников данных. Аргументом функции является источник данных или поддерево.

Синтаксис

```
ds(q(<path_to_ds>))
```

`<path_to_ds>::=<Путь к таргету>[;<Имя источника данных>]`

`<Путь к таргету>` – существующий путь в дереве конфигурации (как в `/Devices`, так и в `/Tree` и т.п.) к целевого объекта (например: `/Tree/SelfMon/localhost/cpu`) или поддереву (например: `/Tree/devs/ws_040`). В последнем случае результат будет векторным – множество значений всех источников данных в поддереве.

`<Имя источника данных>` – имя источника данных (например: `sysPercentProcessorLoad`). Необязательный аргумент. Если задано имя источника, а путь задан для поддерева, то результатом функции будет множество значений всех источников данных с таким именем, имеющихся в поддереве. Если имя источника данных не задано, то результатом функции будет множество значений всех источников в целевом объекте (или поддереве).

Пример

Захват значений загрузки `sysPercentProcessorLoad` по каждому процессору устройства `/Tree/devs/ws_040` (результат – множество значений):

```
ds(q(/Tree/devs/ws_040;sysPercentProcessorLoad))
```

2.2.2. Захват значения/статуса сигнала (alert)

В зависимости от заданного аргумента функция `alert` возвращает значение или статус одного сигнала либо множество значений или статусов сигналов. Под статусами сигналов подразумеваются соответствующие им числовые константы: 0 – синий, 1 – зеленый, 2 – желтый, 3 – красный, 4 – бордовый, 5 – серый. Аргументом функции является путь к сигналу конкретного целевого объекта или ветви дерева конфигурации.

Синтаксис

```
alert(q(<path_to_alert>))->{state|value}
```

`<path_to_alert>::=<Путь к таргету>[;<Имя шаблона сигналов>];[<Имя сигнала>]`

`<Путь к таргету>` – любой существующий путь в дереве конфигурации (только в `/Tree`) к целевому объекту или поддереву. См. также аналогичный атрибут для функции `ds`.

`<Имя шаблона сигналов>` – имя набора сигналов, например: `lnx-cpu`; будут рассматриваться только целевые объекты, которым назначен данный набор сигналов. Необязательный атрибут. Если он не задан, то используется любой обнаруженный набор сигналов.

`<Имя сигнала>` – имя сигнала, например: `load`; будет рассматриваться только данный сигнал. Необязательный атрибут. Если он не задан, то используются все сигналы (результат будет векторным).

`{state|value}`

Тип возвращаемого значения:

`state` – статус сигнала или множество статусов сигналов;

`value` – значение сигнала или множество значений сигналов.

Примеры

Получение статусов сигналов `load` по всем процессорам устройства `ws_040`:

```
alert(q(/Tree/devs/ws_040;load))->state
```

Результат выполнения функции – множество числовых констант, соответствующих статусам сигналов.

Получение значений всех сигналов из набора `lnx-storage`, привязанного к целевым объектам устройства `localhost`:

```
alert(q(/Tree/SelfMon/localhost;lnx-storage;))->value
```

Получение значений всех сигналов всех наборов сигналов в дереве `/Tree`:

```
alert(q(/Tree;))->value
```

2.2.3. Определение минимального, максимального и среднего значения (min, max, avg)

Функция `min` возвращает минимальное значение, функция `max` – максимальное значение, функция `avg` – среднее арифметическое значение. Аргументами функций могут быть любые числовые значения, в частности, значения источников данных, полученные (захваченные) из шины ESB.

Функции имеют одинаковый синтаксис. Рассмотрим его на примере функции `min`.

Синтаксис

```
min(<множество чисел>)
```

`<множество чисел>` – последовательность чисел и/или функций, возвращающих числовые значения. Аргументы перечисляются через запятую. В качестве аргументов можно использовать результат функций `ds` и `alert`, а также функций `min`, `max`, `avg`.

Примеры

Минимальное значение среди всех сигналов `load` в поддереве `/Tree/devs`:

```
min(alert(q(/Tree/devs;load))->value)
```

Средняя загрузка двухъядерного процессора на компьютере `/Tree/devs/ws_040` вычисляется как среднее арифметическое значений источников данных, показывающих загрузку ядра процессора:

```
avg(
  ds(q(/Tree/devs/ws_040/cpu1;sysPercentProcessorLoad)),
  ds(q(/Tree/devs/ws_040/cpu2;sysPercentProcessorLoad))
   )
```

2.2.4. Определение наихудшего и наилучшего статуса сигнала (worst и best)

Функция `worst` возвращает наихудший статус сигнала, функция `best` – наилучший статус сигнала. Аргументами функции должны быть числовые константы, соответствующие статусам сигналов. Результирующим значением функции также является числовая константа. Для определения наихудшего / наилучшего статуса функция использует следующий порядок сортировки (от лучшего к худшему):

(1 🟢) зеленый – наилучший статус;

(2 🟡) желтый;

(0 🔵) синий;

(5 ⚫) серый;

(3 🔴) красный;

(4 🟤) бордовый – наихудший статус.

Примечание. Порядок сортировки статусов сигналов задается параметром `states_priorities` в файле `usr/local/lancelot/conf/svgd.conf`.

Функции имеют одинаковый синтаксис. Рассмотрим его на примере функции `worst`.

Синтаксис

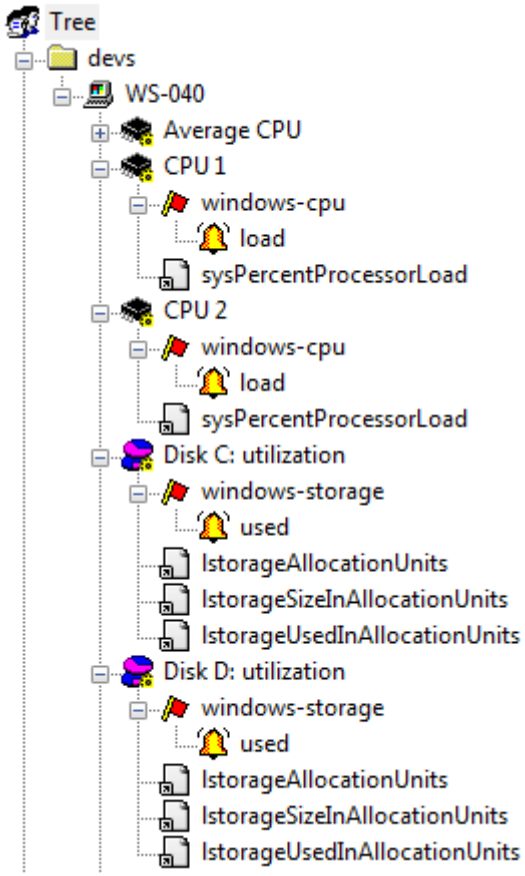
```
worst(<множество статусов>)
```

`<множество статусов>` – последовательность числовых констант, соответствующих статусам сигналов. Аргументы перечисляются через запятую. Можно использовать функции `alert`, возвращающие статус сигнала или множество статусов:

```
alert(q(<path_to_alert>))->state
```

Примеры

Проиллюстрируем применение функций на примере устройства `WS-040`.



Для выявления наиболее загруженного процессора определяется худший статус из всех сигналов `load` на устройстве `WS-040`:

```
worst(alert(q(/Tree/devs/ws_040;load))->state)
```

Для выявления диска с наибольшим свободным пространством определяется лучший статус среди сигналов `used` на дисках C и D:

```
best(
     alert(q(/Tree/devs/ws_040/diskc;windows-storage;used))->state
     alert(q(/Tree/devs/ws_040/diskd;windows-storage;used))->state
     )
```

2.2.5. Определение агрегированного статуса сигнала (havail)

Функция `havail` возвращает агрегированный статус сигнала (*havail* – от *high availabillity*). Аргументом функции является множество статусов сигналов. Функция `havail` вычисляет статус по алгоритму:

(3 🔴) красный, если в аргументе все статусы не зелёные;

(2 🟡) жёлтый, если в аргументе хотя бы один статус зелёный и имеются не зелёные статусы;

(1 🟢) зелёный, если в аргументе все статусы зелёные.

Алгоритм проиллюстрирован на примерах в таблице:

Аргумент (статусы сигналов)	Значение функции `havail()`
0 🔵, 2 🟡, 3 🔴, 4 🟣, 5 ⚫	3 🔴
1 🟢, 2 🟡, 3 🔴, 4 🟣, 5 ⚫	2 🟡
1 🟢, 1 🟢, 1 🟢, 1 🟢, 1 🟢	1 🟢

Синтаксис

```
havail(<множество статусов>)
```

`<множество статусов>` – последовательность числовых констант, соответствующих статусам сигналов. Аргументы перечисляются через запятую. Можно использовать функции `alert`, возвращающие статус сигнала или множество статусов:

```
alert(q(<path_to_alert>))->state
```

2.3. Примеры устройств комплексной логики

В настоящем разделе рассмотрены примеры практического применения устройств комплексной логики, а также приведен код для вычисления их источников данных.

2.3.1. Температура в серверных комнатах

Датчики температуры, расположенные в двух серверных комнатах, размещены в дереве конфигурации в папках `/Tree/ServerRoom1` и `/Tree/ServerRoom2` соответственно (никакие другие устройства в этих папках не сконфигурированы). Требуется контролировать температуру в серверных помещениях. Для этого в устройстве комплексной логики можно настроить источник данных: если хотя бы в одной комнате температура превышает 20 градусов, то источник данных возвращает значение 0, в противном случае – 1.

Код для вычисления значения источника данных:

```
max(ds(q(/Tree/ServerRoom1),ds(q(/Tree/ServerRoom1))>20?0:1
```

2.3.2. Защита от ложных срабатываний сигналов

Для исключения ложных срабатываний сигналов при однократных сбоях, не имеющих фатальных последствий, можно настроить устройство комплексной логики, у которого истоник данных возвращает среднее значение требуемого показателя. Для расчета среднего значения используются последние X значений, полученных от коллектора и сохраненных в кэш. Таким образом, график показателя будет сглажен, и единичные скачки значений не будут заметны.

Рассмотрим в качестве исследуемого показателя среднюю загрузку двух ядер процессора. Для него в устройстве комплексной логики можно настроить источник данных, значение которого вычисляется следующим образом:

- в кэш формируется массив из 30 последних значений исследуемого показателя, т.е. средней загрузки двух ядер процессора (при каждом проходе коллектора этот массив обновляется);
- значение источника данных вычисляется как среднее от значений, записанных в массив.

Код для вычисления значения источника данных:

```
our @history;
push @history,
  avg(
    ds( q(/Tree/devs/ws_040/cpu1) ),
    ds( q(/Tree/devs/ws_040/cpu2) )
  );

my $val = avg( @history );
```

```
@history = @history[ 1 .. 29 ] if @history >= 30;

$val;
```