Класс SystemAlarming

Редакция от 13.10.2026 · docs.kb-agava.ru

Содержание

1 Унаследованный интерфейс 2
2 Создание объекта 2
3 Методы 3
  3.1 AcknowledgeAll() 3
  3.2 GetAlarmList() 3
  3.3 GetActiveNodeAlarms() 3
4 События 4
  4.1 Подписка и отписка 4
5 Тип AlarmList 6
6 Тип EventInfo 6

Класс `SystemAlarming` представляет собой узел типа `Система тревог`.

Класс доступен начиная с версии `AS 1.6.34+`.

Предок: BasicNode.

Описанные ниже события, методы доступа к списку тревог и типы `AlarmList`/`EventInfo` добавлены в ветке `AS 1.7.8+`. Указание `AS 1.6.34+` относится к самому классу и не означает наличия всего расширенного интерфейса в старых рантаймах.

1 Унаследованный интерфейс

`SystemAlarming` предоставляет следующие методы и свойства:

```
string GetName()
string GetShortName()
string GetDescription()
string GetPath()
void SetName(string name)
void SetShortName(string shortname)
void SetDescription(string description)
NodeProperty@ GetProperty(string name)
bool PropertyValueUpdated(string name)
Result Start()
Result Stop()
EventProperty@ OnPropertyUpdated
```

`Start()` и `Stop()` вызывают соответствующие нативные методы и возвращают `Result`. Обычно жизненным циклом узлов управляет рантайм.

`OnPropertyUpdated` — отдельное событие изменения свойства узла, не событие списка тревог. Его обработчик имеет сигнатуру `void EventPropertyCB(NodeProperty@ prop)`.

`PropertyValueUpdated()` возвращает `bool`, как и метод нативного `_t`. Подробнее об общих методах — в описании BasicNode.

2 Создание объекта

```
SystemAlarming()
SystemAlarming(const string &in path)
SystemAlarming(const SystemAlarming &in other)
SystemAlarming(SystemAlarming_t &in source)
```

Конструктор с `path` связывает прокси с существующим узлом проекта. Например:

```
SystemAlarming@ sysal = SystemAlarming("/Project/Station/SystemAlarming");
```

Путь необходимо заменить на путь к узлу своего проекта. Если узел отсутствует или имеет другой тип, возникает исключение скрипта. Конструктор без пути создаёт скриптовый объект и не ищет существующую систему тревог проекта.

Копия прокси использует тот же нативный объект. Для прокси, связанного с узлом проекта, операции со списком тревог, квитирование и подписки на события относятся к исходному узлу, а не к отдельной копии его списка.

3 Методы

3.1 AcknowledgeAll()

```
void AcknowledgeAll()
```

Квитирует все записи текущего списка тревог, включая сохранённые неактивные записи. Окно тревог не обязано быть открыто. Результат — `void`.

```
void AcknowledgeAlarms()
{
    SystemAlarming@ sysal = SystemAlarming("/Project/Station/SystemAlarming");
    sysal.AcknowledgeAll();
}
```

3.2 GetAlarmList()

```
AlarmList@ GetAlarmList()
```

Возвращает живое представление списка тревог исходной системы. Это не массив записей и не копия списка. Для чтения записей используется `GetAlarmsSnapshot()`, для изменения квитирования — `SetAcknowledged()`. Представление удерживает узел-владелец, пока используется возвращённый `AlarmList@`.

```
void AcknowledgeFirstAlarm(SystemAlarming@ sysal)
{
    AlarmList@ alarms = sysal.GetAlarmList();
    array<EventInfo>@ snapshot = alarms.GetAlarmsSnapshot();

    if (snapshot.length() != 0)
    {
        alarms.SetAcknowledged(snapshot[0].m_infoId, true);
    }
}
```

3.3 GetActiveNodeAlarms()

```
array<BasicNode_t@>@ GetActiveNodeAlarms(BasicNode_t@ node)
```

Возвращает массив ссылок на **действия**, зарегистрированные системой как источники активных тревог для заданного узла. Это не массив `EventInfo`: для получения записей тревог следует использовать `GetAlarmList().GetAlarmsSnapshot()`.

Параметр `node` — нативная ссылка на узел. Прокси `BasicNode` и его потомки допускают неявное приведение к `BasicNode_t@`. Для прокси узла проекта используется исходный узел. Если совпадений нет, возвращается пустой массив. Неразрешённая ссылка в массиве может быть `null`.

У нативного `SystemAlarming_t` параметр зарегистрирован как `BasicNode_t@+ node`; модификатор `+` относится к управлению временем жизни аргумента и не меняет способ вызова метода из скрипта.

```
array<string>@ GetAlarmActionNames(SystemAlarming@ sysal, BasicNode@ node)
{
    array<string>@ names = array<string>();
    array<BasicNode_t@>@ actions = sysal.GetActiveNodeAlarms(node);

    for (uint i = 0; i < actions.length(); ++i)
    {
        if (actions[i] !is null)
        {
            names.insertLast(actions[i].GetName());
        }

    }

    return names;
}
```

4 События

```
EventVoid@ OnActiveEventsCleared
EventVoid@ OnActiveEventsAcknowledged
```

События — свойства только для чтения. Они возвращают объекты `EventVoid` с обработчиком следующей сигнатуры:

```
void EventVoidCB()
```

Событие	Условие вызова
OnActiveEventsCleared	Ранее в списке были активные тревоги, а после изменения списка не осталось ни одной записи, для которой `EventInfo.IsActive()` возвращает `true`.
OnActiveEventsAcknowledged	Ранее в списке были неквитированные записи, а после изменения списка не осталось ни одной записи с `m_bAcknowledged == false`. Проверяется весь текущий список, включая сохранённые неактивные записи.

События отражают **переход состояния всего списка**, а не каждое отдельное изменение тревоги. Квитирование одной из нескольких неквитированных записей не вызывает `OnActiveEventsAcknowledged`. Повторное квитирование уже квитированных записей также не вызывает новое событие. При исходно пустом списке события не генерируются только из-за подписки.

Тревога с требованием квитирования может оставаться активной по `IsActive()` после деактивации, пока не будет квитирована. Удаление записей из списка также учитывается при определении переходов состояния.

Обработчики вызываются при обработке изменений списка запущенной системой тревог. Настроенные в проекте действия для соответствующего события выполняются перед скриптовыми обработчиками.

4.1 Подписка и отписка

Поддерживаются оба способа:

```
void EventVoid::AddListener(EventVoidCB@ fn)
void EventVoid::RemoveListener(EventVoidCB@ fn)

event += handler;
event -= handler;
```

Оба способа можно комбинировать: подписаться через `AddListener()`, а отписаться через `-=`, или наоборот.

```
SystemAlarming@ monitoredAlarms = SystemAlarming("/Project/Station/SystemAlarming");

void AllActiveAlarmsCleared()
{
    // Обработка перехода к отсутствию активных тревог.
}

void AllAlarmsAcknowledged()
{
    // Обработка перехода к отсутствию неквитированных записей.
}

void SubscribeToAlarms()
{
    monitoredAlarms.OnActiveEventsCleared.AddListener(AllActiveAlarmsCleared);
    monitoredAlarms.OnActiveEventsAcknowledged += AllAlarmsAcknowledged;
}

void UnsubscribeFromAlarms()
{
    monitoredAlarms.OnActiveEventsCleared.RemoveListener(AllActiveAlarmsCleared);
    monitoredAlarms.OnActiveEventsAcknowledged -= AllAlarmsAcknowledged;
}
```

Подписываться следует один раз, например при инициализации приложения, а не при каждом цикле задачи. Повторная подписка добавляет ещё один вызов обработчика; одна отписка удаляет одну соответствующую подписку.

Для метода скриптового объекта используется делегат `EventVoidCB`:

```
class AlarmObserver
{
    void OnCleared() {}
}

void AttachObserver(SystemAlarming@ sysal, AlarmObserver@ observer)
{
    sysal.OnActiveEventsCleared += EventVoidCB(observer.OnCleared);
}

void DetachObserver(SystemAlarming@ sysal, AlarmObserver@ observer)
{
    sysal.OnActiveEventsCleared -= EventVoidCB(observer.OnCleared);
}
```

При отписке делегат сопоставляется по объекту и методу. `EventVoid` также предоставляет `Count()`, `RemoveAllListeners()` и `Clear()`. Последние два метода удаляют **все** подписки на данном событии, включая подписки других прокси того же узла.

5 Тип AlarmList

Объекты типа `AlarmList` получаются через вызов метода `SystemAlarming.GetAlarmList()` .

Метод	Описание
`bool IsEmpty() const`	Проверяет отсутствие любых записей в списке, а не только активных тревог.
`array<EventInfo>@ GetAlarmsSnapshot() const`	Возвращает независимый снимок всех текущих записей. После изменения списка необходимо получить новый снимок.
`void SetAcknowledged(uint id, bool state)`	Устанавливает квитирование записи: `true` — квитирована, `false` — не квитирована. `id` — значение `EventInfo.m_infoId`, не идентификатор узла. Отсутствующий идентификатор не изменяет список.
`void SetRemoveInactiveAlarms(bool state)`	Включает или выключает удаление неактивных записей. Включение сразу обрабатывает уже неактивные записи; дальнейшее удаление выполняется при изменениях списка. Активность определяется методом `IsActive()`, в том числе с учётом требования квитирования.

Изменение квитирования и удаление записей через `AlarmList` участвуют в обычном механизме уведомлений системы тревог.

6 Тип EventInfo

`EventInfo` — запись снимка, возвращаемого `AlarmList.GetAlarmsSnapshot()`. Поддерживаются создание, копирование и присваивание значения `EventInfo`.

Поля и свойство `Node` доступны только для чтения:

Поле/свойство	Тип	Описание
`m_infoId`	`uint`	Идентификатор записи для `AlarmList.SetAcknowledged()`.
`m_strMessage`	`string`	Текст тревоги.
`m_strUser`	`string`	Пользователь, связанный с записью события.
`m_bAcknowledged`	`bool`	Признак квитирования на момент создания снимка.
`m_bRequiresAcknowledgement`	`bool`	Требование квитирования для завершения активности деактивированной тревоги.
`Node`	`BasicNode_t@`	Ссылка на узел, сгенерировавший событие. Может быть `null`, если ссылка не разрешается.

Методы записи:

```
bool IsActive() const
string GetModeString() const
string GetGroupString() const
string GetEventLevel() const
string GetEventType() const
```

`IsActive()` учитывает состояние и требование квитирования. Остальные методы возвращают текстовые представления состояния, группы, уровня и типа события.

Запись снимка сама по себе не изменяет список тревог. Для квитирования следует вызвать `AlarmList.SetAcknowledged()` или `SystemAlarming.AcknowledgeAll()`.