Создание расширения, включающего поддержку нового протокола в СКАУТ-Платформе

Введение

С помощью расширения разработчик может включить в состав СКАУТ-Платформы следующие новые элементы системы:

1. Тип устройств (терминалов).
2. Тип источников данных.
3. Протокол обмена данными.

Тип устройств (терминалов)

Все создаваемые в системе терминалы имеют какой-либо тип устройства (терминалы МТ-500, МТ-600 и т.д.). Добавление нового типа устройства может потребоваться в том случае, если устройство должно иметь какую-либо специфическую конфигурацию по умолчанию (базовый набор датчиков и т.д.). В рамках демонстрационного расширения в СКАУТ-Платформу добавляется новый тип устройств «**Demo Device**».

Для добавления нового устройства необходимо на сервере приложений, в терминальном шлюзе, в СКАУТ-Менеджере, СКАУТ-Студио и СКАУТ-Сильверстудио зарегистрировать описатель данного устройства (то, что будет видеть конечный пользователь, и что будет отображаться в логах).

Тип источников данных

Тип источников данных определяет, как следует из названия, откуда и как в систему будут поступать сообщения от устройств. На данный момент доступен единственный источник данных «**TCP»**. В демонстрационном расширении добавляется новый тип источников данных «**Demo Device File**».

Для добавления нового типа источника данных:

- на сервере приложений и в СКАУТ-Менеджере зарегистрировать

описатель источника данных (аналогично описателю типа устройства);
- на сервере приложений, в терминальном шлюзе и в СКАУТ-Менеджере зарегистрировать фабрику сериализаторов/десериализаторов для передачи параметров конфигурации между приложениями;
- в СКАУТ-Менеджере зарегистрировать редактор конфигурации, который будет предоставлять конфигурацию по умолчанию, а так же визуальное представление этой конфигурации для редактирования пользователем.

Протокол обмена данными.

Протокол обмена данными определяет, каким образом необходимо преобразовывать данные, поступающие из источника, чтобы с этими данными могла работать СКАУТ-Платформа. В демонстрационном расширении добавляется новый протокол «**Demo Device Protocol**».

Для добавления нового протокола на сервере приложений, в терминальном шлюзе и в СКАУТ-Менеджере необходимо зарегистрировать описатель протокола (аналогично описателю типа устройства), а также в терминальном шлюзе – фабрику обработчиков портов, работающих с указанным протоколом.

В общем случае, реализация протоколов никак не связана с реализациями используемых источников данных, и любой протокол может работать с любым источником данных.

Добавление нового типа устройств, источников данных, протоколов с позиции реализации в программном обеспечении

Для добавления каждого из этих элементов необходимо произвести регистрацию определенных объектов системы в плагинах расширения.

1. Добавление нового типа устройств (DeviceType).

Для добавления нового типа устройств необходимо произвести следующие действия:

1.1. Регистрация экземпляра класса, реализующего `IDeviceTypeDescriptor` (в контроллере `DeviceTypeDescriptionController`), в плагинах для:

- сервера приложений;
- терминального шлюза;
- СКАУТ-Менеджера;
- СКАУТ-Студио;
- СКАУТ-Сильверстудио.

2. Добавление нового типа источника данных (DataProvider).

Для добавления нового типа источника данных:

2.1. Регистрация экземпляра класса, реализующего `IDeviceDataProviderDescriptor` (в контроллере `DeviceDataProviderDescriptionController`), в плагинах для:

- сервера приложений;

- СКАУТ-Менеджера.

2.2. Регистрация экземпляра класса, реализующего `IDeviceDataProviderSerializer` (в контроллере `DeviceDataProviderSerializationController`), в плагинах для:

- сервера приложений;
- терминального шлюза;
- СКАУТ-Менеджера.

2.3. Регистрация экземпляра класса, реализующего `IDeviceDataProviderEditor` (в контроллере `DeviceDataProviderViewController`), в плагине для:

- СКАУТ-Менеджера.

2.4. Регистрация экземпляра класса, реализующего `IDeviceDataProvider` (в контроллере `DeviceDataProviderController`), в плагине для:

- терминального шлюза.

3. Добавление нового протокола (DataProtocol).

Для добавления нового протокола необходимо произвести следующие действия:

3.1. Регистрация экземпляра класса, реализующего `IDeviceDataProtocolDescriptor` (в контроллере `DeviceDataProtocolDescriptionController`), в плагинах для:

- сервера приложений;
- терминального шлюза;
- СКАУТ-Менеджера.

3.2. Регистрация экземпляра класса, реализующего `IDeviceDataProtocol` (в контроллере `DeviceDataProtocolController`), в плагине для:

- терминального шлюза.

Регистрация и разрегистрация всех объектов, касающихся добавляемого элемента системы, должна производиться с использованием одного и того же GUID.

Подробнее о реализуемых интерфейсах написано в пункте «**Реализация интерфейсов**».

Демонстрационное расширение

Как уже упоминалось ранее, предоставляемое демонстрационное расширение добавляет в СКАУТ-Платформу новый тип устройств «**Demo Device**», новый тип источника данных «**Demo Device File**» и новый протокол «**Demo Device Protocol**». С точки зрения пользователя в СКАУТ-Менеджере появляется возможность создавать терминалы с типом «**Demo Device**», порты с источником данных «**Demo Device File**» и протоколом «**Demo Device Protocol**». Источник «**Demo Device File**» реализует чтение данных из локального файла, указываемого при создании или редактирования свойств порта. Протокол «**Demo Device Protocol**» преобразует данные, полученные из файла, в формат, используемый в системе СКАУТ.

Далее по порядку описано, что и в каких плагинах регистрируется (**Требуемые плагины расширения**), регистраторы объектов (**Регистраторы объектов в СКАУТ-Платформе**), реализуемые интерфейсы (**Реализация интерфейсов**).

Требуемые плагины расширения

Ниже на основе плагинов из демо-расширения описаны действия, которые необходимо произвести для каждого элемента СКАУТ-Платформы.

1. Плагин для сервера приложений

`Scout.Plugins.SpfTestProtocolPlugin.Server.AppServer.DemoAppServerPlugin`.

В методе `Plug()` регистрируются:

1.1. Описание типа устройства (`DemoDeviceTypeDescriptor`).

1.2. Описание источника данных (`DemoDataProviderDescriptor`).

1.3. Описание протокола (`DemoDeviceProtocolDescriptor`).

1.4. Сериализатор конфигурации (`DemoDeviceDataProviderSerializer`).

В методе `Unplug()` производится разрегистрация соответствующих объектов.

2. Плагин для терминального шлюза

`Scout.Plugins.SpfTestProtocolPlugin.Server.Gateway.DemoGatewayPlugin`.

В методе `Plug()` регистрируются:

2.1. Описание типа устройства (`DemoDeviceTypeDescriptor`).

2.2. Описание протокола (`DemoDeviceProtocolDescriptor`).

2.3. Сериализатор конфигурации (`DemoDeviceDataProviderSerializer`).

2.4. Источник данных (`DemoDataProvider`).

2.5. Протокол (`DemoDataProtocol`).

В методе `Unplug()` производится разрегистрация соответствующих объектов.

3. Плагин для СКАУТ-Менеджера

`Scout.Plugins.SpfTestProtocolPlugin.Client.Manager.DemoManagerPlugin`.

В методе `Plug()` регистрируются:

3.1. Описатель типа устройства (`DemoDeviceTypeDescriptor`).

3.2. Описатель источника данных (`DemoDataProviderDescriptor`).

3.3. Описатель протокола (`DemoDeviceProtocolDescriptor`).

3.4. Сериализатор конфигурации (`DemoDeviceDataProviderSerializer`).

3.5. Редактор конфигурации источника данных (`DemoDataProviderEditor`).

В методе `Unplug()` производится разрегистрация соответствующих объектов.

4. Плагин для СКАУТ-Студио

`Scout.Plugins.SpfTestProtocolPlugin.Client.Studio.DemoStudioPlugin`.

В методе `Plug()` регистрируется описатель типа устройства (`DemoDeviceTypeDescriptor`).

В методе `Unplug()` он разрегистрируется.

5. Плагин для СКАУТ-СильверСтудио

`Scout.Plugins.SpfTestProtocolPlugin.Client.SilverStudio.DemoSilverStudioPlugin`.
Здесь производятся те же действия, что и для плагина для СКАУТ-Студио.

Регистраторы объектов в СКАУТ-Платформе

Для регистрации объектов из клиентского расширения в СКАУТ-Платформе используются следующие классы:

1. `DeviceTypeDescriptionController` – класс-синглтон, регистрирует и разрегистрирует текстовое описание устройства.

- Регистрация:

`public void RegisterDescriptor(Guid deviceType, IDeviceTypeDescriptor descriptor)`, где:
`deviceType` – идентификатор типа устройства; в демо-расширении используется `DemoDeviceTypeId.DeviceTypeId` со значением `ScoutDevices.Reserve0` – это сделано для корректной работы с текущей версией сервера лицензирования (до обновления сервера лицензирования требуется использовать идентификаторы из пула `ScoutDevices.Reserve0…10`).
`descriptor` – описатель устройства; в демо-расширении используется `DemoDeviceTypeDescriptor`.

- Разрегистрация:

`public void UnregisterDescriptor(Guid deviceType)`, где `deviceType` – тот же идентификатор, что использовался при регистрации.

2. `DeviceDataProviderDescriptionController` – класс-синглтон, регистрирует и разрегистрирует текстовое описание источника данных.

- Регистрация:

`public void RegisterDescriptor(Guid dataProviderType, IDeviceDataProviderDescriptor descriptor)`, где:
`dataProviderType` – идентификатор источника данных (значение выбирается разработчиком расширения на свое усмотрение); в демо-расширении используется `DemoDeviceDataProviderId.DataProviderId`.
`descriptor` – описатель источника данных; в демо-расширении используется `DemoDataProviderDescriptor`.

- Разрегистрация:

`public void UnregisterDescriptor(Guid dataProviderType)`, где `dataProviderType` – тот же идентификатор, что использовался при регистрации.

3. `DeviceDataProtocolDescriptionController` – класс-синглтон, регистрирует и разрегистрирует текстовое описание протокола.

- Регистрация:

`public void RegisterDescriptor(Guid dataProtocolType, IDeviceDataProtocolDescriptor descriptor)`, где:
`dataProviderType` – идентификатор протокола (значение выбирается разработчиком расширения на свое усмотрение); в демо-расширении используется `DemoDeviceProtocolId.ProtocolId`.
`descriptor` – описатель источника данных; в демо-расширении используется `DemoDeviceProtocolDescriptor`.

- Разрегистрация:

`public void UnregisterDescriptor(Guid dataProtocolType)`, где `dataProtocolType` – тот же идентификатор, что использовался при регистрации.

4. `DeviceDataProviderSerializationController` – класс-синглтон, регистрирует и разрегистрирует фабрику сериализаторов конфигурации источника данных.

- Регистрация:

`public void RegisterSerializer(Guid dataProviderType, IDeviceDataProviderSerializer descriptor)`, где:
`dataProviderType` – идентификатор источника, должен использоваться тот же, что и при регистрации описания источника данных (в демо-расширении – `DemoDeviceDataProviderId.DataProviderId`);
`serializer` – фабрика сериализаторов/десерализаторов конфигурации источника данных; в демо-расширении используется `DemoDeviceDataProviderSerializer`.

- Разрегистрация:

`public void UnregisterSerializer(Guid dataProviderType)`, где `dataProviderType` – тот же идентификатор, что использовался при регистрации.

5. `DeviceDataProviderController` – класс-синглтон, регистрирует и разрегистрирует фабрику источников данных (портов).

- Регистрация:

`public void RegisterDataProvider(Guid dataProviderId, IDeviceDataProvider dataProvider)`, где:
`dataProviderId` – идентификатор источника, должен использоваться тот же, что и при регистрации описания источника данных (в демо-расширении – `DemoDeviceDataProviderId.DataProviderId`).
`dataProvider` – фабрика портов; в демо-расширении используется `DemoDataProvider`.

- Разрегистрация:

`public void UnregisterDataProvider(Guid dataProviderId)`, где `dataProviderId` – тот же идентификатор, что использовался при регистрации.

6. `DeviceDataProtocolController` – класс-синглтон, регистрирует и разрегистрирует фабрику обработчиков данных из портов.

- Регистрация:

`public void RegisterDataProtocol(Guid protocolId, IDeviceDataProtocol protocol)`, где:
`protocolId` – фабрика обработчиков данных из портов, должен использоваться тот же, что и при регистрации описания протокола (в демо-расширении – `DemoDeviceProtocolId.ProtocolId`).
`protocol` – регистрируемый источник данных; в демо-расширении используется `DemoDataProtocol`.

- Разрегистрация:

`public void UnregisterDataProtocol(Guid protocolId)`, где `protocolId` – тот же идентификатор, что использовался при регистрации.

7. `DeviceDataProviderViewController` – класс-синглтон, регистрирует и разрегистрирует редактор конфигурации источника данных.

- Регистрация:

`public void RegisterConfigurationEditor(Guid dataProviderId, IDeviceDataProviderEditor editor)`, где:
`dataProviderId` – идентификатор источника, должен использоваться тот же, что и при регистрации описания источника данных (в демо-расширении – `DemoDeviceDataProviderId.DataProviderId`).
`editor` – фабрика конфигураций по умолчанию и визуальных представлений конфигурации; в демо-расширении используется `DemoDataProviderEditor`.

- Разрегистрация:

`public void UnregisterConfigurationEditor(Guid dataProviderId)`, где

`dataProviderId` – тот же идентификатор, что использовался при регистрации.

Реализация интерфейсов

Далее перечислены интерфейсы, реализация которых требуется для внедрения поддержки новых устройств, типов источников данных и протоколов в СКАУТ-Платформе. В заголовке пункта указан интерфейс, в скобках – название класса, реализующего его в демо-расширении. В подпунктах – методы и свойства интерфейсов с кратким описанием их назначения.

1. `IDeviceDataPort (DemoDataPort)` – порт, источник данных.

1.1. Метод `bool TryStart(DeviceDataProtocolHandler protocol)` – должен формировать поток данных `IDeviceDataStream (DemoDataProtocol)`, соответствующий своему протоколу, и передавать созданный поток в обработчик протокола.

1.2. Метод `void Stop()` должен закрывать созданный ранее поток данных.

2. `IDeviceDataProtocol (DemoDataProtocol)` – Фабрика обработчиков данных из порта.

2.1. Метод `IDeviceDataProtocolHandler CreateHandler(IDeviceDataStorage dataStorage, IDeviceFilter deviceFilter)` – должен создавать обработчик (`DemoDataProtocolHandler`) потока данных (`DemoDataStream`).

2.1.1. `IDeviceDataStorage` – интерфейс взаимодействия обработчиков данных с системой СКАУТ. Позволяет сохранять успешно распарсенные данные в виде объектов типа `DeviceMessage`.

2.1.2. `IDeviceFilter` – интерфейс, предоставляющий доступ к параметрам фильтрации входящих данных по Id устройства. Позволяет определить, сохранять ли данные, приходящие от тех или иных устройств.

3. `IDeviceDataProtocolHandler (DemoDataProtocolHandler)` – обработчик данных порта.

3.1. Метод `void Handle(IDeviceDataStream stream)` – должен обрабатывать данные от устройства, находящиеся в потоке `stream (DemoDataStream)`, и сохранять их через экземпляр `IDeviceDataStorage` в виде экземпляров `DeviceMessage`.

4. `IDeviceDataProvider (DemoDataProvider)` – фабрика портов с соответствующей конфигурацией.

4.1. Метод `bool TryCreatePort(IDeviceDataProviderConfiguration configuration, out IDeviceDataPort dataPort)` – создание порта (`DemoDataPort`) по предоставляемой конфигурации (`DemoDeviceDataProviderConfiguration`).

4.2. Метод `string SerializeConfiguration(IDeviceDataProviderConfiguration configuration)` – сериализация конфигурации (`DemoDeviceDataProviderConfiguration`).

4.3. Метод `IDeviceDataProviderConfiguration DeserializeConfiguration(string data)` – десериализация конфигурации (`DemoDeviceDataProviderConfiguration`).

5. `IDeviceDataProviderEditor (DemoDataProviderEditor)` – фабрика объектов, отвечающих за представление конфигурации порта.

5.1. Метод `string GetConfigurationDisplayText(IDeviceDataProviderConfiguration configuration)` – должен возвращать строковое представление конфигурации (`DemoDeviceDataProviderConfiguration`), которое будет отображаться пользователю в общей информации об источнике данных

5.2. Метод `DeviceDataProviderViewInfo GetEditView(IDeviceDataProviderConfiguration configuration)` – должен возвращать на основе конфигурации представление (`DemoDataProviderView`) и связанный с ним модель представления (`DemoDataProviderViewModel`) в объекте `DeviceDataProviderViewInfo`, позволяющие вносить изменения в конфигурацию. Данное представление будет отображаться в диалоге изменения параметров порта в СКАУТ-Менеджере.

5.3. Метод `IDeviceDataProviderConfiguration GetEditedConfiguration(DeviceDataProviderViewInfo editView)` – должен извлекать из модели представления (`DemoDataProviderViewModel`) данные конфигурации (`DemoDeviceDataProviderConfiguration`).

5.4. Метод `IDeviceDataProviderConfiguration CreateDefaultConfiguration()` – создает конфигурацию по умолчанию.

6. `IDeviceDataStream (DemoDataStream)` – источник данных от терминалов.

6.1. Свойство `Id` – идентификатор потока, должен назначаться в конструкторе произвольным образом.

6.2. Метод `void ReadAll(Action<IDeviceDataStream, DeviceDataPackage> callback)` – для чтения всех данных, содержащихся в данном потоке.

6.2.1. `Callback` – должен быть вызван при завершении операции чтения.

6.3. Метод `void Read(int byteCount, Action<IDeviceDataStream, DeviceDataPackage> callback)` – аналогично `ReadAll`, но с указанием максимальной длины считываемого массива байт.

6.4. Метод `void Read(byte[] buffer, int offset, int byteCount, Action<IDeviceDataStream, DeviceDataPackage> callback)` – аналогично `ReadAll`, но с указанием максимальной длины считываемого массива байт, отступа от начала потока и целевого массива, который должен быть возвращен в `DeviceDataPackage`.

6.5. Методы `void Write(byte[] data, Action<IDeviceDataStream, DeviceDataPackage> callback)` и `void Write(byte[] data, int offset, int byteCount, Action<IDeviceDataStream, DeviceDataPackage> callback)` – для передачи данных устройству, если требуется такая функциональность.

6.6. Метод `void Close()` – вызывается при завершении работы с потоком. После выполнения операции закрытия должно быть сформировано событие `Closed`.

7. `IDeviceDataProviderConfiguration (DemoDeviceDataProviderConfiguration)` – Конфигурация порта, контейнер для передачи настроек. Нет обязательных для реализации членов.

8. `IDeviceDataProviderSerializer (DemoDeviceDataProviderSerializer)` – фабрика сериализаторов/десерализаторов конфигурации порта.

8.1. Метод `ICustomWriter<IDeviceDataProviderConfiguration> GetWriter()` – возвращает сериализатор конфигурации.

8.2. Метод `ICustomReader<IDeviceDataProviderConfiguration> GetReader()` – Возвращает десериализатор конфигурации.

9. `ICustomWriter<IDeviceDataProviderConfiguration> (DemoDeviceDataProviderWriter)` – сериализатор конфигурации порта.

9.1. Метод `void Write(SerializationStream stream, IDeviceDataProviderConfiguration`

data) – запись объекта в поток сериализации stream. Поток сериализации реализует методы записи объектов различных базовых типов.

10. ICustomReader<IDeviceDataProviderConfiguration> (DemoDeviceDataProviderReader) – десериализатор конфигурации порта.

10.1. Метод IDeviceDataProviderConfiguration Read(DeserializationStream stream) – Чтение из потока десериализации. Чтение данных должно производиться в том же порядке и тех же типов, что и запись при сериализации.

11. IDeviceTypeDescriptor (DemoDeviceTypeDescriptor) – описатель типа терминала.

11.1. Свойство Name – название типа устройства, используется в системе логгирования.

11.2. Свойство Title – название типа устройства, используется в GUI.

11.3. Свойство Description – описание типа устройства, используется в GUI.

12. IDeviceDataProtocolDescriptor (DemoDeviceProtocolDescriptor) – описатель протокола.

12.1. Свойство Name – название протокола, используется в системе логгирования.

12.2. Свойство Title – название протокола, используется в GUI.

12.3. Свойство Description – описание протокола, используется в GUI.

13. IDeviceDataProviderDescriptor (DemoDataProviderDescriptor) – описатель типа источника данных.

13.1. Свойство Name – название источника данных, используется в системе логгирования.

13.2. Свойство Title – название источника данных, используется в GUI.

13.3. Свойство Description – описание источника данных, используется в GUI.