MobileSMARTS 2008

Подключение к внешним системам

Версия 1.0.0.4 от 17 августа 2008

«Mobile SMARTS. Руководство разработчика»

Для использования на территории России и СНГ.

Содержание

ООО «Клеверенс Софт»,

тел.: (495) 589-03-69,

www.cleverence.ru

Введение

Настоящее Руководство посвящено описанию средств интеграции системы «Mobile SMARTS» с внешними системами.

Данное Руководство ориентировано на специалистов по внедрению и программистов прикладных систем.

Что Вы должны знать

Характер изложения данного Руководства предполагает, что вы знакомы с платформой разработки Microsoft .NET, библиотекой .NET Framework, способны отлаживать свои программы и прочитали главу 3 «Разработка и внедрение» руководства «Mobile SMARTS 2008 - Установка и внедрение».

Если вы недостаточно хорошо владеете требуемыми понятиями и навыками, рекомендуем обратиться к документации платформе .NET и документации разработчика «Mobile SMARTS».

Глава 1. Общие сведения

Mobile SMARTS 2008 предоставляет несколько средств интеграции с внешними системами:

1. COM-компонента доступа к серверу «Mobile SMARTS» для внешней системы

   Позволяет выгружать на сервер «Mobile SMARTS» справочники пользователей, их групп, справочник номенклатур и единиц измерения, выгружать документы-задания и загружать обратно документы исполненных заданий, а также формировать на сервере «Mobile SMARTS» необходимые дополнительные справочники произвольного формата.

2. Реализации интерфейса IConnector

   Позволяют серверу «Mobile SMARTS» вызывать произвольные методы внешней системы или служебной компоненты, тем самым позволяя серверу «Mobile SMARTS» сигнализировать внешней системе или компоненте о различных событиях, если это явно прописано в конфигурации, а также обеспечивать вызовы сервисных методов с толстого клиента на терминале и возврат соответствующих значений. Интерфейс IConnector можно использовать для написания любых расширений.

3. Реализации интерфейса IExtSystemConnector

   Обеспечивают бесшовную интеграцию сервера «Mobile SMARTS» с внешней системой, позволяя не только вызывать произвольные методы внешней системы с толстого клиента терминала и сигнализировать ей о различных событиях сервера, но и включать внешнюю систему в процесс поиска номенклатуры и документов-заданий по кодам, избавляя итоговое решение от необходимости периодической синхронизации данных.

Реализация любого из указанных интерфейсов должна быть выполнена под Microsoft .NET Framework 1.1 или 2.0 на любом из доступных языков .NET, скомпилирована в виде отдельной DLL и помещена в папку bin сервера «Mobile SMARTS». Она будет подхвачена сервером сразу после его перезапуска. Если во время загрузки возникли проблемы, они будут отражены в server_errors.log в папке установки сервера.

Код коннекторов не только находится в одном приложении с сервером, но и имеет прямой доступ ко всем данным сервера, в частности к среде окружения Environment, задающей текущую конфигурацию, хранилищу документов DocumentStorage и всем справочникам, таким как ProductsBook, UnitsBook и т.д. (см. «Mobile SMARTS 2008 Компонента доступа»). В то время как в COM-компоненте все

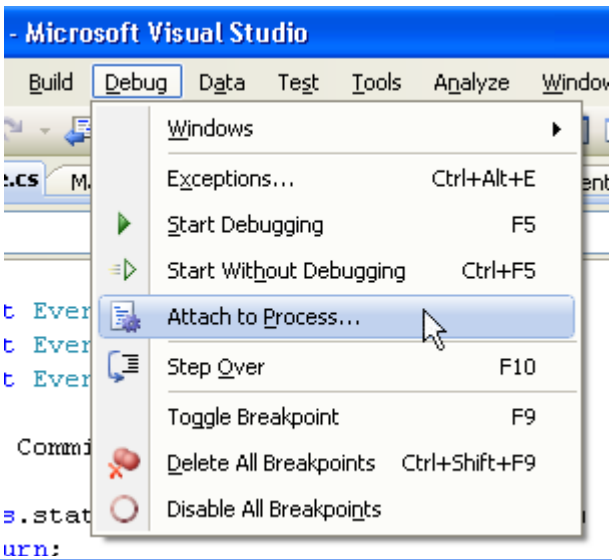
такие классы указаны, как имеющие экземпляры (чтобы создавать их в COM), на стороне сервера они представлены в виде статических классов с прямым доступом ко всем свойствам и коллекциям.

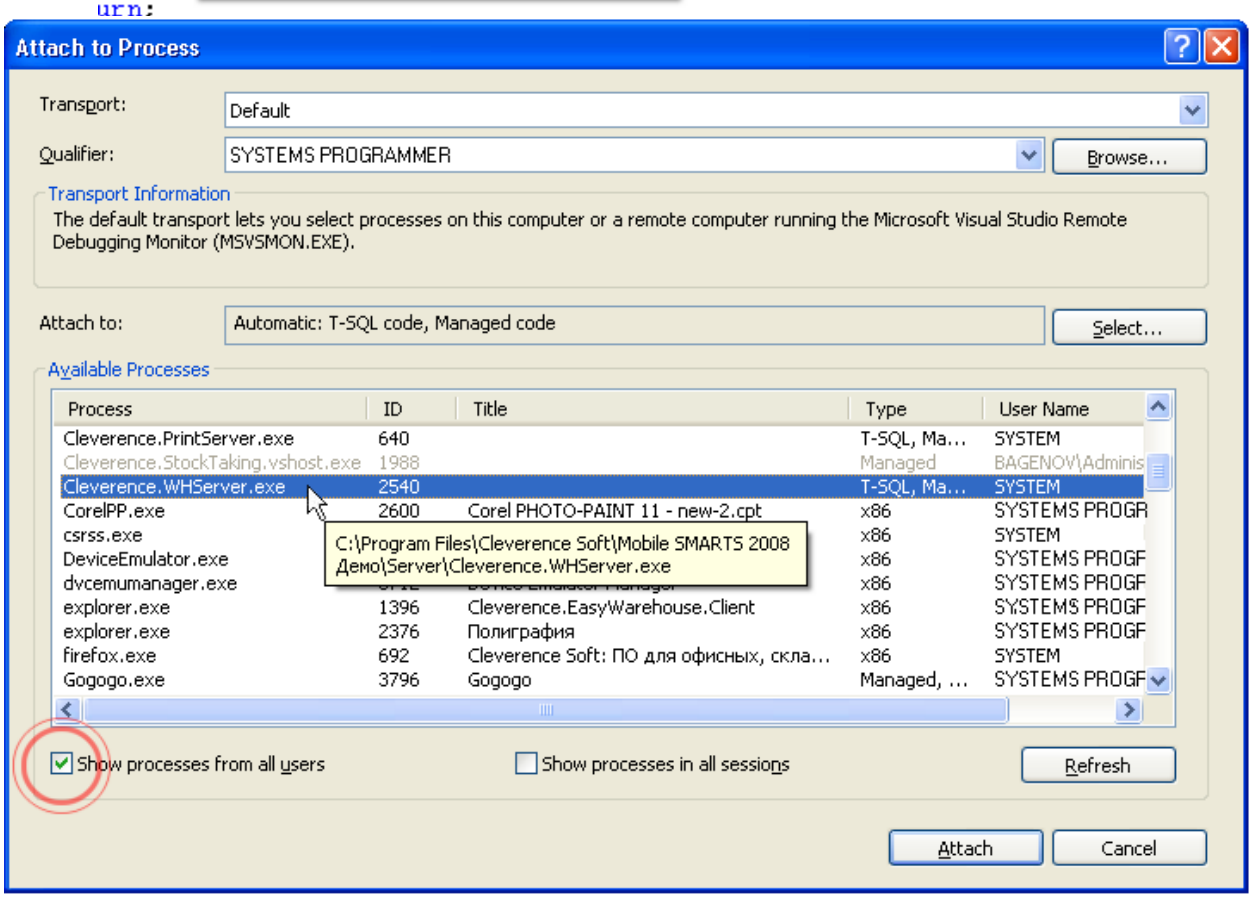
Создание коннекторов под Mobile SMARTS – это серверное это многопоточное программирование с высокими требованиями к скорости выполнения операций. Доступ к методам и коллекциям статических данных сервера не залочен и они не потокобезопасны. При обращении к данным сервера (в особенности на запись) код коннектора должен самостоятельно организовывать критические сессии, обрамляя код структурой:

```
lock(Cleverence.Warehouse.Lock.<Нужное свойство>)
{
        ...
}
```

где «Нужное свойство» – свойство класса Lock, из имени которого понятно объект sync root каких данных оно возвращает.

Среда Microsoft Visual Studio позволяет выполнять отладку разработанного коннектора во время его реальной работы при помощи функции «Attach to process»:





Для этого необходимо открыть свой проект Microsoft Visual Studio, в котором ведется разработка коннектора, скомпилировать его и скопировать полученные dll и pdb файлы в папку bin сервера (вручную или путем PostBuild Event), а затем перезапустить сервер и выполнить «Attach to process». Теперь можно устанавливать в исходном коде коннектора точки останова и проводить отладку.

Глава 2. Разработка коннекторов к внешним системам

§ 1. Интерфейс IConnector

Интерфейс IConnector существует в двух вариантах – серверном и клиентском. Серверный вариант реализует собственно логику работы коннектора. Клиентский вариант позволяет пользователю задавать параметры подключения и любые другие свойства коннектора в панели управления «Mobile SMARTS». Оба варианта должны быть реализованы в виде двух отдельных dll.

Клиентская версия должна реализовать только два свойства: Id и Enabled. Свойство Id отличает конкретный экземпляр коннектора ото всех других коннекторов к той же системе (но другим базам).

Серверная версия должна реализовать свойства Id и Enabled, а также методы Initialize и InvokeMethod. Подключение коннектора к системе происходит не в момент его создания, а в момент первого к нему обращения через метод Initialize, о вызове которого позаботится сервер «Mobile SMARTS».

Оба класса-реализации должны находиться в одинаковых namespace и иметь совершенно одинаковые имена. Схема передачи параметров из панели управления на сервер работает следующим образом: в панели управления пользователь создает экземпляр коннектора, выбрав его по имени в диалоговом окне создания нового внешнего соединения. Пользователь редактирует публичные свойства этого объекта при помощи PropertyGrid. В момент сохранения информации на сервер клиентский экземпляр коннектора сериализуется в XML автоматической сериализацией инфраструктуры Mobile SMARTS, при этом в XML попадают все публичные редактируемые свойства по их именам. Затем на сервере инфраструктура Mobile SMARTS пытается десеарилизовать полученный XML в объект путем поиска класса по имени, указанном в XML, создания его экземпляра и проставления ему всех переданных публичных свойств по именам. Именно поэтому полные имена реализаций должны совпадать, а серверная реализация предоставлять на запись все те свойства с теми же именами, что были реализованы в клиентской версии (при этом серверная версия может содержать любое число собственных свойств и методов сверх этого).

Для реализации клиентской версии интерфейса IConnector следует создать новый проект в Microsoft Visual Studio, подключить в него в качестве Referenced Assembly следующие сборку Cleverence.MobileSMARTS.ComConnector.dll из папки панели управления и создать новый класс, реализующий интерфейс Cleverence.Connectivity.IConnector из этой сборки.

Для реализации серверной версии интерфейса IConnector следует создать новый проект в Microsoft Visual Studio, подключить в него в качестве Referenced Assembly следующие сборки из папки bin сервера:

- Cleverence.DataCollection.dll
- Cleverence.Connetivity.dll
- Cleverence.MobileSMARTS.dll

и создать новый класс, реализующий интерфейс Cleverence.Connectivity.IConnector из серверной сборки Cleverence.Connetivity.

Пример реализации клиентского варианта интерфейса

```
using System;
using System.Configuration;
using System.Runtime.InteropServices;

namespace MySystem
{
	[Guid("CD86EA85-B168-4823-B509-E916D685BB72")]
	[ProgId("MySystem.MySystemClientConnector")]
	[Cleverence.Warehouse.Design.DisplayTypeName("Шлюз к моей системе v.1")]
	public class MySystemConnector : IConnector
	{
		public MySystemConnector()
		{
		}

		#region IConnector Members

		private string id;
		[System.ComponentModel.Description("Идентификатор.")]
		public string Id
		{
			get { return this.id; }
			set { this.id = value; }
		}

		private bool enabled;
		[System.ComponentModel.Browsable(false)]
		public bool Enabled
		{
			get { return this.enabled; }
			set { this.enabled = value; }
		}

		#endregion

		#region Дополнительные свойства

		private string user;
		[System.ComponentModel.Description("Имя пользователя.")]
		public string User
		{
			get { return this.user; }
			set { this.user = value; }
		}

		private string password;
		[System.ComponentModel.Description("Пароль.")]
		[System.ComponentModel.TypeConverter("Cleverence.Warehouse.Design.
		PasswordConverter, Cleverence.Warehouse.Com.Design")]
		public string Password
		{
			get { return this.password; }
			set { this.password = value; }
		}

		#endregion
	}

}
```

Пример реализации серверного варианта интерфейса

```
using System;
using System.Configuration;
using System.Runtime.InteropServices;

namespace MySystem
{
	public class MySystemConnector : IConnector, IDisposable
	{
		public MySystemConnector()
		{
		}


		private MYSYS.Interop.COMConnector connector;


		private string user;
		public string User
		{
			get { return this.user; }
			set { this.user = value; }
		}

		private string password;
		public string Password
		{
			get { return this.password; }
			set { this.password = value; }
		}

		#region IConnector Members

		private string id;
		public string Id
		{
			get { return this.id; }
			set { this.id = value; }
		}

		public bool Initialized
		{
			get { return this.connector != null; }
		}

		private bool enabled = true;
		public bool Enabled
		{
			get { return this.enabled; }
			set
			{
				if(value == false && this.connector != null)
					this.Dispose();

				this.enabled = value;
			}
		}
```

```
            public void Initialize()
            {
                  this.Dispose();
                  this.connector = new MYSYS.Interop.COMConnector();
                  this.connector.Logon(this.user, this.password);
            }

            public object InvokeMethod(string methodName, object[] args)
            {
                  if(this.connector == null)
                        throw new InvalidOperationException(
                              this.GetType().Name + " is not initialized.");

                  if(this.enabled == false)
                        throw new InvalidOperationException(
                              this.GetType().Name + " is not enabled.");

                  return this.connector.Call(methodName, args, null, null);
            }


            #endregion


            #region IDisposable Members

            public void Dispose()
            {
                  if(this.connector == null)
                        return;

                  Marshal.ReleaseComObject(this.connector);
                  this.connector = null;
            }

            #endregion

            ~ MyConnector()
            {
                  this.Dispose();
            }
      }
}
```

§ 2. Интерфейс IExtSystemConnector

Для реализации интерфейса IExtSystemConnector следует создать новый проект в Microsoft Visual Studio, подключить в него в качестве Referenced Assembly следующие сборки:

- Cleverence.DataCollection.dll
- Cleverence.Connetivity.dll
- Cleverence.MobileSMARTS.dll

и создать новый класс, реализующий интерфейс Cleverence.Connectivity.IExtSystemConnector, например так:

```
using System;
using System.Collections;
using Cleverence.Connectivity;
using Cleverence.Warehouse;
```

```
namespace MySystem
{
      public class MySystemConnector : IExtSystemConnector
      {
            …
      }
}
```

Конструктор коннектора IExtSystemConnector

Несмотря на то, что dll с коннектором подхватывается сервером «Mobile SMARTS» сразу после его запуска, экземпляр коннектора будет создан только при первой необходимости обращения к нему. Это сделано по той причине, что Microsoft Visual Studio не может подключаться на отладку к процессам, которые не загрузили dll, которую необходимо отлаживать. Если создание экземпляра коннектора будет происходить одновременно с загрузкой dll и запуском сервера, у разработчика коннектора не будет возможности отладить код конструктора, т.к. он будет исполнен еще до того, как Microsoft Visual Studio подключится к серверу «Mobile SMARTS» на отладку.

Методы IExtSystemConnector

	Наименование	Описание
	InvokeMethod	Позволяет выполнять вызовы методов системы с толстого клиента терминала.
	QueryProduct	Перегружен. Запрашивает объект номенклатуры, соответствующий переданным кодам.
	QueryDocument	Перегружен. Запрашивает объект документа Document, соответствующий переданному коду или подлежащий выдаче указанному сотруднику.
	OnDocumentAdded	Вызывается при появлении на сервере нового документа с терминала или выгруженного через COM-компоненту из учетной системы.
	OnDocumentAppointed	Вызывается при отправке документа на терминал конкретному пользователю.
	OnDocumentChanged	Вызывается при изменении документа на сервере путем сохранения новой версии документа с терминала на сервер или выгрузки новой версии документа из учетной системы через COM-компоненту.

	OnDocumentCompleted	Вызывается при выполнении задания пользователем на терминале и оправке им документа с терминала на сервер (автоматически или в батч-режиме через кредл).
	OnDocumentRemoved	Вызывается при удалении документа учетной системой через COM-компоненту.

Метод InvokeMethod

```
public InkokeResult ImvokeMethod(string methodName, InvokeArgs args)
```

Позволяет выполнять вызовы методов системы с толстого клиента терминала.

methodName	Имя метода, как это указано в конфигурации на терминале в действии Вызов метода внешней системы. Если в действии задано также имя класса, то это будет «имя класса.имя метода».
args	Переданные именованные аргументы в виде объекта InvokeArgs (см. «Mobile SMARTS 2008 Компонента доступа»).
результат	Результат исполнения в виде объекта InvokeResult (может содержать несколько именованных результатов, см. «Mobile SMARTS 2008 Компонента доступа») или null.

Метод QueryProduct для поиска по коду

```
public Product QueryProduct(string productId)
```

Возвращает товар, соответствующий переданному коду. Вызывается при получении номенклатур из кодов, указанных в стоках документов, а также при поисках по коду на терминале. Коннектор может как кешировать возвращаемые объекты Product, так и создавать их каждый раз с нуля – никаких ограничений на это не накладывается. Возвращаемые коннектором объекты сервером не кешируются, а отправляются клиентскому приложению на терминал в виде XML и собираются сборщиком мусора .NET.

productId	Строковый код номенклатуры.
Результат	Результат исполнения в виде объекта Product (может содержать несколько упаковок, см. «Mobile SMARTS 2008 Компонента доступа») или null, если ничего не найдено.

Метод QueryProduct для вводимых данных

```
public PacketProductCollection QueryProduct(string userData,
                                            string documentId, User user)
```

Возвращает товар, соответствующий сканированным данным или введенной строке, в том случае, если он не был найден сервером в справочнике товаров путем поиска по штрихкодам номенклатуры и её упаковок с учетом шаблонов штрихкода, а также, кодам и артикулам. Коннектор может как кешировать

возвращаемые объекты Product, так и создавать их каждый раз с нуля – никаких ограничений на это не накладывается. Возвращаемые коннектором объекты сервером не кешируются, а отправляются клиентскому приложению на терминал в виде XML и собираются сборщиком мусора .NET.

userData	Буквально строка символов, введенная на терминале для поиска номенклатуры – может быть штрихкод, может быть артикул, может быть мусор.
documentId	Код документа, из которого производится запрос номенклатуры. Для документов, созданных на терминале, имеет вид «new_....» и может быть найден в документах на сервере только в том случае, если был перед этим сохранен с терминала на сервер действием  Сохранение документа на сервер. Для виртуальных документов имеет вид «new_....», никогда не может быть найден в документах на сервере и не может быть сохранен на сервер действием  Сохранение документа на сервер.
user	Объект пользователя, который работает на терминале и желает получить номенклатуру.
Результат	Список подходящей номенклатуры в виде коллекции объектов PacketProduct (может содержать количество и любые другие данные, см. «Mobile SMARTS 2008 Компонента доступа») или `null`, если ничего не найдено.

Метод QueryDocument для известного кода документа

```
public Document QueryDocument(string barcode, DocumentType documentType)
```

Возвращает документ, соответствующий сканированному штрихкоду или введенной строке, в том случае, если он не был найден сервером в журнале документов.

barcode	Буквально строка символов, введенная на терминале для поиска документа – может быть штрихкод, может быть код, может быть мусор.
documentType	Тип документа, который пользователь пытается открыть по штрихкоду.
Результат	Результат исполнения в виде объекта Document или `null`, если ничего не найдено.

Метод QueryDocument для запроса задания

```
public DocumentCollection QueryDocument(User user, Warehouse warehouse)
```

Возвращает документ, соответствующий новому заданию для указанного пользователя. Вызывается каждый раз, когда терминал выполняет проверку новых документов-заданий (раз в несколько секунд – указывается для конкретного терминала при установке «Mobile SMARTS» на терминал). Код коннектора

может самостоятельно отсеивать слишком частые обращения, заведя у себя соответствующую переменную «Когда к нам обращались в последний раз».

user	Объект пользователя, который запрашивает нет ли для него новых заданий.
warehouse	Склад, на котором работает пользователь.
Результат	Результат исполнения в виде объекта Document или `null`, если ничего нет.

Метод OnDocumentAdded

```
public void OnDocumentAdded(Document document)
```

Вызывается при появлении на сервере нового документа с терминала или выгруженного через COM-компоненту из учетной системы.

document	Объект документа, связанного с событием.

Метод OnDocumentAppointed

```
public void OnDocumentAppointed(Document document)
```

Вызывается при отправке документа на терминал конкретному пользователю.

document	Объект документа, связанного с событием.

Метод OnDocumentChanged

```
public void OnDocumentChanged(Document document)
```

Вызывается при изменении документа на сервере путем сохранения новой версии документа с терминала на сервер или выгрузки новой версии документа из учетной системы через COM-компоненту.

document	Объект документа, связанного с событием.

Метод OnDocumentCompleted

```
public void OnDocumentCompleted(Document document)
```

Вызывается при выполнении задания пользователем на терминале и оправке им документа с терминала на сервер (автоматически или в батч-режиме через кредл).

document	Объект документа, связанного с событием.

Метод OnDocumentRemoved

```
public void OnDocumentRemoved(Document document)
```

Вызывается при удалении документа учетной системой через COM-компоненту.

document	Объект документа, связанного с событием.

Контакты

 По вопросам поддержки обращайтесь по указанным реквизитам компании:

Cleverence Soft,
2й проезд Перова поля, д.2, стр.4,
email: support@cleverence.ru