Управляемые операции

# Введение

### Общее

Модуль *Xafari.ManagedOperations* реализует функциональность управляемых операций. Под управляемыми операциями понимается исполнение некоторого прикладного метода с возможностью его приостановки, продолжения и прерывания. Во время исполнения осуществляется логирование и мониторинг процесса.

# Возможности разработчика

### Настройка сервиса

Для начала работы с управляемыми операциями разработчику необходимо сначала описать их функциональность.

**Важно**: Для описания управляемой операции необходимо добавить ссылку на *Xafari.ManagedOperations.dll* в модуль, где она описана.

**Важно**: Для использования сервиса управляемых операций необходимо добавить модуль XafariManagedOperationsWinModule через дизайнер для WinApplication, либо XafariManagedOperationsWebModule для WebApplication. Добавлять модуль можно и через код, но через дизайнер это делать удобнее и проще, т.к. тогда автоматически будут добавлены другие модули, от которых зависят модули управляемых операций.

**Важно**: Для просмотра и управления управляемыми операциями служит конфигурационный модуль **XafariManagedOperationsCfgModule** (сборка *Xafari.ManagedOperations.Cfg.dll*). Использование данного модуля предполагает следующие стратегии:

1. Модуль непосредственно включается в состав конечного приложения. В этом случае функции администрирования будут доступны в конечном приложении.
2. Конечное приложение разрабатывается в расчете на использование отдельного инструмента администрирования - XAS (Xafari Applications Support). Дополнительную информацию см. в документации “[XAS. Руководство разработчика](https://by01-tfs05.topsoft.local/sites/Ranet/Xafari/Shared%20Documents/x09/Documentation/XAS.Developers%20Guide.ru.docx)”.

Модуль **XafariManagedOperationsCfgModule** добавляет в систему навигации NavigationItems группу «**Конфигурация**(Configuration)\**Управляемые операции** (Managed operations)». Она содержит элементы навигации для просмотра и управления управляемыми операциями. Подробности см. [Возможности администратора](#_Возможности_администратора).

Ниже приведен пример управляемой операции:

using System.Threading;

using DevExpress.ExpressApp;

using Xafari.ManagedOperations;

namespace ManagedOperationApp.Module.Operations

{

 public class SimpleOperation

 {

 private ManagedOperation Process { get; set; }

 public int Count { get; set; }

 public int Timeout { get; set; }

 public SimpleOperation(XafApplication application)

 {

 this.Process = new ManagedOperation(application)

 {

 ProcessCode = (item => this.ExecuteCore(item)),

 };

 Count = 4;

 Timeout = 5;

 this.Process.TotalStep = Count;

 }

 public void ExecuteCore(IManagedOperation operation)

 {

 for (int i = 0; i < Count; i++)

 {

 operation.NextStep(string.Format("Stage {0}", i), i, Count);

 Thread.Sleep(Timeout \* 1000);

 }

 }

 }

}

### Синхронные операции

Некоторые операции запускаются синхронно, т.е. для продолжения работы и выполнения любых действий в клиентском приложении необходимо сначала дождаться окончания запущенной операции.

Для отображения хода выполнения синхронной операции существует три основных формы:

* с индикатором в виде прогресс-бара;
* с индикатором в виде бесконечной прокрутки;
* форма для операции с подпроцессами.

**Операция с прогресс-баром**. Инициировать запуск управляемой операции можно разными способами. Рассмотрим самый простой вариант – с помощью кнопки. Для этого создадим контроллер с действием StartSimpleOperation. За то, чтобы ход выполнения отображался в виде прогресс бара, отвечает метод

SyncManagedOperationHelper.InitShowViewParametersProgress.

Ниже приведен пример кода, запускающий такую операцию:

using DevExpress.ExpressApp;

using ManagedOperationApp.Module.Operations;

using Xafari.ManagedOperations;

namespace ManagedOperationApp.Module.Controllers

{

 public partial class StartManagedOperationViewController : ViewController

 {

 public StartManagedOperationViewController()

 {

 InitializeComponent();

 RegisterActions(components);

 }

 private void startSimpleOperation\_Execute(object sender, DevExpress.ExpressApp.Actions.SimpleActionExecuteEventArgs e)

 {

 var managedOperation = new ManagedOperation(this.Application)

 {

 ZoneType = ManagedOperationZoneTypes.Local,

 Name = "Managed operation with progress bar",

 ProcessCode = (item => new SimpleOperation(this.Application).ExecuteCore(item)),

 };

 managedOperation.Start();

 SyncManagedOperationHelper.CreateHelper(managedOperation).

InitShowViewParametersProgress(e.ShowViewParameters, false);

 }

 }

}

По нажатию кнопки «StartSimpleOperation» будет запущена наша созданная управляемая операция (Рис. 1).



Рис.1 – Запуск операции с индикатором выполнения в виде прогресс-бара

**Операция с индикатором в виде бесконечной прокрутки**. Создается аналогично, только используется метод InitShowViewParametersMarquee:

private void startSimpleOperation\_Execute(object sender, DevExpress.

ExpressApp.Actions.SimpleActionExecuteEventArgs e)

{

 var managedOperation = new ManagedOperation(this.Application)

 {

 ZoneType = ManagedOperationZoneTypes.Local,

 Name = "Managed operation with marquee bar",

 ProcessCode = (item => new SimpleOperation(this.Application).

ExecuteCore(item)),

 };

 managedOperation.Start();

 SyncManagedOperationHelper.CreateHelper(managedOperation).

InitShowViewParametersMarquee(e.ShowViewParameters, true);

}

На рисунке 2 показано окно выполнения в виде бесконечной прокрутки.



Рис.2 – Запуск операции с индикатором выполнения в виде бесконечной прокрутки

**Операция с подпроцессами.** Бывают случаи, когда необходимо запустить одну управляемую операцию в рамках другой. Тогда эта «вложенная» операция называется подпроцессом. Для управляемых операций с подпроцессами необходимо отображать собственную форму выполнения. Ниже показан пример создания сложной операции:

private void startSimpleOperation\_Execute(object sender, DevExpress.ExpressApp.Actions.SimpleActionExecuteEventArgs e)

{

 var managedOperation = new ManagedOperation(this.Application)

 {

 ZoneType = ManagedOperationZoneTypes.Local,

 Name = "Managed operation with subprocesses",

 ProcessCode = (item => new SimpleOperation(this.Application).ExecuteCore(item)),

 };

 managedOperation.CreateSubProcess("Subprocess 1", item => new SimpleOperation(this.Application).ExecuteCore(item));

 managedOperation.CreateSubProcess("Subprocess 2", item => new SimpleOperation(this.Application).ExecuteCore(item));

 managedOperation.Start();

 SyncManagedOperationHelper.CreateHelper(managedOperation).InitShowViewParametersSubprocess(e.ShowViewParameters, false);

}

Подпроцессы создаются с помощью метода ManagedOperation.CreateSubrocess(), куда передаются название операции, а также исполняемый функционал. Форма для операции с подпроцессами (рисунок 3) инициализируется методом InitShowViewParametersSubprocess().



Рис.3 – Запуск операции подпроцессами

**Автоматическое закрытие формы выполнения операции после завершения** настраивается через параметр, передаваемый в метод SyncManagedOperationHelper’a:

InitShowViewParameters...(viewParameters, **true/false**)

true – закрывать автоматически форму, false – не закрывать.

### Асинхронные операции

**Простая асинхронная операция.** Асинхронные операции дают возможность пользователю продолжать работу с приложением во время выполнения. Ниже показан пример создания и запуска асинхронной операции:

 public class AsyncProcess

 {

 private ManagedOperation Process { get; set; }

 public int Count { get; set; }

 public int Timeout { get; set; }

 public AsyncProcess(XafApplication application)

 {

 Count = 4;

 Timeout = 15;

 Process.TotalStep = Count;

 this.Process = new ManagedOperation(application)

 {

 Name = "AsyncProcess",

 ProcessCode = (item => this.Stage1(item)),

 };

 }

 public void Run()

 {

 this.Process.Start();

 }

 void Stage1(IManagedOperation operation)

 {

 for (int i = 0; i < Count; i++)

 {

 operation.NextStep(string.Format("{0} step", i), i, Count);

 Thread.Sleep(Timeout \* 1000);

 }

 }

 }

Занесение информации в лог (с индикатором процента выполнения) осуществляется методом

 NextStep(string message, int message, int message);

Для сохранения только сообщений (без прогресса) используется метод

NextStep(string message);

Вызов данной операции осуществляется также через контроллер, по нажатию кнопки на форме:

private void startSimpleOperation\_Execute(object sender, DevExpress.ExpressApp.Actions.SimpleActionExecuteEventArgs e)

{

 new AsyncProcess(this.Application).Run();

}

**Асинхронная операция с подпроцессами.**

Отличие от AsyncProcess только в конструкторе операции, в котором создаются подпроцессы. Для создания подпроцессов воспользуемся методом класса ManagedOperation

CreateSubProcess(string name, Action<IManagedOperation> action):

public AsyncProcessWithSubprocesses(XafApplication application)

{

 Process = new ManagedOperation(application)

 {

 Name = "AsyncWithSubprocesses",

 TracingStrategy = TracingStrategy.CopyToParent

 };

 Process.CreateSubProcess("Subprocess 1", item => this.Stage1(item));

 Process.CreateSubProcess("Subprocess 2", item => this.Stage1(item));

 Process.CreateSubProcess("Subprocess 3", item => this.Stage1(item));

 Process.CreateSubProcess("Subprocess 4", item => this.Stage1(item));

 Process.CreateSubProcess("Subprocess 5", item => this.Stage1(item));

 Count = 5;

 Timeout = 15;

}

Вызов операции на выполнение аналогичен вызову простой операции.

### Уникальные операции

Для некоторых операций должна обеспечиваться уникальность исполнения в рамках базы данных. Например, не может одновременно исполняться два процесса пересчета плана или расчета зарплаты (в каждый момент времени может быть запущен только одним пользователем, только один раз). Уникальность реализована как запрет создания экземпляра при наличии исполняющейся операции.

Рассмотрим пример такой операции. Исходный код аналогичен рассмотренным выше примерам. Изменение – в конструкторе. В нем создаем объект ManagedOperation при помощи конструктора с двумя параметрами, вторым является уникальный идентификатор.

 public class UniqueOperationWithSubProcesses

 {

 private ManagedOperation Process { get; set; }

 public int Count { get; set; }

 public int Timeout { get; set; }

 public UniqueOperationWithSubProcesses(XafApplication application)

 {

 var operationId = new Guid("DCEDA7A6-B8E2-4306-AB52-0B0220C826CD");

 Process = new ManagedOperation(application, operationId)

 {

 Name = "Unique process with subprocesses"

 };

 Process.CreateSubProcess("Subprocess 1", item => this.ExecuteCore(item));

 Process.CreateSubProcess("Subprocess 2", item => this.ExecuteCore(item));

 Process.CreateSubProcess("Subprocess 3", item => this.ExecuteCore(item));

 Process.CreateSubProcess("Subprocess 4", item => this.ExecuteCore(item));

 Process.CreateSubProcess("Subprocess 5", item => this.ExecuteCore(item));

 Count = 4;

 Timeout = 15;

 }

 public void Run()

 {

 this.Process.Start(true);

 }

 public void ExecuteCore(IManagedOperation operation)

 {

 for (int i = 0; i < Count; i++)

 {

 operation.NextStep(string.Format("Stage {0}", i), i, Count);

 Thread.Sleep(Timeout \* 1000);

 }

 }

 }

Если уникальная операция уже была запущена и в данный момент исполняется, то не обходимо уведомить пользователя об этом. При необходимости отображать подробную информацию об ошибке, нужно в методе Run() вызвать this.Process.Start() либо передать в метод this.Process.Start(bool generateDuplicateException) параметр true. Подробное сообщение показано на рисунке 4:



Рис.4 – Детальное описание ошибки при повторном запуске уникальной операции

Если достаточно просто уведомить пользователя о том, что уникальная операция уже запущена и выполняется, нужно передать в метод Start(bool generateDuplicateException) параметр false.

 public void Run()

 {

 this.Process.Start(false);

 }

Простое уведомление показано на рисунке 5:



Рис.5 – Простое уведомление

### Индексирование операций

Если необходимо задать подпроцессу другой порядковый номер, используйте следующий метод:

 public SomeProcess(XafApplication application)

 {

 Process = new ManagedOperation(application)

 {

 Name = "SomeProcessWithSubprocesses",

 TracingStrategy = TracingStrategy.CopyToParent

 };

 Process.CreateSubProcess("Subprocess 1", item => this.Stage1(item));

 Process.CreateSubProcess("Subprocess 2", item => this.Stage1(item));

 Process.CreateSubProcess("Subprocess 3", item => this.Stage1(item));

 Process.CreateSubProcess("Subprocess 4", item => this.Stage1(item));

 Process.CreateSubProcess("Subprocess 5", item => this.Stage1(item));

 Process.CreateSubProcess("Subprocess with 92 index", item => this.Stage1(item), this.Count, 92);

 Process.CreateSubProcess("Subprocess with 789 index", item => this.Stage1(item), this.Count, 789);

 Process.CreateSubProcess("Subprocess with 8 index", item => this.Stage1(item), this.Count);}

По умолчанию, если не передавать index в качестве параметра метода

CreateSubProcess(string name, Action<IManagedOperation> action, int total, int? index = null),

то будет использован порядковый номер, равный количеству подпроцессов данной операции. Если будет передан какой-либо номер, он будет присвоен подпроцессу в качестве индекса.

### Локальные и глобальные операции

По умолчанию, все операции имеют глобальный тип, т.е. ZoneType = ManagedOperationZoneTypes.Global. В этом случае данные о всех управляемых операциях хранятся в базе данных.

Если информацию об операции необязательно хранить в базе, то при инициализации ее можно пометить как локальную:

 private void startSimpleOperation\_Execute(object sender, DevExpress.ExpressApp.Actions.SimpleActionExecuteEventArgs e)

 {

 var managedOperation = new ManagedOperation(this.Application)

 {

 ZoneType = ManagedOperationZoneTypes.Local,

 Name = "Managed operation with subprocesses",

 ProcessCode = (item => new SimpleOperation(this.Application).ExecuteCore(item)),

 };

 managedOperation.Start();

 SyncManagedOperationHelper.CreateHelper(managedOperation).InitShowViewParametersProgress(e.ShowViewParameters, false);

 }

Данные о локальных операциях хранятся в памяти, поэтому следует учитывать, что при завершении приложения информация об этих операциях будет утеряна.

### Логирование управляемых операций

Логирование позволяет сохранить историю выполнения управляемых операций для последующего просмотра и анализа. Просмотр информации о выполнении операции осуществляется по нажатию кнопки «Показать лог» («Show log») на форме управляемой операции (Рис. 6).



Рис.6 – Кнопка для просмотра лог-информации

Настройка параметров логирования производится при инициализации операции. Для управления механизмом логирования применяются следующие параметры:

* Interval (int) – этот параметр отвечает за то, с каким временным интервалом будет обновляться информация в лог-файле. Задается в миллисекундах. По умолчанию установлен на 2000 миллисекунд. Если выставлено значение 0, то обновление будет происходить на каждом шаге.
* Split (bool) – указывает на необходимость разбиения лога на части. По умолчанию false. HHРазбиение на части будет полезно, если операция довольно «тяжелая» либо в процессе ее выполнения формируется большое количество информации.
* Count (int) – если необходимо разбивать лог на части (Split = true), то данный параметр устанавливает минимальное значение записей, которое будет храниться в одной части. По умолчанию выставлена 1000 записей. Если (Split = false), тогда данный параметр не учитывается.

Например, в данном примере будет происходить логирование в одну запись. Информация добавляется с интервалом в 2 сек.

private void SyncProgress\_Execute(object sender, SimpleActionExecuteEventArgs e)

{

 var operation = new Operation1(this.Application);

 var mo = new ManagedOperation(this.Application)

{

 ZoneType = ((ActionBase)sender).Id == "SyncLocalProgress" ? ManagedOperationZoneTypes.Local : ManagedOperationZoneTypes.Global,

 Name = "SyncProgress",

 ProcessCode = (operation.Stage2),

 TotalStep = operation.Count,

 Split = false,

 Interval = 2000,

 };

 mo.Start();

 SyncManagedOperationHelper.CreateHelper(mo).InitShowViewParametersProgress

(e.ShowViewParameters, false);}

Так выглядит окно лога для данной операции (Рис. 7):



Рис.7 – Хранение лога в одной записи

Например, если необходимо разбивать лог на части, тогда:

 Split = true,

 Interval = 2000,

 Count = 4000,

Так выглядит окно лога для данной операции (Рис. 8):



Рис.8 – Разбиение на части

Если в результате выполнения управляемой операции возникли предупреждения либо ошибки, то логирование происходит следующим образом: если в блоке возникло предупреждение, то весь блок помечается типом TraceMessageTypes.Warning. При ошибке весь блок помечается типом TraceMessageTypes.Error. Если в одном блоке возникли и предупреждения и ошибки, тогда он помечается TraceMessageTypes.ErrorAndWarning. На рис.9 показаны различные типы сообщений:



Рис.9 – Типы сообщений

### Настройки уровней критичности для логирования

 Уровни: Off, Errors, Warnings, Info, Verbose

Используется enum System.Diagnostics.TraceLevel. В случае длительных операций значения Info и Verbose имеют одинаковый смысл – будет записываться вся информация о ходе выполнения. Errors – будут сохраняться только ошибки, Warnings – ошибки и предупреждения. Если выбран Off, то кнопка открытия протокола на форме будет недоступна.

Настройка уровней критичности осуществляется:

1. Через .config файл приложения(App.config/Web.config)
2. Непосредственно при объявлении операции

Настройка уровня критичности с помощью .config файла осуществляется через переменную XafariOperations

<system.diagnostics>

<switches>

<!-- 0-Off, 1-Errors, 2-Warnings, 3-Info, 4-Verbose. -->

<add name="eXpressAppFramework" value="3" />

<add name="XafariManagedOperationsLogLevel" value="3"/>

</switches>

</system.diagnostics>

Для каждой операции можно вручную задать желаемый уровень критичности. Это делается при объявлении операции:

var operation = new Operation1(this.Application);

var mo = new ManagedOperation(this.Application)

{

 ZoneType = ((ActionBase)sender).Id == "SyncLocalProgress" ? ManagedOperationZoneTypes.Local : ManagedOperationZoneTypes.Global,

 Name = "Синхронный длительный процесс Progress",

 ProcessCode = (operation.Stage1),

 TotalStep = operation.Count,

 **TraceLevel = TraceLevel.Off,**

 };

 Если уровень задан явно при создании управляемой операции, то именно он, а не значение из .config файла будет использован при логировании процесса.

# Возможности администратора

### Добавление списка операций

Для просмотра и управления операциями используется список **Управляемые операции** (ManagedOperationStorage\_ListView) (путь по умолчанию в [приложении XAS](https://by01-tfs05.topsoft.local/sites/Ranet/Xafari/Shared%20Documents/x09/Documentation/XAS.Administrators%20Guide.ru.docx) **Управляемые операции**-> **Управляемые операции**, в приложении используется узел NavigationItems\Items\Configuration\Items\Managed operations\ Items\Managed operations). Если возникла необходимость в данном меню, но оно не было добавлено в приложение XAS или в приложение на этапе разработки, то необходимо добавить в панель навигации список операций. Для этого надо запустить редактор модели из приложения. Далее выбрать пункт NavigationItems и добавить новый. Для созданного элемента выставить значение View = ManagedOperationStorage\_ListView(Рис. 10):



Рис.10 – Добавление списка операций в меню навигации

### Панель действий

В приложении добавится списковая форма с управляемыми операциями. На данной форме будут доступны следующие элементы (Рис. 11):



Рис.11 – Элементы управления операциями

* Кнопка «Приостановить» – ставит управляемую операцию «на паузу».
* Кнопка «Возобновить» – продолжает операцию, если она была приостановлена.
* Кнопка «Остановить» – останавливает выбранную операцию.
* Кнопка «Проверить» – если операция была остановлена либо каким-то причинам зависла, то через 45 секунд она становится просроченной. Данная кнопка поможет выявить просроченные операции.
* Интервал автообновления – здесь задается интервал, с которым будет обновляться списковая форма управляемых операций. Обновление необходимо для своевременных отображения изменений (прогресса, изменений статусов операций и пр.). Автообновление можно отключить.



* Переключатель зоны. При смене значения показывает список либо глобальных либо локальных операций соответственно.



### Вид операций на платформе Web

На рисунке 12 показана детальная форма управляемой операции с подпроцессами:



Рис.12 – Детальная форма операции

На рисунке 13 показаны подпроцессы управляемой операции:



Рис.13 – Вид подпроцессов

Пока в дереве операций не выбран узел с соответствующей операцией, кнопки управления недоступны. Для управления операцией необходимо ее выделить, кликнув на чекбоксе данной операции в первой колонке.

