

Задание-сценарий

Элемент процесса [Задание-сценарий]

Версия 8.0



Эта документация предоставляется с ограничениями на использование и защищена законами об интеллектуальной собственности. За исключением случаев, прямо разрешенных в вашем лицензионном соглашении или разрешенных законом, вы не можете использовать, копировать, воспроизводить, переводить, транслировать, изменять, лицензировать, передавать, распространять, демонстрировать, выполнять, публиковать или отображать любую часть в любой форме или посредством любые значения. Обратный инжиниринг, дизассемблирование или декомпиляция этой документации, если это не требуется по закону для взаимодействия, запрещены.

Информация, содержащаяся в данном документе, может быть изменена без предварительного уведомления и не может гарантировать отсутствие ошибок. Если вы обнаружите какие-либо ошибки, сообщите нам о них в письменной форме.

Содержание

Элемент процесса [Задание-сценарий]	4
Методы Get и Set	5
Соответствие типов параметров Creatio и C#	6
Примеры работы с различными типами параметров	7

Элемент процесса [Задание-сценарий]

ПРОДУКТЫ: **ВСЕ ПРОДУКТЫ**

Элемент процесса [*Задание-сценарий*] является системным действием, которое выполняет программный код C# сценария и обеспечивает его взаимодействие с другими элементами и данными бизнес-процесса. При помощи элемента реализуется расширенная логика, не предусмотренная базовыми элементами Creatio, например:

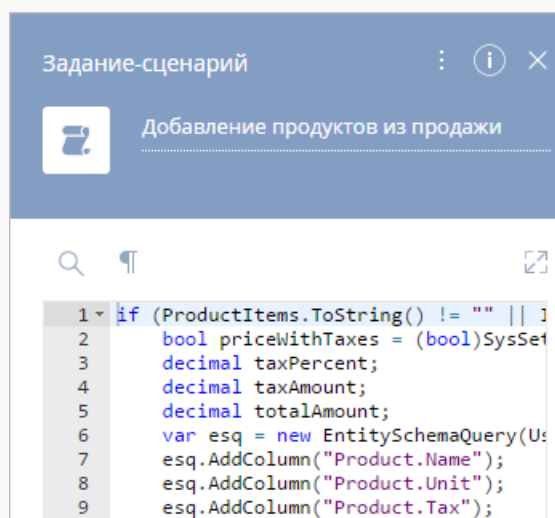
- выполнение сложных математических расчетов;
- выполнение массовой обработки записей;
- выполнение циклов;
- выполнение задач, которые затруднительно реализовать с помощью других элементов системы;
- замена нескольких последовательных элементов [*Формула*].


Работая с элементом [*Задание-сценарий*], придерживайтесь следующих рекомендаций.

- Соблюдайте структуру кода. После сохранения протестируйте работу элемента, выполнив процесс. Корректность и скорость отработки элемента зависят от качества кода и квалификации разработчика.
- Выделяйте комментариями основные блоки и задачи, выполняемые в задании-сценарии, для пояснения функций и предназначения той или иной части кода, ее влияния в процессе. Это позволит пользователю лучше понимать, что происходит в процессе, а разработчику комментарии помогут быстрее разобраться в ранее созданном процессе.
- При использовании сторонних схем указывайте точные названия таких схем. Руководствуйтесь данными рекомендациями при написании сторонних схем.
- Избегайте временных решений, поскольку нестандартный код ухудшает возможности поддержки кода. При выявлении ошибок в процессах контролируйте корректное исправление ошибок.

Чтобы редактировать код сценария, дважды щелкните по элементу на диаграмме. На панели настройки элемента откроется окно для ввода и редактирования программного кода (Рис. 1).


Рис. 1 — Вкладка для редактирования кода задания-сценария



 — развернуть окно для ввода программного кода.

 — свернуть окно для ввода программного кода.

 — поиск в тексте программного кода.

 — отображение скрытых символов (например, пробелов, табуляций) в тексте кода.

На заметку. Платформа Creatio позволяет выполнять процессы без необходимости их публикации. Однако использование методов и элементов [*Задание-сценарий*] требует публикации схемы. Подробнее о вызове методов процесса читайте в статьях: [“Настроить интеграцию с веб-сервисом SOAP”](#) и [“Настроить интеграцию с веб-сервисом REST”](#).

Методы Get и Set

Все создаваемые бизнес-процессы в Creatio являются **интерпретируемыми**. Для обращения к значениям параметра процесса следует использовать методы Get и Set (Рис. 2).

Метод **Get** возвращает значение параметра элемента или процесса.

Сигнатура метода:

```
Get<T>(string path)
```

T — тип значения параметра;

path — строка, определяющая путь к параметру или свойству. Путь формируется согласно правилам:

- “имя параметра”,
- “имя свойства”,
- “имя элемента.имя параметра”,
- “имя элемента.имя свойства”.

Метод **Set** указывает значение параметру элемента или процесса.

Сигнатура метода:

```
Set(string path, T value)
```

value — указываемое значение,

path — строка, определяющая путь к параметру или свойству. Путь формируется согласно правилам, описанным выше для метода Get.

"T" — универсальный параметр, который принимает значение типа соответствующего параметра в терминах C#. Соответствие типов параметров для Creatio и C# приведено в таблице ниже.

Соответствие типов параметров Creatio и C#

Тип параметра Creatio	Тип параметра C#
Целое число	int
Дробное число (0.00000001)	decimal
Дробное число (0.0001)	
Дробное число (0.001)	
Дробное число (0.01)	
Дробное число (0.1)	
Деньги	
Дата/Время	DateTime
Дата	
Время	
Уникальный идентификатор	Guid
Справочник	
Логическое	bool
Строка (50 символов)	string
Строка (250 символов)	
Строка (500 символов)	
Строка неограниченной длины	
Не локализуемая строка	
Коллекция значений	ICollection и любые классы, которые реализуют эти интерфейсы
Коллекция записей	ICompositeObjectList<ICompositeObject> и любые классы, которые реализуют эти интерфейсы

Примеры работы с различными типами параметров

Параметр типа "Целое число"

```
int integerValue = Get<int>("IntegerParameter");
integerValue += 5;
Set<int>("IntegerParameter", integerValue);
```

Параметры типа "Деньги" и "Дробное число" любой точности

```
decimal decimalValue = Get<decimal>("DecimalParameter");
decimalValue += 5.5m;
Set<decimal>("DecimalParameter", decimalValue);
```

Параметр типа "Уникальный идентификатор"

```
Guid uniqueIdentifierValue = Get<Guid>("UniqueIdentifierParameter");
if (uniqueIdentifierValue != Guid.Empty) {
    uniqueIdentifierValue = Guid.Empty;
    Set<Guid>("UniqueIdentifierParameter", uniqueIdentifierValue);
}
```

Параметры типов "Дата", "Время" и "Дата/Время"

```
DateTime dateTimeValue = Get<DateTime>("DateTimeParameter");
dateTimeValue = dateTimeValue.AddDays(1);
Set<DateTime>("DateTimeParameter", dateTimeValue);
```

Параметр типа "Справочник"

```
Guid lookupValue = Get<Guid>("LookupParameter");
if (lookupValue.IsEmpty()) {
    lookupValue = (Guid)UserConnection.SystemValueManager.GetValue(UserConnection, "CurrentUserC
    Set<Guid>("LookupParameter", lookupValue);
}
```

Параметры типа "Не локализуемая строка" и "Строка" любой длины

```
string textValue = Get<string>("TextParameter");
textValue += " and something else";
Set<string>("TextParameter", textValue);
```


Параметры для локализуемых строк

```

LocalizableString localizableStringValue = Get<LocalizableString>("LocalizableStringParameter");
CultureInfo cultureRu = CultureInfo.GetCultureInfo("ru-RU");
CultureInfo cultureEn = CultureInfo.GetCultureInfo("en-US");
localizableStringValue.SetCultureValue(cultureRu, "Здравствуйте!");
localizableStringValue.SetCultureValue(cultureEn, "Hello!");
Set<LocalizableString>("LocalizableStringParameter", localizableStringValue);

```

Параметр типа "Логический"

```

bool booleanValue = Get<bool>("BooleanParameter");
booleanValue = !booleanValue;
Set<bool>("BooleanParameter", booleanValue);

```

Параметр типа "Коллекция значений"

```

ObjectList<int> numbers = ObjectList.Create(1, 2, 3, 4);
Set<ObjectList<int>>("IntegerValuesParameter", numbers);
var items = Get<ObjectList<int>>("IntegerValuesParameter");
items.Add(5);
Set<ObjectList<int>>("IntegerValuesParameter", items);

ObjectList<bool> booleanValues = ObjectList.Create(false, true, true, false);
Set<ObjectList<bool>>("BooleanValuesParameter", booleanValues);
booleanValues = Get<ObjectList<bool>>("BooleanValuesParameter");
if (booleanValues.Count == 4) {
    booleanValues.Clear();
}
Set<ObjectList<bool>>("BooleanValuesParameter", booleanValues);
ObjectList<DateTime> dateTimeValues = ObjectList.Create(new DateTime(2020, 08, 03, 13, 15, 14),
Set<ObjectList<DateTime>>("DateTimeValuesParameter", dateTimeValues);

ObjectList<Guid> guidValues = ObjectList.Create(Guid.NewGuid(), Guid.NewGuid());
Set<ObjectList<Guid>>("GuidValuesParameter", guidValues);
guidValues = Get<ObjectList<Guid>>("GuidValuesParameter");
if (!guidValues.Contains(Guid.Empty)) {
    guidValues.Add(Guid.Empty);
}
Set<ObjectList<Guid>>("GuidValuesParameter", guidValues);

ObjectList<decimal> decimalValues = ObjectList.Create(3.14m, 432434.00032m);

```

```

Set<ObjectList<decimal>>("DecimalValuesParameter", decimalValues);
decimalValues = Get<ObjectList<decimal>>("DecimalValuesParameter");
decimalValues.RemoveAt(1);
Set<ObjectList<decimal>>("DecimalValuesParameter", decimalValues);

ObjectList<string> stringValues = ObjectList.Create("string value 1", "string value 2");
Set<ObjectList<string>>("StringValuesParameter", stringValues);
stringValues = Get<ObjectList<string>>("StringValuesParameter");
stringValues.Remove("string value 1");
Set<ObjectList<string>>("StringValuesParameter", stringValues);

```

Параметр типа "Коллекция записей"

```

var list = Get<ICompositeObjectList<ICompositeObject>>("ReadDataUserTask1.ResultCompositeObjectList");
var sb = new StringBuilder();
foreach (ICompositeObject item in list) {
    if (item.TryGetValue<string>("Name", out string value)) {
        sb.Append(value).Append( " | ");
    }
}
Set<string>("FieldsOfCompositeObjectListParameter", sb.ToString());

```

Параметр типа "Коллекция записей", создающий новый перечень записей

```

var list = new CompositeObjectList<CompositeObject>();
var item1 = new CompositeObject();
item1["Id"] = Guid.NewGuid();
item1["Name"] = "Name1";
list.Add(item1);
var item2 = new CompositeObject();
item2["Id"] = Guid.NewGuid();
item2["Name"] = "Name2";
list.Add(item2);
Set<CompositeObjectList<CompositeObject>>("CompositeObjectListParameter", list);

```