Article
Michael Braam · Nov 24, 2016 7m read

Building and using Plugins in DeepSee

Introduction

A calculated measure is a powerful feature in DeepSee and can help to enrich your analyzes. In the case of complex or long running computations plugins can be useful. This article shows with a simple example how you can build and use a plugin in DeepSee.

The goal of this article is to provide a short introduction to Plugins in DeepSee. It also covers the questions "**What are use-cases for plugins**" and "**What are the building blocks of a Plugin**".

When you read this article you will learn how to build your own plugin.

Before you read you should be familiar with Caché Objects and the basic tools and concepts of DeepSee (i.e. Analyzer, Pivot table etc.).

The sample provided in this article is based on Caché/Ensemble 2016.2.1, but Plugins were introduced a while back.

Our sample plugin operates on the ***HoleFoods*** cube in the ***SAMPLES*** namespace.

We use Atelier in our development, but you can use Studio as well.

Use-cases

Plugins are useful if you have complex and long running computations. One of the characteristics of a plugin is its asynchronous execution. This means that within a pivot table the other content can already be displayed while the plugin is still executed asynchronously. If the result for a plugin based cell is available, the pivot table content will be updated accordingly.

In addition to that a plugin has access to the lowest data level.

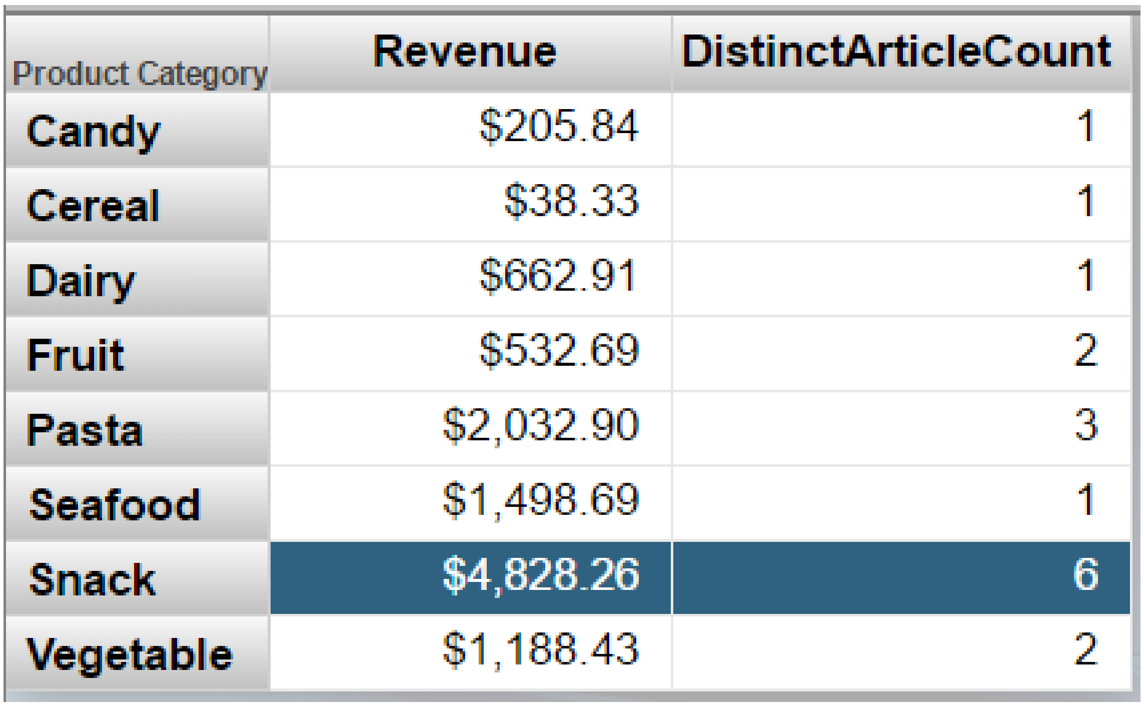
From a technical perspective Plugins are similar to **KPIs** (Key Performance Indicator) in DeepSee.

How to build a sample Plugin

You can have different types of plugins

- The plugin can be visible and draggable in the Analyzer or can be used in Dashboard widgets
- The plugin can be used only with the ***%KPI-function***
- The plugin can be cube-specific or can be usable with multiple cubes

In this article a ***DISTINCT*** Plugin is implemented. It operates on the HoleFoods cube and can be used to calculate the distinct articles for a particular row in the pivot table. See a sample pivot table below



Product Category	Revenue	DistinctArticleCount
Candy	$205.84	1
Cereal	$38.33	1
Dairy	$662.91	1
Fruit	$532.69	2
Pasta	$2,032.90	3
Seafood	$1,498.69	1
Snack	$4,828.26	6
Vegetable	$1,188.43	2

The above pivot table shows that the revenue for Snacks ($4,828.26) is based on six distinct products in the snack category

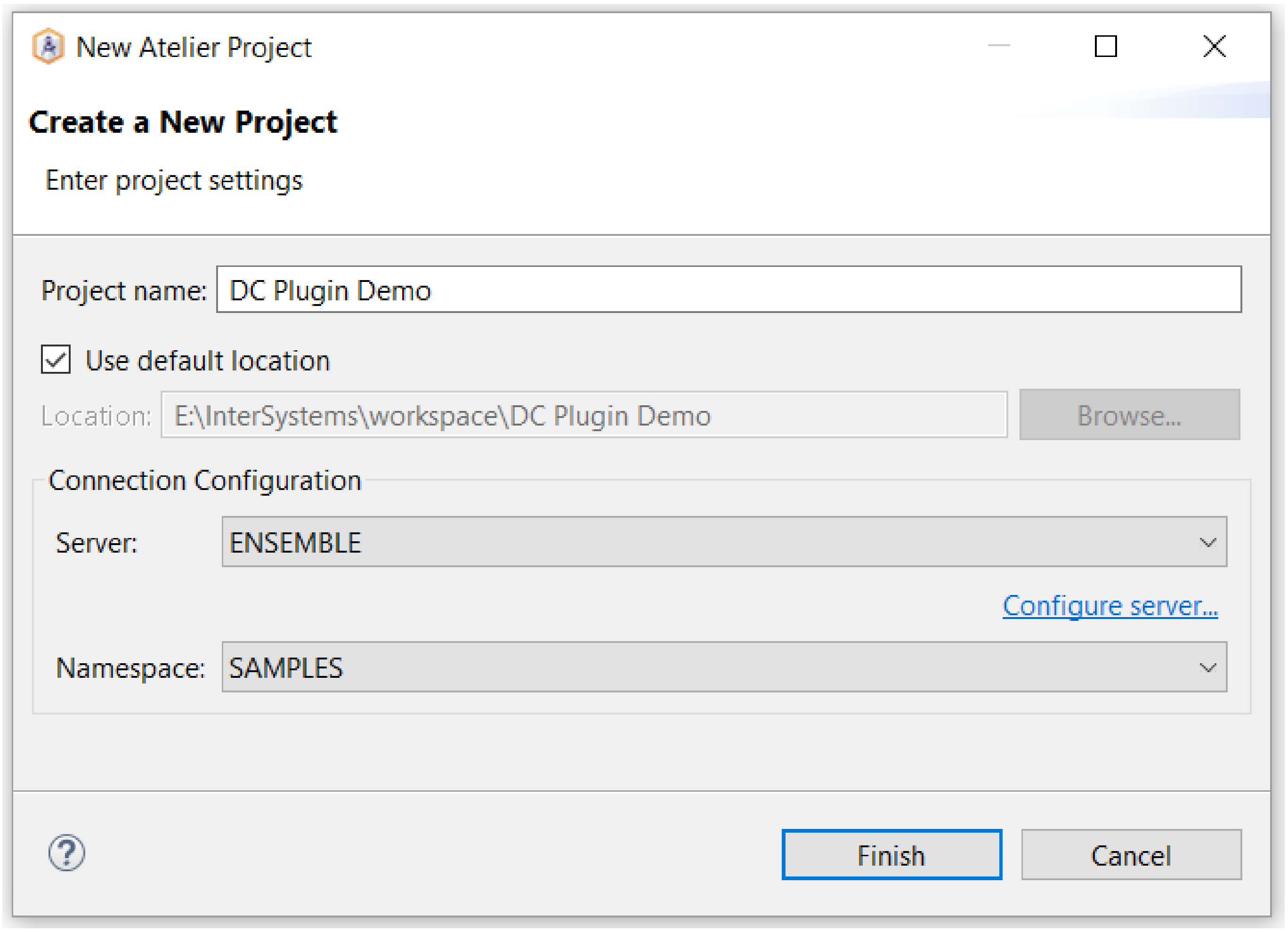
DeepSee contains a more generic version of the DISTINCT plugin ***%DeepSee.Plugin.Distinct***. But this sample here is chosen to demonstrate the necessary basic steps to implement your own plugins.

Creating a plugin

A plugin is a class that inherits from ***%DeepSee.KPIPlugIn***. To build your own plugin you have to create a subclass of ***%DeepSee.KPIPlugIn***, overwrite some parameters and implement one or more methods.

In the following section you learn how to do that with Atelier. Our plugin will be available in the Analyzer and usable in dashboard widgets.

- Please launch Atelier and create a new project (***File->New->Atelier Project***)
- Assign a name, a server connection and a namespace to your new project
- Click ***Finish***


New Atelier Project

Create a New Project

Enter project settings

Project name: DC Plugin Demo

☑ Use default location

Location: E:\InterSystems\workspace\DC Plugin Demo

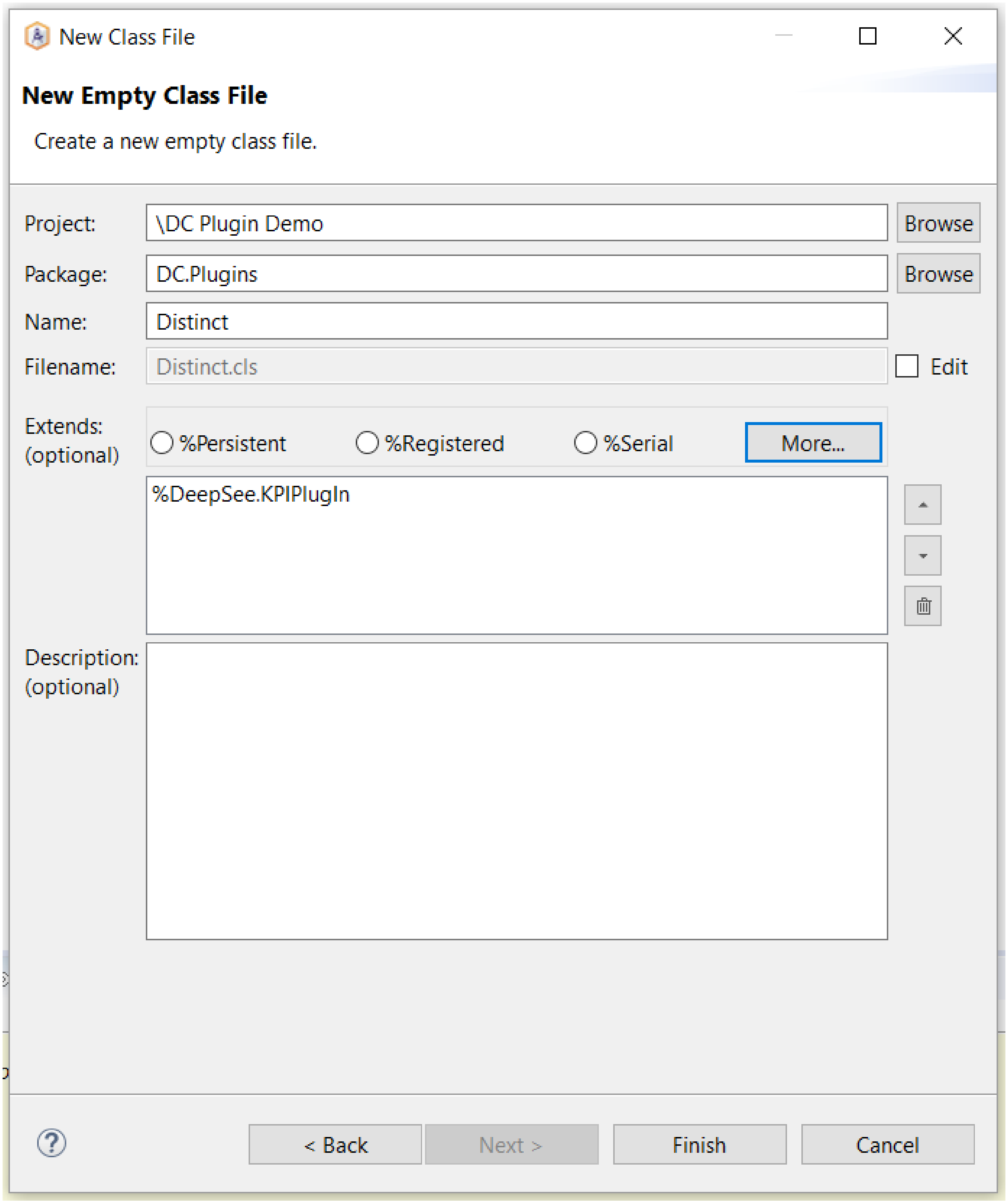
Connection Configuration

Server: ENSEMBLE

Namespace: SAMPLES

- Next create a new class (***File->New->Class File***)

- In the upcoming wizard select ***Empty Class*** and click ***Next***
- Next select your project for the newly created class
- Assign a package or a package hierarchy to it and give it a classname
- In the row ***Extends*** click ***More*** and select ***%DeepSee.KPIPlugIn***
- Click ***Finish***



New Empty Class File

Create a new empty class file.

Project:	\DC Plugin Demo	Browse
Package:	DC.Plugins	Browse
Name:	Distinct	
Filename:	Distinct.cls	☐ Edit
Extends: (optional)	○ %Persistent ○ %Registered ○ %Serial More...	
	%DeepSee.KPIPlugIn	
Description: (optional)		

As said before our plugin works with the HoleFoods cube and it should be accessible in the Analyzer. We will specify that in our class now.

- Create a parameter ***BASECUBE***. This parameter determines whether a plugin is only implemented for a particular cube or subject area or is supposed to work for multiple cubes/subject areas. In our case we simply set this parameter to ***HoleFoods***. If you want to specify multiple cubes/subject areas either use a comma-separated list of names or use wildcards "*"

- `Parameter BASECUBE = "HoleFoods";`

- Next we make sure that our plugin is accessible in the Analyzer. To achieve that we have to set two parameters ***PLUGINTYPE*** and ***PUBLIC***. ***PLUGINTYPE*** must be set to "**pivot**" and ***PUBLIC*** must be set to "**1**". If you want to prevent the access in Analyzer set ***PLUGINTYPE*** to "**aggregate**" and/or ***PUBLIC*** to "**0**".

```
?Parameter PLUGINTYPE = "pivot";
Parameter PUBLIC = 1;
```

- We have to specify two more parameters. ***LISTINGSOURCE*** and ***LISTINGFIELDS***. The allowed values for ***LISTINGSOURCE*** are "**SourceTable**" (default) or "**FactTable**". "**SourceTable**" means that the plugin queries the source table of the cube. "**FactTable**" means that the datasource for that plugin is the facttable of the cube. ***LISTINGFIELDS*** specifies the fields from the listing source that are part of the result set. In our example we choose "**FactTable**" and "**DxProduct**".

```
Parameter LISTINGSOURCE = "FactTable";

Parameter LISTINGFIELDS = "DxProduct";
```

Alternatively you can implement the callback-method ***%OnGetListingFields()***. This method returns the listing-fields as a comma separated list.

```
Method %OnGetListingFields() As %String
```

A plugin can have one or more properties. Each of these properties has a value. These properties are used in MDX or in pivot tables or widgets. The properties of the plugin are specified in a XData block with name ***KPI***.

- Add the following XData block to your plugin class

```
XData KPI [ XMLNamespace = "http://www.intersystems.com/kpi" ]
{
<kpi name="DCDemoPlugin" displayName="DCDemoPlugin" caption="DCDemoPlugin">
<property name="DistinctArticleCount" displayName="DistinctArticleCount"/>
</kpi>
}
```

The above XData block specifies the plugin ***DCDemoPlugin***. This plugin provides one property ***DistinctArticleCount***.

All we need to do to make this plugin work, is to implement the computation code. This is done in the method ***%OnCompute()***. For each cell in the pivot table where the plugin is used, the listing query is executed and the ***%OnCompute*** method is invoked. It receives the result set of the listing query as a input parameter. It also receives the count of the underlying facts for that pivot table cell. See below the signature of ***%OnCompute***.

```
 Method %OnCompute(pSQLRS As %SQL.StatementResult,pFactCount As %Integer) As %Status
```

Within your method code you can iterate through the result set. Each row in the result set represents one row in the facttable or sourcetable depending on your setting in ***LISTINGSOURCE***. The available columns in the result set are the columns you have specified in the ***LISTINGFIELDS*** parameter above. In our case we want to calculate the distinct article/product count for each cell. A simple code for this looks like this:

```
Method %OnCompute(
    pSQLRS As %SQL.StatementResult,
    pFactCount As %Integer) As %Status
{
    #dim tCounter as %Integer = 0
    #dim tProduct as %String
    #dim tSc as %Status = $$$OK

    try {
        set ..%seriesCount = 1
        set ..%seriesNames(1) = "DCDemoPlugin"

        while pSQLRS.%Next() {
            set tCounter = tCounter + 1
            if '$data(tmpProduct(pSQLRS.DxProduct)) set tmpProduct(pSQLRS.DxProduct)
= ""
            if (tCounter#10 = 0) {
                do ..%SetPercentComplete(100*(tCounter/pFactCount))
                }
            }

        set tCounter = 0
        set tProduct = $order(tmpProduct(""))
        while tProduct '= "" {
            set tCounter = tCounter + 1
            set tProduct = $order(tmpProduct(tProduct))
        }

        set ..%data(1,"DistinctArticleCount") = tCounter
        }
    catch tEx {
        set tSc = tEx.AsStatus()
    }

    return tSc
}
```

Let's take a closer look to the code

```
set ..%seriesCount = 1

set ..%seriesNames(1) = "DCDemoPlugin"

...

set ..%data(1,"DistinctArticleCount") = tCounter
```

- ***%seriesCount*** defines the number of series this plugin provides. It is recommended to have only one for a plugin
- ***%seriesNames*** is a multidimensional property which defines the names of the series
- ***%data*** is a multidimensional property of the plugin instance which is used to assign values to the plugin properties ***%data*** is indexed by the series' number, followed by the property name

To indicate the state of completion (remember plugins are executed asynchronously) the method ***%SetPercentComplete*** is used:

```
do ..%SetPercentComplete(100*(tCounter/pFactCount))
```

Below you find the complete code for the plugin class

```
Class DC.Plugins.Distinct extends %DeepSee.KPIPlugIn {

Parameter BASECUBE = "HoleFoods";

Parameter PLUGINTYPE = "pivot";

Parameter PUBLIC = 1;

Parameter LISTINGSOURCE = "FactTable";

Parameter LISTINGFIELDS = "DxProduct";



XData KPI [ XMLNamespace = "http://www.intersystems.com/kpi" ]
{
&lt;kpi name="DCDemoPlugin" displayName="DCDemoPlugin" caption="DCDemoPlugin"&gt;
&lt;property name="DistinctArticleCount" displayName="DistinctArticleCount"/&gt;
&lt;/kpi&gt;
}



Method %OnCompute(
    pSQLRS As %SQL.StatementResult,
    pFactCount As %Integer) As %Status
{
    #dim tCounter as %Integer = 0
    #dim tProduct as %String
    #dim tSc as %Status = $$$OK

    try {
        set ..%seriesCount = 1
        set ..%seriesNames(1) = "DCDemoPlugin"

        while pSQLRS.%Next() {
            set tCounter = tCounter + 1
            if '$data(tmpProduct(pSQLRS.DxProduct)) set tmpProduct(pSQLRS.DxProduct)
= ""
            if (tCounter#10 = 0) {
                do ..%SetPercentComplete(100*(tCounter/pFactCount))
                }
            }
        set tCounter = 0
        set tProduct = $order(tmpProduct(""))
        while tProduct '= "" {
            set tCounter = tCounter + 1
            set tProduct = $order(tmpProduct(tProduct))
        }

        set ..%data(1,"DistinctArticleCount") = tCounter
        }
    catch tEx {
        set tSc = tEx.AsStatus()
    }

    return tSc
}



}
```

#Analyzer #Dashboards #MDX #InterSystems IRIS BI (DeepSee)

Source URL:https://community.intersystems.com/post/building-and-using-plugins-deepsee