

Creating a Spotfire (2.0-2.1) Extension

A Spotfire extension is the smallest functional unit added to the platform. It is developed in Visual Studio® and is included in a Spotfire AddIn, enabling versioning, licensing, deployment and loading.

Overview

Required software: Spotfire SDK

The SDK provides required Spotfire developer resources: the Spotfire Extension Project Template, development assemblies, example projects, and the Package Builder application wrapping the extensions for deployment.

- **Spotfire Extension Project Template**
The Spotfire Extension Project Template is used in Visual Studio® to create new Spotfire extension projects. It takes care of implementation details and enables you to focus on adding new functionality.
- **Running the Spotfire SDK Examples**
The Spotfire SDK provides a set of examples. These are fully functional and correspond to common use cases, but are still simple enough to constitute a source of insight as well as code.

To create Spotfire extensions, use the *Spotfire Extension Template* and reuse code from the *SDK examples*. The Spotfire extension concepts and the recommended procedure to create an extension are described below.

Concepts

A Spotfire extension is typically implemented as a C# project in Visual Studio® using the *Spotfire Extension Project Template*. The project contains the following:

- **One or more extensions**
Each extension is implemented in a set of classes, typically overriding a base class defining an specific extension type. Additional extensions can easily be added to the project.
The extensions in a project are bundled for loading into Spotfire by an *AddIn* unit.
- **One class derived from the AddIn class**
The AddIn is defined using information you provide when creating the project from the SDK template. Spotfire, deploys, version handles and loads functionality as AddIns. The *Spotfire Extension Project Template* assumes that only one Add-In implementation is available in your project. This is in accordance with design guidelines for Spotfire Extensions. If you wish to add more than one Add-In to your project, you must alter `modules.xml`, the module definition file.
- **One The Module Definition File (`modules.xml`)**
A module is defined as qualified for loading in Spotfire if it has a valid `modules.xml` file. It declares the Add-In to the application. It defines the fully qualified type name for the AddIn class and the assembly name. It also declares a unique project GUID and a strong name for the assembly. Combinations of these uniquely identifies the project and the AddIn.
If you change the name of your project, assembly, or add-in class, you must also alter the names in the module definition file.
- **Strong name key file (`.snk`)**
Extension assemblies must be signed with a strong name key. The extension project template contains a default key. You may replace this key file with your own, but if you do, make sure to alter the module definition file.

Procedure

1. Download the Spotfire SDK.
2. Unzip the Spotfire SDK into the recommended folder structure: For *SDK v2.1.0*: `TIBCO Spotfire SDK\2.1\SDK` The structure may be placed in any location.

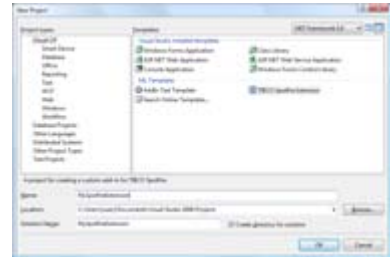
For *Spotfire SDK v2.0.1* and *v2.0.3*: `C:\DxpProjects\SDK`.

The recommended structures ensure that project references work out of the box. Other locations require the references to be redirected.

3. In *Visual Studio®*, create a new project based on the Spotfire Extension Project Template:

Fill out the **Name**, **Location** and **Solution Name** fields, and press **OK**.

Visual Studio® creates a solution for the extension project. In the *Solution Explorer*, expand the **References** folder and verify that the Spotfire SDK assembly references are not broken. To resolve broken references, right-click the project in the *Solution Explorer* and select **Properties**. Then select the *Reference Paths* tab, add the path to the **Binaries** folder of the SDK, and press the **Add Folder** button.



4. Implement your extension.
5. Proceed to Debugging an Extension.

Debugging an Extension

Debugging behavior is defined in Visual Studio and in the Spotfire install directory.

1. To prepare debugging of your extension in Visual Studio®, first right-click the project in the *Solution Explorer*, then select **Properties** and the *Debug* tab, and finally select the **Start external program** radio button, and verify that the path to your Spotfire installation is correct.

To start Spotfire with specific parameters, refer to the [Command Line Parameters](#).

2. To prepare debugging of your extension in Spotfire v2.1, open the `Spotfire.Dxp.exe.config` file in a text editor.

From Spotfire v2.1 it is located in the folder `C:\Program Files\Tibco\Spotfire\[Version]`.

For v2.0 it is located in the folder `C:\Program Files\Spotfire\DXP\2.0`.

Then add a **folder** tag to reference your project debug build:

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <configSections>
    <section name="Spotfire.Dxp" type="Spotfire.Dxp.Starter.DxpStarterSettings,
Spotfire.Dxp"/>
  </configSections>
  <Spotfire.Dxp>
    <modules runLocally="false">
      <folder path="[DRIVE]:\Program Files\Tibco\Spotfire\2.1\Modules"/>
      <folder path="%LOCALAPPDATA%\Tibco\Spotfire\2.1\Modules"/>
      <folder
path="[DRIVE]:\SpotfireProjects\MySpotfireExtension\MySpotfireExtension\bin\Debug"/>
    </modules>
  </Spotfire.Dxp>
</configuration>
```

You may add several **folder** tags to indicate multiple extension directories to be loaded when running Spotfire.

Packaging an Extension

An extension must be packaged before it can be deployed to an Analytics Server.

Overview

When the extension code is finalized, sign it and perform a release build. Then package it using the *Spotfire Package Builder* before it is deployed to the server.

Signing the code

Spotfire extensions must be signed with a strong name key in order to be eligible. The Spotfire Extension Project Template contains a default key used to sign your project during debugging. It is however strongly recommended that you change this key before you build a Release version of your extension and deploy it on a Spotfire Analytics Server.

To change the default key:

1. In the *Solution Explorer* window, right-click your project and select **Properties**.
2. In the *Signing* tab, click the **Choose a strong name key file** drop-down box and either select **<New...>** or **<Browse...>** based on whether or not you have a key.

Note: At this point, you will no longer be able to debug the extension with the default module definition file generated by the project template since the full type name of your assembly and classes have changed. To debug, you must alter the project's `module.xml` file manually.

Building the target

It is highly recommended that you deploy your extension build with the **Release** target in Visual Studio®. To build a release version of your extension:

1. Change the compile target drop-down box in Visual Studio® to **Release**.
2. In the *Build*, select **Rebuild Solution**.

Creating the package

To create a Spotfire package containing your extension:

1. Launch the **Spotfire Package Builder** application.
It is bundled in the `Package Builder` folder of the Spotfire SDK.
2. In the *Information* tab, enter the description of your package.
This includes the name, version, manufacturer, and the description fields.

Set the package version to `1.0.0.0` for the first version of the extension.

3. In the *Files* tab, add the assemblies and/or resources that you wish to bundle with your extension.
4. It is highly recommended that you sign your package:

1. In the *File* menu, open the **Package Signing Options** dialog.
2. Click **Browse...** and select your certificate.
If your certificate contains a private password, you need to define it in the appropriate field.
3. Click **OK** to close the dialog.

5. Save the package.

The **Package ID** is automatically defined.



Evolving or Refactoring an Extension

When creating a new version of an extension that already has been deployed, the local development and the server deployed versions must not collide. This article outlines recommended practices to create a new version while remaining connected to a server where an older version is deployed.

Overview

This article describes how to avoid pitfalls when evolving or refactoring an existing extension. The DOs and DON'Ts primarily aim at keeping the downloaded package and the evolving extension separate.

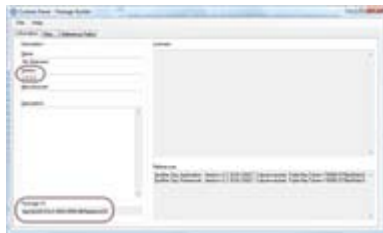
Preconditions

- The Spotfire SDK is installed.
- A Spotfire extension has been developed, packaged, and deployed to an Analytics Server.
- The deployed extension is downloaded and installed in an existing Spotfire instance on the developer's computer prior to any further development to the extension.

Development beyond the First Version

Once the package has been made available by deploying it to an Analytics Server and further development of the extension has started, the following procedures are recommended.

DO



- Update the `id` of the extension in the `module.xml` file.
It must match the `id` of the packaged extension that was deployed on the server.
The `id` of the deployed extension can be obtained from the **Package ID** field in the *Package Builder*.
- Use increments of major version number when stepping version numbers:
If the current version number is `1.0.0.0`, increment to `2.0.0.0`.
- Step the assembly version and file version number of your extension in the `AssemblyInfo.cs` to match the version number in the `module.xml` file.
- Increment the version number in the `module.xml` to a number that is greater than the one of the deployed package containing the same extension.
- Update the `module/assemblies/assembly/fullname` node of your `module.xml` to use the new assembly version number.
- Update the `fullTypeName` attribute of the `module/extension/addIn` node of your development `module.xml` to use the new assembly version number.

DON'T

- NEVER change the `<min-version>` or `<max-version>` values in the development `module.xml` file.