SOFTPRO®
CLOSING AND TITLE SOFTWARE

SoftPro Select Plugins

A Whitepaper on Authoring Extensions to the SoftPro Select platform

Publication date: **May 2019**

Abstract

With the release of SoftPro Select version 4.6.0, the process of authoring and deploying extensions to the SoftPro Select application platform (client and server) has changed dramatically. We acknowledge there are customers who have already created packages for versions of SoftPro Select prior to version 4.6.0, or those who may want to write new custom integrations/extensions to SoftPro Select.

In this whitepaper, we provide a treatise on the concept of a Plugin as applicable to SoftPro Select, and guidance for using the SoftPro Select SDK to create new plugins, upgrade existing custom client-side and server-side integrations (a.k.a. "*Custom Packages*") from versions prior to 4.6.0 and packaging them into plugins for deployment in the SoftPro Select ecosystem of version 4.6.0 and beyond.

Prerequisites

- SoftPro Select client and server installation
- A basic understanding of Microsoft™ Visual C#
- Access to the Visual Studio 2015 Integrated Development Environment
- Access to nuget.org as a configured package source via NuGet Package Manager

Metaphor

CLIENT	See **SHELL**
CLIENT-SIDE	Of or relating to the part of the SoftPro Select application which contains the user interface.
MID-TIER	See **SERVER**
MMC	Microsoft™ Management Console.
PACKAGE	An extension to either the client-side shell application or the server-side application that provides additional functionality, services, jobs, etc. Client-side packages may or may not include user interface elements.
PLUGIN	A bundle that can contain packages, either client, server or both.
SERVER	The extensible part of the application which contains the Windows™ service.
SERVER-SIDE	Of or relating to the part of the SoftPro Select application which contains the Windows™ Service for SoftPro Select. This part of the application does not contain any user interface elements.
SHELL	The extensible part of the application which contains the user interface.
VS	Visual Studio IDE.

Background

Prior to version 4.6.0 of SoftPro Select, development of extensions to the Shell and/or Server required developers to acquire the SoftPro Select SDK through a MSI installer. Development of the extension would result in a package that was bound to the version of the SDK installed. Once the development-test cycle was complete, the package, if it was a server-side package, would be delivered to a system administrator to install on the server via the SoftPro Select configuration MMC snap-in. If the package was a client-side package, it would be delivered to a system administrator to install on the individual user workstations.

This model had several pain-points:

- Changes in major/minor versions of Select required recompilation of the package.
- Recompilation of the package(s) happened after-the-fact. The system had to be upgraded first, then the packages recompiled. This could create a potential outage until the rebuild (and any subsequent errors) were completed.
- Restart of the server (for server-side packages) required a separate application.
- Deploying a client-side package required deployment to individual machines either manually or through system management software.
- Different versions of the SDK could not readily coexist, preventing/hindering development in a heterogeneous environment.

Plugin

The architecture of SoftPro Select version 4.6.0 supports the concept of a Plugin. In essence, a plugin is deployable artifact that may contain custom packages, either client-side, server-side or both. This allows the complete functionality of a feature to be installed as one body of work in the system.

Deployment of individual Shell or Server packages is not supported by SoftPro Select 4.6.0 and beyond.

The features to install, uninstall and upgrade plugins are incorporated into the administrative module (SPAdmin) of SoftPro Select. A full discussion of the underlying architectural changes is outside the scope of this document.

NuGet Integration

The SDK is now distributed as NuGet packages on nuget.org. The following packages are paramount for development of custom packages and plugins:

1. **SoftPro.Select.Shell.Sdk**
   This includes necessary assemblies and tools to develop, build and debug a SoftPro Select Shell package. The package has dependencies on these satellite NuGet packages:
   a. SoftPro.Select.Core
   b. SoftPro.Shell.Core
   c. SoftPro.Select.Controls
2. **SoftPro.Select.Server.Sdk**
   This includes necessary assemblies and tools to develop, build and debug a SoftPro Select Server package.

The package has dependencies on these satellite NuGet packages:

a. SoftPro.Select.Core
b. SoftPro.Server.Core

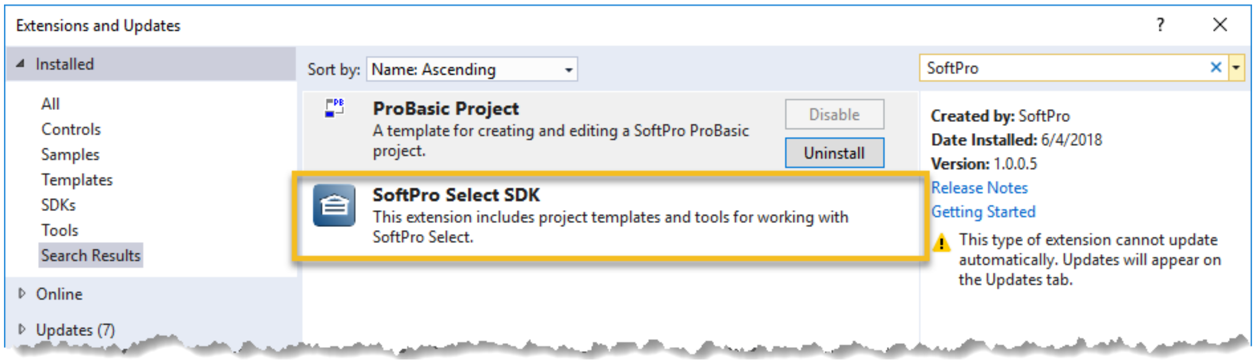
3. **SoftPro.Select.Plugin.Sdk**
This includes necessary tools to define the metadata of a plugin and bundle Shell and/or Server packages into a SoftPro Select plugin.

Distribution of the SDK as NuGet packages offers several advantages:

- Immediate access to prior, current and preview builds of the SDK.
- Administrator privileges are not required to install the SDK.
- Multiple versions can be on the same machine at any one time since they are local to the project.

SoftPro Select SDK Visual Studio Extension

In SoftPro Select 4.6.0 and beyond, developers do not need to run a MSI to install the SoftPro Select SDK. The development toolkit is bundled into a VS extension that is, at the time of publication of this whitepaper, compatible with VS 2015 only. The extension can be installed from the Visual Studio Extensions Gallery.



ProBasic Project
A template for creating and editing a SoftPro ProBasic project.

SoftPro Select SDK
This extension includes project templates and tools for working with SoftPro Select.

Created by: SoftPro
Date Installed: 6/4/2018
Version: 1.0.0.5
Release Notes
Getting Started
This type of extension cannot update automatically. Updates will appear on the Updates tab.

The following development tools and artifacts are packaged into this extension:

1. SoftPro Select SDK NuGet packages listed in the aforementioned section.
2. VS project templates that may be used to create, build and debug Shell, Server and Plugin package projects.
3. The **SoftPro WCF Proxy Generator**, a custom code generator required to auto-generate a WCF service endpoint's client proxy code. This may be required by Shell package code that wants to communicate to a custom WCF endpoint registered by a Server package.

While a developer may be able to create custom package projects without this extension by means of installing relevant SDK NuGet packages into projects, we **strongly recommend** that you install the VS extension for a productive and enhanced development experience.

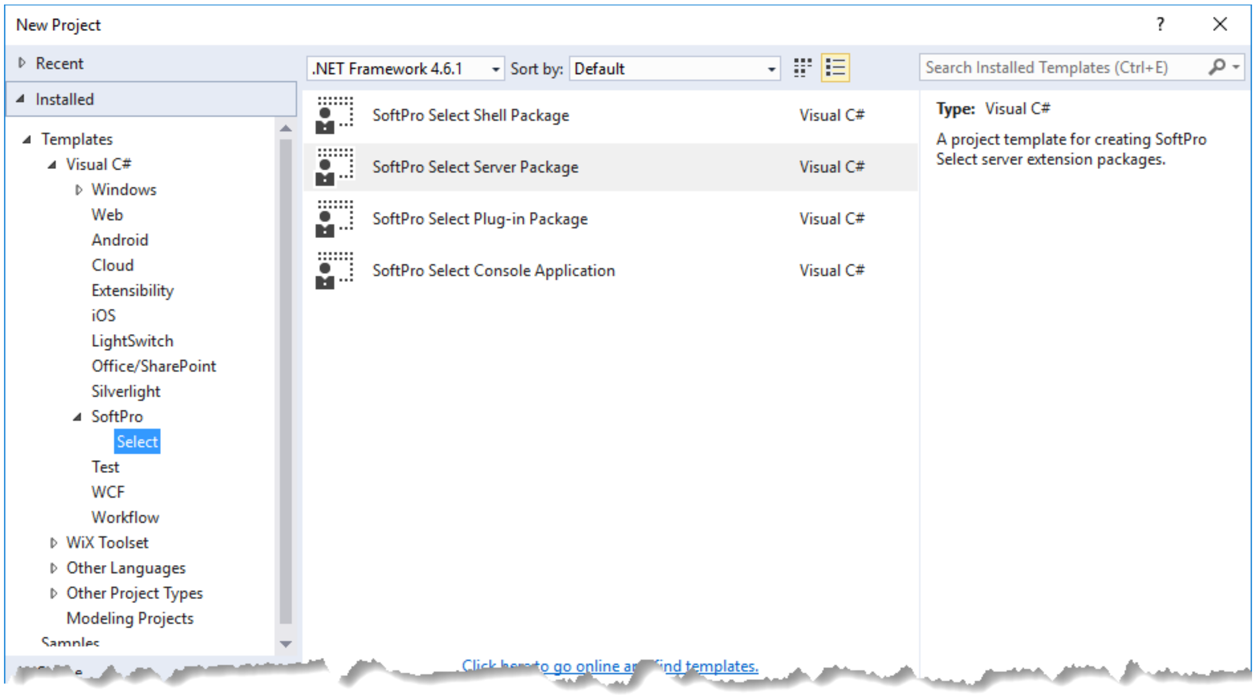
Authoring New Plugins

Prerequisites:

1. Install SoftPro Select client and server on the developer's workstation.
2. Install the SoftPro Select SDK extension through the extensions manager in VS 2015.
   a. Open Visual Studio and select **Tools**, **Extensions and Updates…**
   b. Search for **SoftPro Select SDK** in the **Online** gallery
   c. Click the **Download** button and install the extension.
   d. Restart Visual Studio

SoftPro Select Project Templates

After the SDK extension is installed in Visual Studio, open VS and choose **File**->**New Project**. You will see project templates under **Visual C#**->**SoftPro**->**Select** that you need for development.



New Project

.NET Framework 4.6.1 Sort by: Default

Search Installed Templates (Ctrl+E)

SoftPro Select Shell Package	Visual C#
SoftPro Select Server Package	Visual C#
SoftPro Select Plug-in Package	Visual C#
SoftPro Select Console Application	Visual C#

Type: Visual C#

A project template for creating SoftPro Select server extension packages.

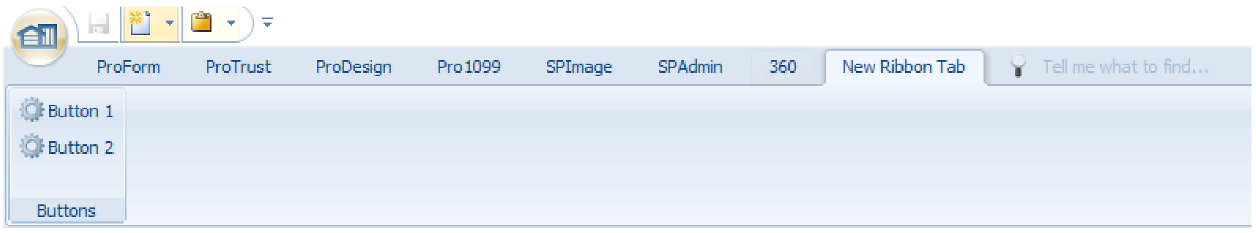
SoftPro Select Shell Package

1. Create a new Shell Package project in Visual Studio by selecting the **SoftPro Select Shell Package** project template**.**

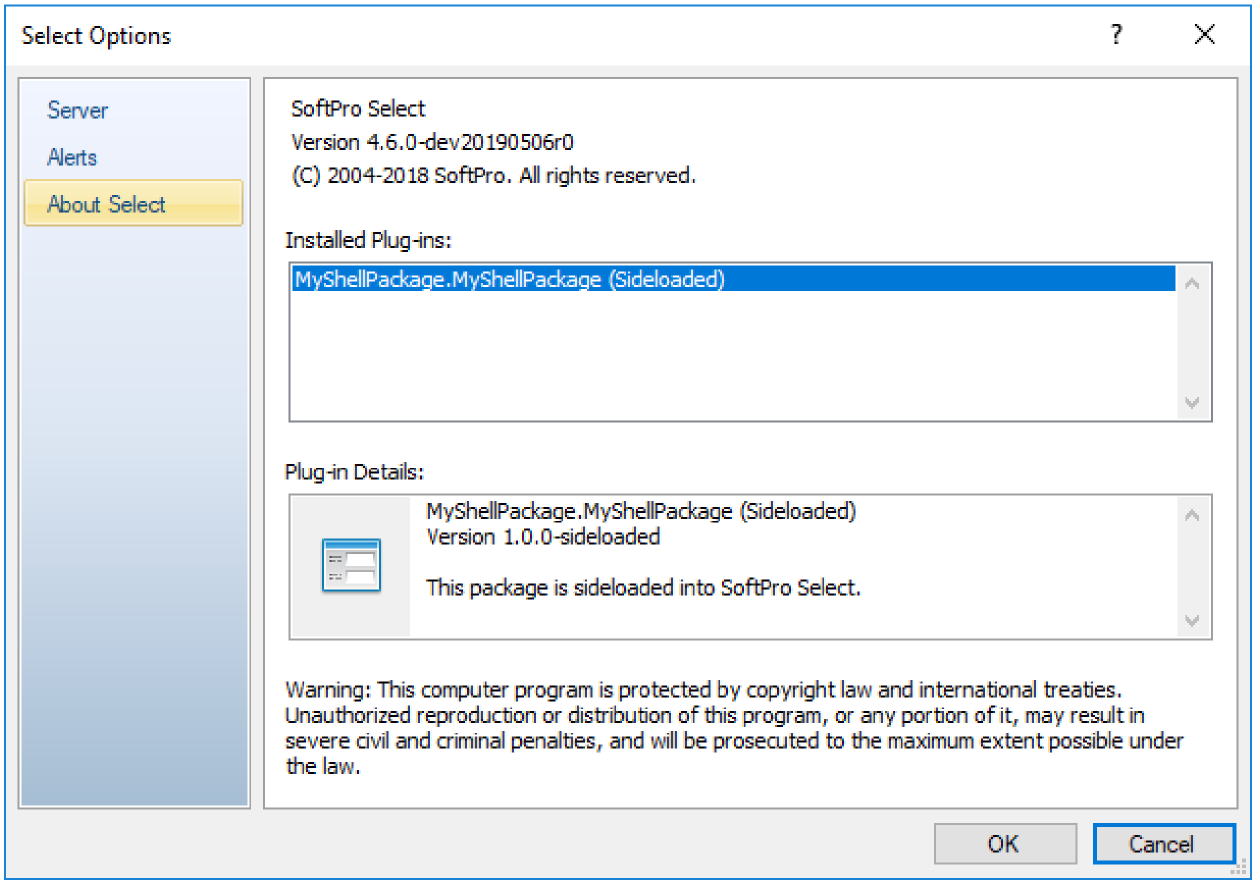
2. Customize the package name and other parameters in the **Shell Package Settings** dialog as required.

3. Build the project and click F5 to debug the package.

4. SoftPro Select client will launch and the sample **New Ribbon Tab** that is included in the project template, will appear in the application.



5. Open the **Select Options** dialog. The Shell package that is being debugged will appear as a *side-loaded plugin* in Select. A side-loaded plugin persists for the lifetime of the debugging session only.



SoftPro Select
Version 4.6.0-dev20190506r0
(C) 2004-2018 SoftPro. All rights reserved.

Installed Plug-ins:

MyShellPackage.MyShellPackage (Sideloaded)

Plug-in Details:

MyShellPackage.MyShellPackage (Sideloaded)
Version 1.0.0-sideloaded

This package is sideloaded into SoftPro Select.

Warning: This computer program is protected by copyright law and international treaties. Unauthorized reproduction or distribution of this program, or any portion of it, may result in severe civil and criminal penalties, and will be prosecuted to the maximum extent possible under the law.

SoftPro Select Server Package

1. Create a new Server Package project in Visual Studio by selecting the **SoftPro Select Server Package** project template**.**

2. Customize the package name and other parameters in the **Server Package Settings** dialog as required.

3. Stop the running instance of SoftPro Select Server via the SoftPro Select Services MMC snap-in.

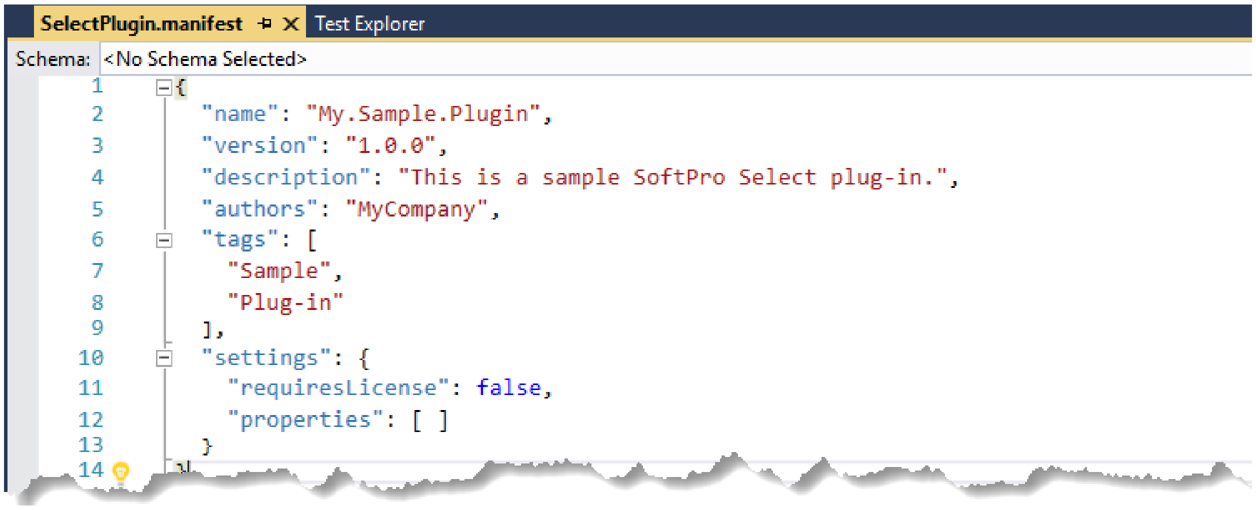
4. Build the project and click F5 to debug the package.

- You will need to run Visual Studio as an **Administrator** to debug Server package projects.
- It is **highly recommended to build strongly-signed Shell and Server package assemblies** before they are packaged into a plugin.

SoftPro Select Plugin

1. Choose the option to **Add**->**New Project** to the solution that contains the Shell/Server package and select the **SoftPro Select Plug-in Package** project template.

2. Open **SelectPlugin.manifest** and modify the plugin's metadata as required. The name of the plugin that contains the Shell/Server packages must be unique and follow naming conventions of NuGet packages.

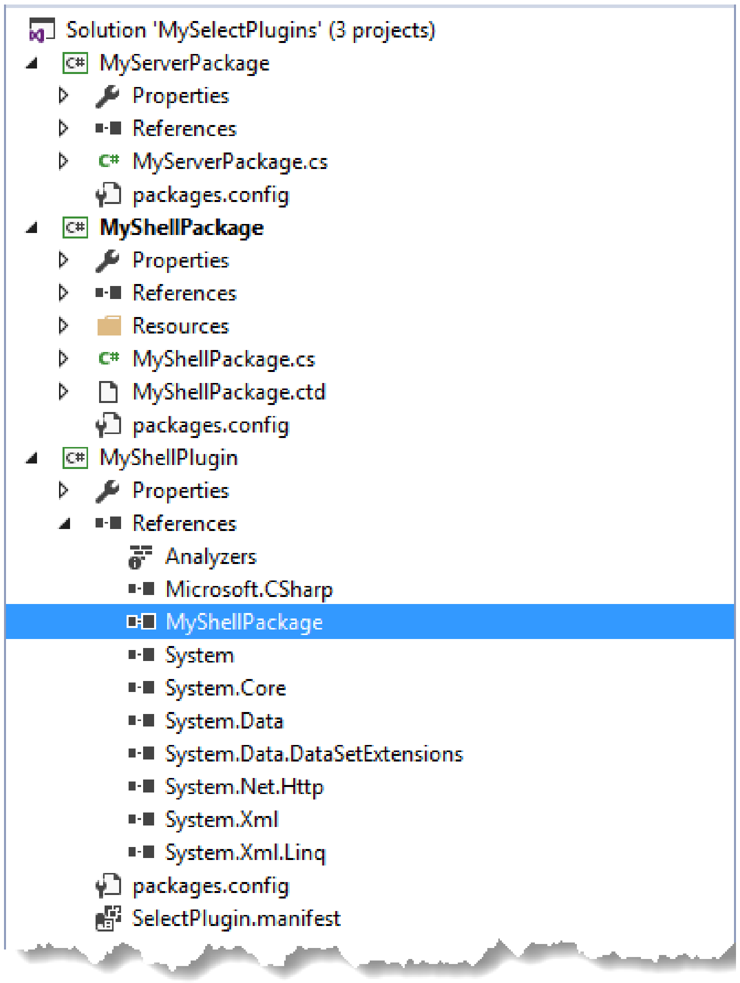


```
{
  "name": "My.Sample.Plugin",
  "version": "1.0.0",
  "description": "This is a sample SoftPro Select plug-in.",
  "authors": "MyCompany",
  "tags": [
    "Sample",
    "Plug-in"
  ],
  "settings": {
    "requiresLicense": false,
    "properties": [ ]
  }
```

3. If the plugin requires a license, set the "**requiresLicense**" flag to true. See Addendum for details on plugin settings.

4. You may also assign configuration settings to the plugin. See Addendum for a thorough discourse on specifying a plugin's settings.

5. If you want this plugin to encapsulate the Shell package project that's in the solution, add a project reference to the Shell package project.



```
Solution 'MySelectPlugins' (3 projects)
  MyServerPackage
    Properties
    References
    MyServerPackage.cs
    packages.config
  MyShellPackage
    Properties
    References
    Resources
    MyShellPackage.cs
    MyShellPackage.ctd
    packages.config
  MyShellPlugin
    Properties
    References
      Analyzers
      Microsoft.CSharp
      MyShellPackage
      System
      System.Core
      System.Data
      System.Data.DataSetExtensions
      System.Net.Http
      System.Xml
      System.Xml.Linq
    packages.config
    SelectPlugin.manifest
```

Note: You may add one or more Shell and/or Server package project references to a plugin project depending on how you want to bundle packages into the resulting plugin.

6. Save your work and do **Rebuild All** of the plugin project.

7. Assuming all metadata specified in the manifest is valid, the project will build a ***<plugin_name>.nupkg*** file in the output path, where *plugin_name* is the name specified in the plugin's manifest file.

8. The Shell and/or Server packages are bundled into the plugin file as one deployable unit. This plugin can now be installed via SPAdmin Plug-ins Manager in the SoftPro Select Client application. Please refer to SoftPro Select Release Notes for information on the installation process.

Upgrading Existing Packages

Prerequisites:

1. Install SoftPro Select client and server on the developer's workstation.
2. Install the SoftPro Select SDK extension through the extensions manager in VS 2015.
   a. Open Visual Studio and select **Tools**, **Extensions and Updates…**
   b. Search for **SoftPro Select SDK** in the **Online** gallery.
   c. Click the **Download** button and install the extension.
   d. Restart Visual Studio.

Select Shell Package

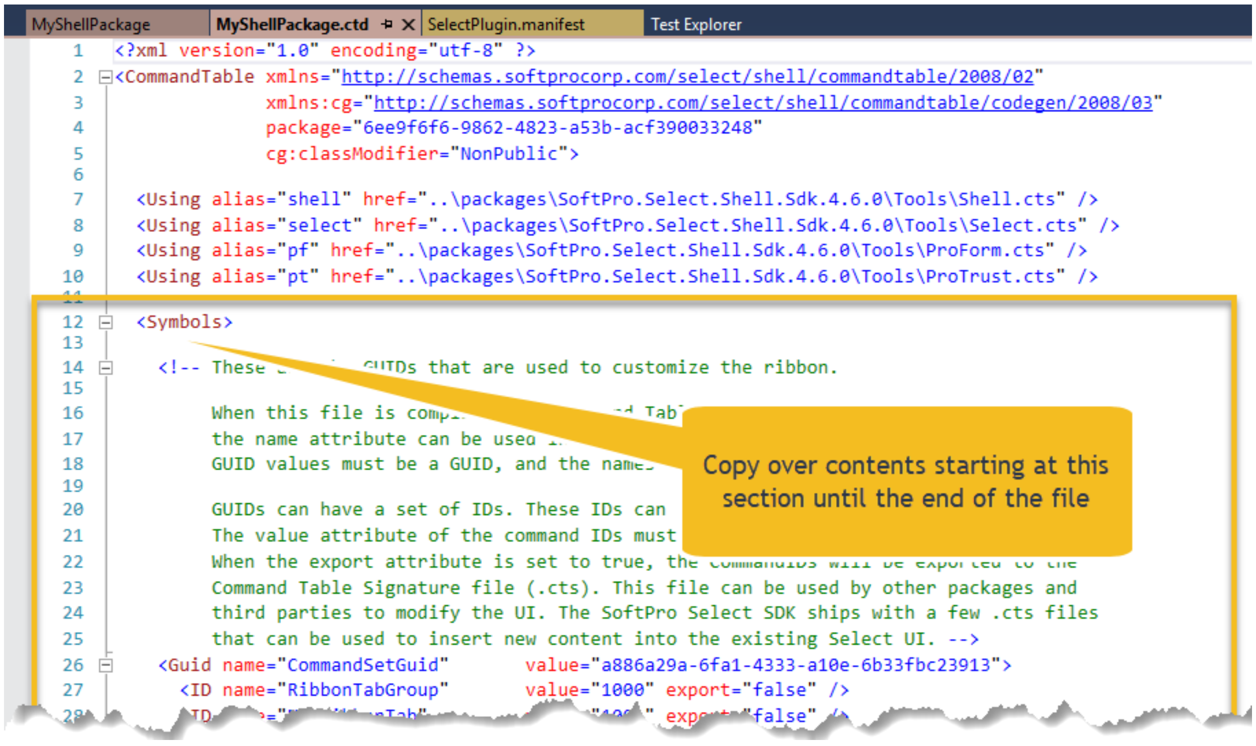
1. Create a **new** Shell Package project in Visual Studio by selecting the **SoftPro Select Shell Package** project template**.**

2. In the **Shell Package Settings** dialog, specify the **same package name** and other parameters as those defined in the existing package code.

3. Open the project's Properties tab and update the **Assembly name** and **Default namespace** such that they match the corresponding metadata of the existing package.

4. Copy and paste contents of the <*package_name*>.cs file (the package class) from the existing package to the new project's <*package_name*>.cs file.

5. Annotate the package class with the [SoftPro.ClientModel.EntryPoint] attribute.
6. Copy and paste contents of the <*package_name*>**.ctd** file based on this screenshot.


MyShellPackage
MyShellPackage.ctd
SelectPlugin.manifest
Test Explorer
Copy over contents starting at this section until the end of the file

```
<?xml version="1.0" encoding="utf-8" ?>
<CommandTable xmlns="http://schemas.softprocorp.com/select/shell/commandtable/2008/02"
              xmlns:cg="http://schemas.softprocorp.com/select/shell/commandtable/codegen/2008/03"
              package="6ee9f6f6-9862-4823-a53b-acf390033248"
              cg:classModifier="NonPublic">

  <Using alias="shell" href="..\packages\SoftPro.Select.Shell.Sdk.4.6.0\Tools\Shell.cts" />
  <Using alias="select" href="..\packages\SoftPro.Select.Shell.Sdk.4.6.0\Tools\Select.cts" />
  <Using alias="pf" href="..\packages\SoftPro.Select.Shell.Sdk.4.6.0\Tools\ProForm.cts" />
  <Using alias="pt" href="..\packages\SoftPro.Select.Shell.Sdk.4.6.0\Tools\ProTrust.cts" />

  <Symbols>

    <!-- These       GUIDs that are used to customize the ribbon.

         When this file is comp
         the name attribute can be used
         GUID values must be a GUID, and the name

         GUIDs can have a set of IDs. These IDs can
         The value attribute of the command IDs must
         When the export attribute is set to true, the
         Command Table Signature file (.cts). This file can be used by other packages and
         third parties to modify the UI. The SoftPro Select SDK ships with a few .cts files
         that can be used to insert new content into the existing Select UI. -->
    <Guid name="CommandSetGuid"       value="a886a29a-6fa1-4333-a10e-6b33fbc23913">
      <ID name="RibbonTabGroup"       value="1000" export="false" />
```

7. Copy over all other files and folders from the legacy package project into the new project.

8. Install other dependencies like NuGet packages, reference common libraries that were referenced by the existing package project, etc.

9. Remove Manifest.xml and/or .lic files from the project, if applicable. They are not relevant to the plugin development model anymore.

10. Build the project and click F5 to debug the package.

SoftPro Select Server Package

1. Create a **new** Server Package project in Visual Studio by selecting the **SoftPro Select Server Package** project template**.**

2. In the **Server Package Settings** dialog, specify the **same package name** and other parameters as those defined in the existing package code.

3. Open the project's Properties tab and update the **Assembly name** and **Default namespace** such that they match the corresponding metadata of the existing package.

4. Copy and paste contents of the <*package_name*>.cs file (the package class) from the existing package to the new project's <*package_name*>.cs file.

5. Annotate the package class with the [SoftPro.ClientModel.EntryPoint] attribute.

6. Copy over all other files and folders from the legacy package project into the new project.

7. Install other dependencies like NuGet packages, reference common libraries that were referenced by the existing package project, etc.

8. Remove Manifest.xml and/or .lic files from the project, if applicable. They are not relevant to the plugin development model anymore.

9. Stop the running instance of SoftPro Select Server via the SoftPro Select Services MMC snap-in.

10. Build the project and click F5 to debug the package.

- You will need to run Visual Studio as an **Administrator** to debug Server package projects.
- It is **highly recommended to build strongly-signed Shell and Server package assemblies** before they are packaged into a plugin.

SoftPro Select Plugin

Finally, you will need to bundle the upgraded Shell and/or Server packages into one or more plugins depending on how you want to control the packaging and deployment of features encapsulated in the custom packages. Follow the same steps for creating and building a plugin file as listed in the "*SoftPro Select Plugin*" section above.

Managing SDK Upgrades

In versions prior to 4.6.0, if there was a change to the major or minor version of SoftPro Select, developers would need to rebuild their custom packages against the upgraded version of the SDK and re-deploy the updated packages on the client and/or server.

The plugin infrastructure of SoftPro Select is engineered such that a Shell/Server package written against version 4.6.0 of the SDK will continue working seamlessly with future versions, regardless of major or minor version changes, of SoftPro Select without requiring developers to rebuild against the latest version of the SDK. Plugin authors will not be required to create and deploy upgrades to their plugin unless of course, there are feature-updates to the plugin itself.

Caveat

We typically manage incremental evolution of the Select platform in a way that there are no breaking changes to the SDK and if there are outliers, their footprint is kept as small as possible. In the event we communicate the need for developers to rebuild their packages against a particular version of the SDK, you will need to install that version of the *SoftPro.Select.Shell.Sdk* NuGet package in Shell package projects, *SoftPro.Select.Server.Sdk* in Server package projects and *SoftPro.Select.Plugin.Sdk* in Plugin package projects.

Common Libraries

As developers, we may want to abstract reusable code and components into class libraries that can be referenced by Shell/Server packages.

Depending on the scope of the SoftPro Select API that you may want to consume in such a library, you can install relevant NuGet package(s) from nuget.org

- **SoftPro.Select.Core**
  This package contains assemblies that are consumed by Shell and Server packages.

  SoftPro.Accounting.Client.dll
  SoftPro.ClientModel.dll
  SoftPro.Documents.Client.dll
  SoftPro.EntityModel.dll
  SoftPro.Imaging.Client.dll
  SoftPro.OrderTracking.Client.dll
  SoftPro.ProceedsTracking.Client.dll
  SoftPro.Register.Client.dll
  SoftPro.Reporting.Client.dll
  SoftPro.Select.Client.dll

- **SoftPro.Shell.Core**
  This package contains SoftPro.Select.Shell.dll which is the core assembly that encapsulates features of the SoftPro Select Shell framework.

- **SoftPro.Server.Core**
  This package contains assemblies that may be consumed by server-side code.

  SoftPro.PersistenceModel.dll
  SoftPro.Select.Service.dll
  SoftPro.ServerModel.dll

- **SoftPro.Select.Controls**
  This package contains assemblies required for consuming SoftPro Select controls and snapsections, writing extensions to them, etc.

  SoftPro.Accounting.Controls.dll
  SoftPro.OrderTracking.Controls.dll
  SoftPro.OrderTracking.SnapSections.dll
  SoftPro.Select.Controls.dll
  SoftPro.Select.OrderTracking.Shared.dll

Conclusion

The SoftPro Select Plugin development model offers a sophisticated framework for writing, packaging and deploying extensions to the SoftPro Select platform. A plugin represents a “Unit of Work” that may host Shell and/or Server packages. It can be considered as a standalone, independent component that encapsulates a set of related features and helps streamline the deployment of those features in the SoftPro Select ecosystem.

This whitepaper guides 3rd party developers with a roadmap to create new Shell or Server packages, plugins and migrate existing Shell and/or Server packages to work with the new plugin model of SoftPro Select.

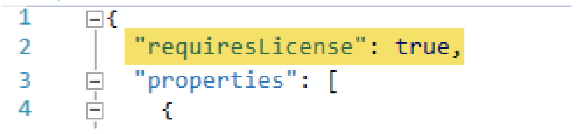
Addendum

Plugin Settings

Developers often find the need to integrate configurable settings that are used by custom Shell or Server packages. Typically, such information would be stored in .config files on disk or custom server-side attributes or having the Server package create custom tables to store settings, etc.

SoftPro Select SDK provides developers with the opportunity to specify custom settings in the plugin's manifest. The settings are defined in JSON as key-value pairs and built into the plugin's .nupkg file. A user who has relevant permissions granted in the SPAdmin module, may view and edit the plugin's settings after the plugin is installed in the system. The plugin package's code may use the plugin manager API in the SoftPro.Select.Client.Plugins namespace in order to read the settings of a given plugin.

"requiresLicense"



```
{
  "requiresLicense": true,
  "properties": [
    {
```

As the name suggests, this setting specifies whether or not the plugin would require a valid license for it to be operational in the system. This is an **optional** setting. If not specified, the plugin is considered to not require a license.

If a plugin requires a license, you may install the plugin via SPAdmin, but the Shell packages in it will not be downloaded to the SoftPro Select client or the Server packages incorporated into the SoftPro Select server's process until a valid license key is assigned to the plugin. You can find details on the user experience regarding plugin licensing in SoftPro Select Release Notes.

"properties"

You can specify an array of setting properties in the manifest JSON file. Select 4.6.0 supports four types of properties:

1. **Boolean:** Translates into a System.Boolean (bool) CLR type.
2. **Integer**: Translates into a System.Int64 (long) CLR type.
3. **Number**: Translates into a System.Double (double) CLR type.
4. **String:** Translates into a System.String (string) CLR type.

A property has attributes that define the metadata of the property and determine the usage and behavior of the property in the system.

"***name***": The canonical name of the property. It must adhere to CLI naming conventions of a property defined in a typical class or struct in C#. A plugin cannot have properties with duplicate names. This is a **required** attribute.

"***type***": This represents the data type of the property. This is a **required** attribute.

“***displayName***”: This provides a “friendly name” that will displayed against the property in the settings user interface in SPAdmin. This is an **optional** attribute. If not specified, the “name” of the property will be displayed.

“***description***”: The description of the property. This is an **optional** attribute.

“***defaultValue***”: The default value that is assigned to the property. Its value must be compatible with the JSON-equivalent data type of the property. This is an **optional** attribute. If not specified, the value will be the default value of the underlying CLR type.

“***required***”: This has a boolean value that specifies whether or not the property can be null in the system. This is an **optional** attribute. If not specified, the property can support null or blank values.

“***allowedValues***”: This specifies an array of unique values that must be of the same type as the “type” of the property. A property with this attribute manifests as a dropdown list on the settings user interface in SPAdmin. You can only assign this property one of the values defined in the allowedValues array. This is an **optional** attribute.

Sample Plugin Settings JSON Snippet

```
{
  "requiresLicense": false,
  "properties": [
    {
      "name": "MyBooleanSetting",
      "type": "boolean",
      "displayName": "The display name",
      "description": "The boolean setting.",
      "defaultValue": true
    },
    {
      "name": "MyIntegerSetting",
      "type": "integer",
      "displayName": "The display name",
      "description": "The integer setting.",
      "defaultValue": 2
    },
    {
      "name": "MyIntegerListTypeSetting",
      "type": "integer",
      "displayName": "The display name",
      "description": "The integer setting that manifests as a list in the UI.",
      "defaultValue": 2,
      "allowedValues": [
        1,
        2,
        3,
        4
      ]
    },
    {
      "name": "MyNumberSetting",
      "type": "number",
      "displayName": "The display name",
      "description": "The number setting.",
      "defaultValue": 2.1
    },
    {
      "name": "MyNumberListTypeSetting",
      "type": "number",
      "displayName": "The display name",
      "description": "The number setting that manifests as a list in the UI.",
      "defaultValue": 2.1,
      "allowedValues": [
        1.1,
        2.1,
        3,
        4
      ]
    },
    {
      "name": "MyStringSetting",
      "type": "string",
      "displayName": "The display name",
      "description": "The string setting.",
      "defaultValue": "default",
      "required": true
    },
    {
      "name": "MyStringListTypeSetting",
      "type": "string",
      "displayName": "The display name",
      "description": "The string setting that manifests as a list in the UI.",
      "defaultValue": "default",
      "allowedValues": [
        "default",
        "abc",
        "def",
        "ghi"
      ]
    }
  ]
}
```

Contributors

The following individuals at SoftPro contributed to this document:

- Phil Barton, Systems Architect.
- Yatin Tawde, Sr. Software Engineer.
- Nathaniel Davenport, QA Automation Engineer.