Documenting the Plugin API

Servoy supplies a Java Eclipse Documentation Generator plugin to help with documenting the plugin API.

This plugin allows to transform the plugin JavaDoc into a Servoy plugin xml api file. The generated XML file needs to be bundled with the plugin, which then will be used in Servoy Developer to provide the information (namely, the documentation) to the developer using the plugin.

All the relevant API information can be supplied by JavaDoc directly on the public methods and properties of the plugin.

The IScriptable Interface

The IScriptable interface is an empty interface that classes that expose methods or properties to the JavaScript layer in Servoy have to implement. When plugins get loaded in Servoy Developer, all documentation provided in the classes implementing this interface will be loaded. For more information refer to the IScriptable API doc in the Servoy API.

The IScriptable interface replaces three methods of the IScriptObject interface previously used to provide documentation for plugin methods and properties: g`etSample`, `getToolTip`, `getParameterNames`.

The `getAllReturnedTypes()` method of the deprecated IScriptObject is replaced by separate interface IReturnedTypesProvider, which only needs to be implemented on plugin classes that have actual returnTypes.

Documenting Method & Properties Using JavaDoc

When implementing the new documentation mechanism, the method sample code, tooltip, parameter names will have to be documented using JavaDoc. For more details on writing documentation in the Javadoc way, see the Javadoc reference pages. This type of code documentation is a standard way of documenting Java code.

The supplied JavaDoc can be extracted in XML format using the Documentation Generator below and then be bundled with the plugin for Servoy Developer to use.

A plugin must annotate itself so that the Document Generator Tool knows what to get. Servoy has a special annotation called `@ServoyDocumented`. The annotation can have additional attributes to specify the public and scripting names. Below, is a sample of the FilePlugin:

```
@ServoyDocumented(publicName = FilePlugin.PLUGIN_NAME, scriptingName = "plugins." + FilePlugin.PLUGIN_NAME)
public class FileProvider implements IReturnedTypesProvider, IScriptable
```

For returnTypes like JSFile the `@ServoyDocumented` annotation marks a class for documentation:

```
@ServoyDocumented
public class JSFile implements IScriptable, IJavaScriptType
```

Note that if a class is documented, documented methods/properties will be looked up in the class hierarchy. If a method is present multiple times in hierarchy, the first occurrence will be used by doc generator.

Methods can be annotated in order to be marked as properties, functions or as deprecated.

If a method is annotated with the special annotation `@JSReadOnlyProperty`, it will bee considered a readonly property of the plugin.

```
/**
 * This is a readonly property
 */
@JSReadOnlyProperty
public String myPropertyVersion()
{
     return "version 1.0";
}
```

Two methods using the `@JSGetter` and `@JSSetter` properties can be used to create a property of the plugin, as follows:

```
@JSGetter
public String getSomeProperty()
{
     //...
}

@JSSetter
public void setSomeProperty(String someProperty)
{
     //...
}
```

The code above will create a JavaScript property called 'someProperty' of the plugin.

If a method is annotated with the `@JSFunction` property it will be considered a JavaScript function:

```
@JSFunction
public String myFunction(String myParameter)
{
     //...
}
```

Note that the method above does not have to start with the `js_` prefix in order to be considered a JavaScript function of the plugin it belongs to.

If a method should be marked as deprecated, the new documentation generator tool also takes into consideration the `@Deprecated` annotation:

```
@Deprecated
public void neverRetire()
{
     //...
}
```

The result is that such a method will not show up in the plugin's (Solution Explorer) list view.

Methods can be documented according to the sample below:

```
/**
 * This is the method description.
 *
 * @sample
 * var message = someMethod('someParameter');
 *
 * @param simpleParameterName this is a simple parameter name
 *
 * @return what the method returns
 */
public String js_someMethod(String simpleParameterName)
{
     //...
}
```

Besides the standard JavaDoc tags like `@param`, `@return` and `@since`, Servoy introduces a few tags to provide sample code part or to clone parts of the JavaDoc of another method.

- @sample: doc below this section, until the next tag is seen as sample code. This should contain the text previously returned by `getSample(String methodName)`
- @sampleas: this will copy over the sample code from another method: `@sampleas js_someMethod(String)`
- @clonedesc: this will copy over the description from another method: `@clonedesc js_someMethod(String)`
- @sameas: this will copy the entire documentation from another method: `@sameas js_someMethod(String)`

The `@param` tag supports so-called rest parameters by using `...` prefix. It also supports marking parameters as optional. The following example is the button a optional and variable arguments parameter.

```
 @param ...button optional
```

To be able to type a specific variable, like `/** @type {JSFile} */` , using a JSDoc annotation, the last slash must be escaped by using the ASCII notation:

```
/** @type {JSFile} *&#47;
```

The Documentation Generator Tool

After annotating the source of the plugin using JavaDoc, the documentation needs to be exported to XML format and bundled with the plugin. Servoy provides an Eclipse plugin to generate the XML, called the Documentation Generator.

In order to have the method documentation available in JavaScript, the documentation generation tool needs to be installed in the Eclipse in which the plugin is being developed. The documentation generation tool must be installed via Eclipse update sites. Go to **Help** > **Install New Software...**, add the http://download.servoy.com/documentbuilder URL and install the documentation generator (Servoy Documentation Generator Feature). It will show up under installed software as the Servoy Documentation Generator Feature.

> **Note**
>
> Note that **Group items by category** has to be unchecked in the Install New Software first wizard page.

After the tool is installed, the documentation of the plugin can be generated by selecting the project/package corresponding to a Servoy plugin, ie. a class that implements IClientPlugin, and using the context menu **Servoy** > **Generate Servoy Documentation XML** in the Package Explorer view. Separate documentation XMLs are generated for each of the topmost package which contains a Servoy plugin, so one documentation file per plugin.

For instance, if the package structure is the following:

com.mycompany.myplugin (this package contains a class implementing IClientPlugin)
com.mycompany.myplugin.feature1
com.mycompany.myplugin.feature2

A *servoydoc.xml* file will be generated in the *com.mycompany.myplugin* package.

In order to have this documentation available with the plugin, in the Servoy Developer, the plugin developer must include and bundle the documentation XML file along with the actual plugin, inside the plugin jar file. Then, once the plugin jar is copied to the Servoy installation's `application_server/plugins` directory, the documentation XML, corresponding to that plugin, will be automatically loaded and the documentation will be available.

The documentation generator tool also has a corresponding preference page, which will show up in Eclipse under **Windows** > **Preferences** > **Servoy Documentation Generator**. There are currently two properties which can be set:

- Show confirmation dialog before overwriting an existing file
- Show a final summary with an overview of the files that were generated