Developing Plug-in Checks for Autodesk Revit® Model Review

Introduction

Autodesk Revit® Model Review is a platform for configuring and executing checks to ensure that Revit models match the requirements for the models. Using the out-of-the-box product, an administrator is able to create many different types of check using the "templates" provided. In some situations, BIM administrators will want to check for things which are not possible to check with the "out-of-the-box" Model Review templates – either because the check has not yet been implemented – or because the check is too specialized to ever exist in the base product.

The plug-in mechanism exists for developers who are familiar with the Revit® API to create new check templates which can evaluate the model using any capabilities of the Revit® API.

NOTE: This document refers to Autodesk Revit® Model Review 2010.

Overview

Developers can create plug-ins for Model Review by implementing the basic behaviors for a check, as well as passing information back and forth between Model Review and the plug-in check (which resides in a separate DLL). These behaviors include:

- Initialization
- Configuration (optional)
- Reporting (optional)
- Run
- Correct (optional)

Procedure

To create a new Model Review plug-in, the developer should use the following steps:

- Create a new Visual Studio Project Class Library Project
- Add a reference to the Revit API DLL
- Add a reference to the ModelReviewPlugins DLL
- Create a class to implement the check
- Add the IPluginCheck interface to your class
- Implement the members of the IPluginCheck interface:
    - Initialize
    - Configure (optional)

    - GetReportVariables (optional)
    - RunCheck
    - CorrectCheck
- Build the Project
- Copy the DLL to the plugins folder.
- The Check Template will appear on the Check + Add + Plugins menu the next time that ModelReview is run.

Filter Concept

The Model Review platform makes use of a "Filter" concept to separate the determination of what elements to check from the actual running of the check. This also makes the filter more easily configurable for each different check. Examples of filters include:

- Element Types – such as rooms
- Element parameters – rooms with Area > 9.0
- View Types – Elevation views with scale > 1:48

Plugins are also capable of leveraging the filter mechanism. Developers do this by setting the filter type during the initialization process. If the plugin does not need to use the filter mechanism, it can specify "NONE" as the Filter Type.

ICheckData Exchange Reference

The ICheckData structure is used to pass information back and forth between the Model Review application and the plugin.

Name	Type	Description
Name	String	Name of the Check
Category	String	Category of the Check
FilterType	Enum	Type of Filter: - None, Elements, Rooms, PlanCircuits, Views
FilterSubTypes	String[]	Array of Filter subtypes: - Elements: The type name or "ALL" - Views: the ViewType
PassMessage	String	Message to show when the check passes
FailMessage	String	Message to show when the check fails
ConfigData	String	Configuration data provided by the plugin which will be stored in the check file.
PluginName	String	Name of the Plugin
PluginGUID	String	Unique identifier for the plugin
SupportsFamilyDocuments	Boolean	Whether the plugin supports standalone family files.
SupportsEmbeddedFamilies	Boolean	Whether the plugin supports running on

		embedded families in a project.
SupportsProjects	Boolean	Whether the plugin supports running on projects.
IsConfigurable	Boolean	Whether the plugin offers a configuration user interface.
IsCorrectable	Boolean	Whether the plugin check is correctable
RevitDocument	Document	The current Revit document
App	Application	The current Revit Application
FilterData	List<object>	List of Revit objects to be checked by the plugin.
Result	Enum	Result Type from the plugin: - UNKNOWN: no result - Passed: the plugin passed - Failed: the plugin failed - ReportOnly: the plugin does not return pass or fail – just a report - Skipped: the check is not appropriate to run on this model - ERROR: the check encountered an error
ReportTokens	Dictionary<string,string>	List of replaceable report tokens
ResultData	Object	Object for the plugin to store result data in (for later use in correction). DO NOT STORE ELEMENTS, please use ELEMENTIDs.
ResultsTree	TreeNode	Tree illustrating the failed items from the plugin. NOTE: each TreeNode tag may have an integer ElementId. If the tag is set, Model Review will attempt to "Show" the selected Element when the user clicks on that node.

Method Reference

This section describes the methods in the IPluginCheck interface.

```
Initialize(ICheckData data)
```

The Initialize method is used by the plugin to initialize information about the check, typically including:

- Filter Type
- Filter Sub Types (optional – if fixed)
- IsConfigurable
- Correctable
- PluginGUID
- SupportsProjects/SupportsFamilyDocuments

```
Configure(System.Windows.Forms.IWin32Window window, ICheckData data)
```

The configure method is optional – it is used to provide a graphical user interface to configure the plugin check. The window is the configuration window, which should be used as the parent of any forms displayed. The data is the mechanism for passing information back and forth between Model Review and the plugin. The ConfigData string can be used to persist data back to the check file.

```
List<ReportVariable> GetReportVariables()
```

The GetReportableVariables method is optional – it enables the developer to define variables which can be inserted into the report. This method is called from the configuration tool.

```
void RunCheck(ICheckData data)
```

The RunCheck method performs the actual checking process, determining if the check passes or fails, as well as providing the supporting information. The typical process that the RunCheck method must follow is along these lines:

Retrieving Configuration Data
If the check stored configuration data, this needs to be extracted/deserialized from the ConfigData string.

Filter Retrieval
If the plugin uses filters, the FilterData will be populated with a List of objects containing the references to the Revit objects to be checked. This list will likely need to be cast into the specific types being checked by this plugin.

Determining Pass or Fail
The plugin must fundamentally decide whether the current model passes or fails the check.

Building Reports
The plugin may build ReportTokens based on reportable variables that the plugin provides to show results.

Building the ResultsTree
The plugin may build a tree structure to illustrate the results.

Storing Results
If the plugin is able to do automatic correction, it may be desirable to store information about each failed element and perhaps the correct value. This is recommended so that when the “CorrectCheck” method is called later, that method does not have to “re-check” everything to identify what to fix.

The RunCheck method can store information in the ResultData property – and this information will be given back to the plugin during correction.

NOTE: It is very important not to store Element pointers in the result data, because they will almost certainly not exist at the point when correction happens. Instead, store only ElementIds, GUIDs, or anything else which can be re-found later.

```
void CorrectCheck(System.Windows.Forms.IWin32Window window, ICheckData data)
```

In the case of a plugin which supports correction, the CorrectCheck method will be called if the user clicks on the “Correct” button.

The CorrectCheck method is used to make changes to the current Revit model so that (hopefully) the check will pass. The method has access to the ICheckData structure, which should contain configuration data as well as result data (if appropriate).

Important Notes about correction:

- Because this is running using a modeless approach – it is MANDATORY for developers to implement any changes to the Revit model in a transaction (using the Document.BeginTransaction and Document.EndTransaction calls).
- Because the model could have changed between the time it was run and the correction was applied, ALWAYS test each stored element to ensure that it still exists before attempting to modify it.
- The correction should make whatever changes are necessary – but the plugin check will automatically be re-run after the correction has been completed (so do not bother to reset the result, etc).