SPECIAL NOTES:

dynamic keyword – Allows objects to be defined at run-time without static type checking during compile time. Perfect for IronPython’s dynamic nature. Downside there is no Intellisense pointing you back to function calls back in the Python code.

IMPORTANT CLASSES & VARIABLES

Project:Data

Class:AnalysisData

Provides a significant amount of data handling. Where most the communication between Python and C# data passing happens.

Variable:dynamic \_podDoc – IronPython reference to the Python class CPodDoc which is the main Python class in the TestingPythonCode library.

Function:TransformData – Perform data transforms on crack size and flaw data.  
Function:SetFlawData – Set crack size data.  
Function:SetResponseData – Set response data.  
Function:SetAllMissingData – Set removed data points so we can create a curve with all points for reference visualization for the user

Function:UpdateHitMissOutput – Get data out of \_podDoc and into C# data structures for hit miss data  
UpdateAHatOutput– Get data out of \_podDoc and into C# data structures for ahat vs a data

Variable: IPy4C \_python – This contains IronPython engine that lets C# start talking to the Python classes.

Project:Globale

Class:IPy4C

Provides functions to transform data types between Python & .Net types suchs as enumerations and more complicated data types like dictionaries or lists.

Constructor – where you decide to run Python from the compiled DLLs or the Python files themselves to access them for debugging.