
Test Documentation

Release test

Test

March 26, 2015

1	Sphinx .Net API Reference Design	1
1.1	Goals	1
1.2	Introduction	2
1.3	Proposed Architecture	2
1.4	.Net Domain	3
1.5	Examples	4
1.6	Other Ideas	4

Sphinx .Net API Reference Design

This document talks about the design of a .NET Sphinx integration. This will include a Sphinx “dotnet” domain, and a mechanism for generating Javadoc style API references automatically. We will describe decisions that lead to specific implementation details.

1.1 Goals

The main goal of this project is to be able to generate a MSDN or Javadoc style API reference from a .Net project in Sphinx.

1.1.1 Primary Goals

- Build a dotnet Sphinx domain that allows .Net code to be modeled in Sphinx.
- Build MSDN/Javadoc style HTML output for arbitrary .Net code.
- Have specific pages for each Package, Class, and all class-level structures.
 - */api/System/*
 - */api/System/String/*
 - */api/System/String/Constructors/*
- Allow users to link to specific Doc ID's from any location in Sphinx.
 - `:ref: `System.String#ctor(Char*)``
- Have syntactic sugar for common reference types:
 - `:ctor: `System.String``
 - `:properties: `System.String``
 - `:methods: `System.String``

1.1.2 Secondary Goals

- Allow for definition of .Net classes inside of normal Sphinx prose docs (classic Sphinx style definition).
- Allow generation of Javadoc style docs from other languages.

1.1.3 Requirements

All of this work is hinged on the ability to generate YAML from .Net code bases, which is still a work in progress.

1.2 Introduction

We are working with Sphinx, which has an existing way of doing this. Generally, you define a *Domain* which describes the various language structure, a *Class* or *Method*, for example. Then the user will write RST that uses these definitions, and Sphinx will create output from that markup.

```
.. py:function:: spam(eggs)

    Spam the foo.
```

The author of the documentation will have now told Sphinx that the *spam* function exists in the Python project that is being documented.

1.2.1 Autogenerated Output

Sphinx then built a series of tools to make the generation of this markup easier and more automatic:

- [Autodoc](#)
- [Autosummary](#)

Autodoc is a Python-only solution that imports the author's code into memory, and then allows the author to more automatically document full objects. For example, you can document a whole class on a page.

```
.. autoclass:: Noodle
```

This will generate output that looks like:

```
class Noodle
    Noodle's docstring.
```

There are also options for it to include a full listing of the classes attributes, methods, and other things, automatically.

Warning: Remember, this depends on `Noodle` being importable by the Python interpreter running Sphinx.

1.3 Proposed Architecture

The proposed architecture for this project is as follows:

- A program that will generate a YAML file from a .Net project, representing it's full API information.
- Read the YAML and generate an appropriate tree structure that will the outputted HTML will look like (YAML-Tree)
 - If time allows, we will allow a merging of these objects with multiple YAML files to allow for prose content to be injected into the output
- Take the YAML structure and generate in-memory rst that corresponds to the Sphinx dotnet Domain objects
- dotnet Domain will output HTML based on the doctree generated from the in-memory RST

In diagram form:

```
.Net API -> YAML -> YAMLTree -> RST (Dotnet Domain) -> Sphinx -> HTML
```

1.3.1 YAMLTree

One of the main problems is how to actually structure the outputted HTML pages. The YAML file will likely be ordered, but we need to have a place to define the page structure in the HTML.

This can be done before or after the loading of the content into RST. We decided to do it before loading into RST because that matches standard Sphinx convention. Generally the markup being fed in as RST is considered to be in a file that maps to it's output location. If we tried to manipulate this structure after loading into the Domain, that could lead to unexpected consequences like wrong indexes and missing references.

1.3.2 Sphinx Implementation

The user will run a normal *make html* as part of the experience. The generation and loading will be done as an extension that can be configured.

There will be Sphinx configuration for how things get built:

```
autoapi_root = 'api' # Where HTML is generated
autoapi_dir = 'yaml' # Directory of YAML sources
```

We will then loop over all YAML files in the `autoapi_dir` and parse them. They will then be output into `autoapi_root` inside the documentation.

1.4 .Net Domain

The .Net (dotnet) domain will include the following concepts.

1.4.1 Structure

- Namespace Collection
- Sub Namespace
- Namespace
- **Types**
 - Classes
 - Structures
 - Interfaces
 - Delegates
 - Enumerations
 - Nested types
- **Members**
 - Constructors

- Methods
 - Properties
 - Fields
 - Operator
 - Indexers
 - Events
 - Member groups
 - Overload lists
 - Extension methods
 - Attached properties
 - Inherited members
 - Overrides
- Parameters
 - Attributes
 - Generics (Type Parameters)
 - Versioning
 - Obsolete members and types
 - NuGet Packages

1.5 Examples

A nice example of Sphinx Python output similar to what we want:

- http://dta.googlecode.com/git/doc/_build/html/index.html
- Src: <https://raw.githubusercontent.com/sfcta/dta/master/doc/index.rst>

An example domain for Spec:

- <https://subversion.xray.aps.anl.gov/bcdaext/specdomain/trunk/src/specdomain/sphinxcontrib/specdomain.py>

1.6 Other Ideas

Warning: Things in this section might not get implemented.

The .Net domain will not be able to depend on importing code from the users code base. We might be able to implement similar authoring tools with the YAML file. We might be able to output the YAML subtree of an object with autodoc style tools:

```
.. autodocclass:: System.String
   :members:
```


This is an autogenerated index file.

Please create a `/Users/eric/projects/sphinx-markdown-test/index.rst` or
`/Users/eric/projects/sphinx-markdown-test/README.rst` file with your own content.

Here is the `Newtonsoft.Json.Bson.BsonObjectId`

N

Noodle (built-in class), [2](#)