**Breeze Entity Generation**

Version 1.0

Prepared For: Fox Networks Group, Inc.

Prepared By: IdeaBlade, Inc.

Authors: Marcel Good, Tiberiu Covaci, Shefali Sinha

Date: June 28, 2013

Table of Contents

[Architectural Overview 1](#_Toc360193827)

[Tools 2](#_Toc360193828)

[Overview 2](#_Toc360193829)

[Metadata Generator 3](#_Toc360193830)

[TypeScript Generator 4](#_Toc360193831)

[Complete Tool Chain 6](#_Toc360193832)

# Architectural Overview

FNG DbContext

CSDL Metadata

Me

Breeze Native Metadata (Optional with configuration)

Breeze

Node

Figure Metadata Generator

Breeze Native Metadata

TypeScript Classes

Figure TypeScript Generator

CSDL Metadata

Breeze

Handlebars

Breeze

Figure Tool Chain Flow

Metadata Generator

TypeScript Generator

# Tools

## Overview

The tools described herein form the tool chain set needed to generate TypeScript classes. They can be invoked from the console as per their respective usage. Initial input to this set of tools consists of the compiled DbContext and entities. The final output of the TypeScript Generator is a TypeScript class for each entity and optionally complex types found in the DbContext.

The environment of execution should have the following setup prior to invoking the tools:

* Node.js should be installed. This is available at nodejs.org.
* Node.exe should be in the PATH environment variable

The following prerequisites are recommended for better understanding:

* Node.js
* Breeze.js
* Handlebar.js

## Metadata Generator

Metadata Generator is the first tool that is invoked in the tool chain set. It accepts the DLL containing the DBContext and generates metadata.

|  |  |
| --- | --- |
| **Usage:** | metagenerator[.exe] –i <file> [-n] [-o <output file>] [-d <directory>] [-m] –t <DBContext Type> |
| **Input(s):** | Assembly containing the DbContext, or the ObjectContext class definition |
| **Output(s):** | CSDL or Breeze native metadata stored in one or more files using the naming convention <Fully Qualified DBContext Type Names>.json |
| **Description:** | This tool has a dependency on the nodeFiles directory. It loads the assembly containing the entities. For each DbContext, this tool generates a CSDL file. If there are multiple DbContext, the tool should be run once for each DbContext. Following this, optionally to obtain Breeze metadata, invoke Node.js that converts the CSDL format to Breeze native format and stores in specified file.  Parameters:  **-input-file <file>:** Specifies the assembly containing the DBContext or ObjectContext class definition  **-native:** Optionally, specify this flag if output format is to be native, otherwise output format is CSDL  **-output-file <file>:** The name of the output file. Default value is the fully qualified DbContext type name  **-output-directory <directory>:** The name of the directory in which to save the output file(s). If not specified, the current directory is used as the location.  **-multiple-type:** Optionally, specify this flag to generate one file per DbContext type found in the assembly.  **-type-name:** The DBContext Type to extract from the input assembly |
| **Files:** | metagenerator.exe  <Fully Qualified DBContext Type Names>.json |

## TypeScript Generator

TypeScript Generator, a platform independent tool, is the second tool in the tool chain set. It accepts CSDL or Breeze native metadata from the Metadata Generator and generates TypeScript Classes relevant to a DbContext and entities.

|  |  |
| --- | --- |
| **Usage:** | node[.exe] .\tsgen.js -input <file> [-output <directory>] [-camelCase] [-baseClass <file>] |
| **Input(s):** | A file containing CSDL or Breeze native metadata |
| **Output(s):** | One typescript file with the corresponding class definition per entity and complex type plus a helper class to register all constructors with Breeze |

|  |  |
| --- | --- |
| **Description:** | Parameters:  **-input <file>:** Specifies the file containing the metadata  **-output <directory>:** Optionally specifies the location for the generate typescript files. If not specified, the current directory is used as the location  **-camelCase:** Optionally generates the property names using camel case. This parameter has no effect if the input file contains Breeze native metadata. (See [http://www.breezejs.com/sites/all/apidocs/classes/NamingConvention.html#property\_camelCase](http://www.breezejs.com/sites/all/apidocs/classes/NamingConvention.html%23property_camelCase))  **-baseClass <file>:** Optionally specifies a typescript base class for all the generated entity classes. The generated entity classes will directly or indirectly inherit from this class. The file must contain a single module and exported class.  At the core of the typescript generator sits handlebars which is responsible for generating the actual typescript source code. The output that handlebars generate can be customized by modifying the templates. More information about handlebars can be found here: <http://handlebarsjs.com/>  **Note:** node.js (<http://nodejs.org/>) must be installed and node must be part of the PATH.  **Custom code and custom references**  The typescript generator preserves two special sections for each class when regenerating the code. Those sections are <code-reference> and <code>. The <code-reference> section is for custom references and the <code> section is for custom methods etc.  Following is an example of a class after it got generated showing the two sections. Everything between the opening and closing tags is preserved.  /// <reference path="Order.ts" />  /// <code-reference> Place custom references between <code-reference> tags  /// </code-reference>  module DomainModel.NorthwindIB {     export class InternationalOrder extends DomainModel.NorthwindIB.Order {         /// <code> Place custom code between <code> tags           /// </code>         // Generated code. Do not place code below this line.         customsDescription: string;         exciseTax: number;       }  } |
| **Files:** | **node\_modules** (Directory containing the third-party node libraries including Breeze)  **entity.template.txt** (Handlebars template for an entity class)  **register.template.txt** (Handlebars template for the ctor registration helper class.  **tsgen.js** (The node script) |

## Complete Tool Chain

A PowerShell script is used to execute the Metadata generator and Typescript generator in the proper sequence with one command line.

|  |  |
| --- | --- |
| **Usage:** | PowerShell[.exe] .\tsg.ps1 -assembly <file> [-outputDir <directory>] [-baseClass <file>] |
| **Input(s):** | An assembly containing the DbContext and Entity classes. |
| **Output(s):** | One typescript file with the corresponding class definition per entity and complex type plus a helper class to register all constructors with Breeze. |
| **Description:** | Parameters:  **-assembly <file>:** Specifies the assembly containing the DbContext and Entities.  **-outputDir <directory>:** Optionally specifies the location for the generate typescript files. If not specified, the current directory is used as the location.  **-baseClass <file>:** Optionally specifies a typescript base class for all the generated entity classes. The generated entity classes will directly or indirectly inherit from this class. The file must contain a single module and exported class.  **Note:** node.js (<http://nodejs.org/>) must be installed and node must be part of the PATH.  **Note:** The PowerShell script execution policy must be configured to allow for the script to run. See the following link for detailed information.  [http://technet.microsoft.com/en-us/library/ee176949.aspx#EEAA](http://technet.microsoft.com/en-us/library/ee176949.aspx%23EEAA) |
| **Files:** | **.bin** (The metadata generator binaries)  **node\_modules** (Directory containing the third-party node libraries including Breeze)  **entity.template.txt** (Handlebars template for an entity class)  **register.template.txt** (Handlebars template for the ctor registration helper class.  **tsgen.js** (The typescript generator node script)  **tsg.ps1** (The powershell script) |