



sqlbits

SPEAKEASY



Analysis Services DevOps using Tabular Editor, git and Azure DevOps

Daniel Otykier / Kapacity

Session scope

- › Tabular models and Source Control (git)
- › Topic branches and workflows
- › Dealing with conflicts
- › Automated builds (Azure DevOps)
- › Automated unit tests (Azure DevOps)*
- › Automated deployment (Azure DevOps)
- › Continuous Integration (Combining all of the above)

*Not covered in this talk

Motivation (why DevOps?)

- › Work simultaneously on multiple features
- › Work in teams of developers
- › Support multiple environments
- › Avoid merge conflicts / manual merging headaches
- › Enforce code review processes (“pull requests”)
- › Automate testing and avoid regressions
- › Easier and more consistent deployments



Demo



Getting started

Prerequisites

- › Tabular Editor

 - › `setx PATH "%PATH%;C:\Program Files (x86)\Tabular Editor\"`

- › Instance of Analysis Services

- › Azure DevOps account

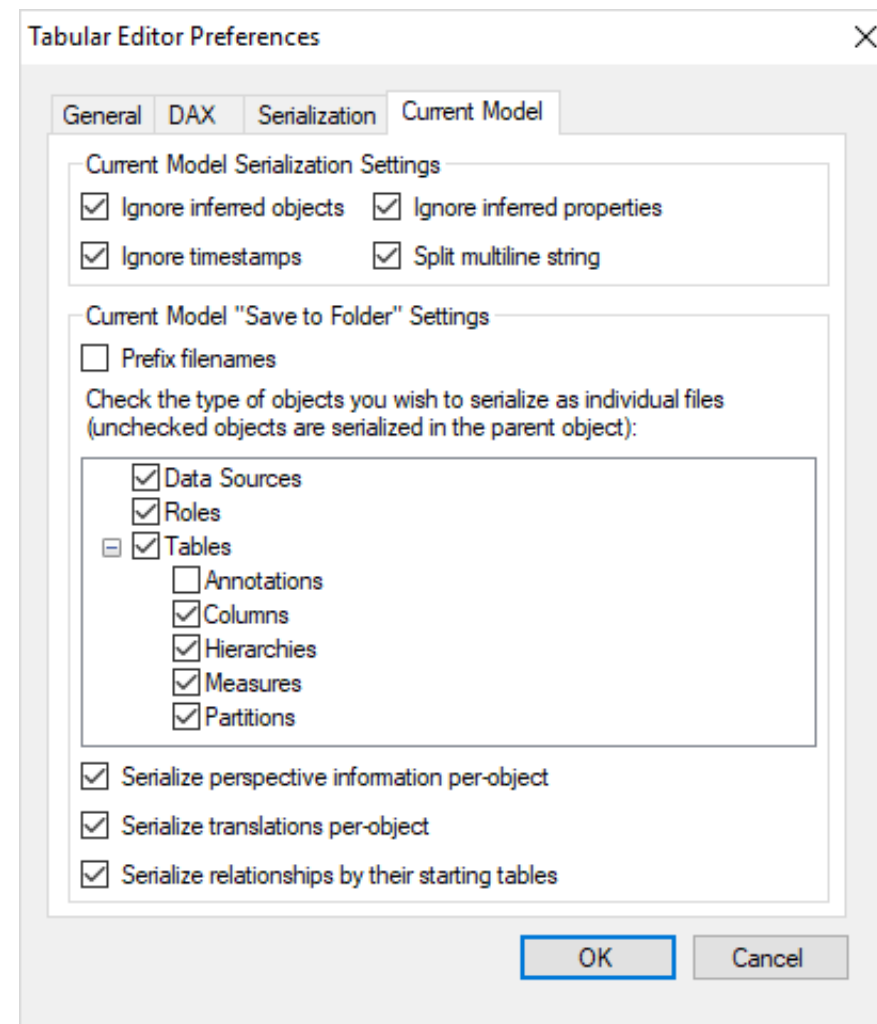
- › Local installation of Git (included with Visual Studio)

“Save to Folder” + Git =

- › Save to Folder
 - › Split model into smaller files
 - › Remove timestamps and inferred metadata
 - › No metadata ordering conflicts
- › Git
 - › Uses directory snapshots
 - › no need to manually “include” items like in TFVC
 - › Prefers many small files over few big files
- › Combine features and resolve conflicts using standard diff tools, instead of specialized tools for Tabular Models (BISM Normalizer)

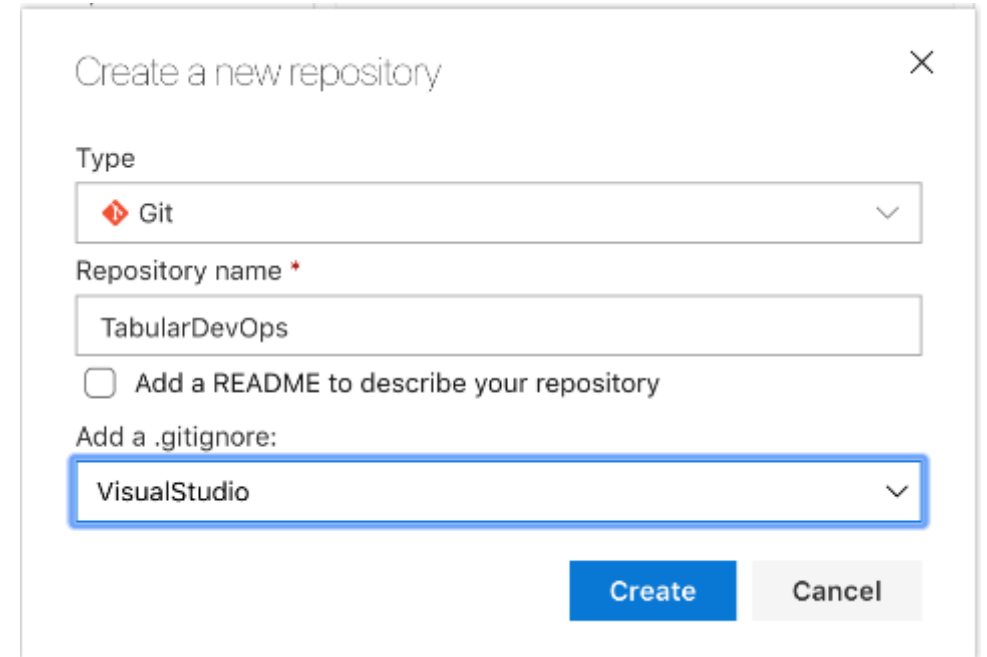
Step 1) Prepare your Tabular Model metadata

- > Load your Model.bim file in Tabular Editor
 - > (alternatively) Load your model from an existing database
- > Go to File > Preferences > "Current Model"
 - > Ensure settings are as seen on the right
- > Save the model using File > Save as Folder...



Step 2) Set up your Git repository

- > One or multiple Git repositories? (Discussion)
- > For demo purposes:
 - > Use default repository in new Azure DevOps project...
 - > ...or create a new repository in existing project
 - > Add a "VisualStudio" .Gitignore file
- > Many ways to add your Tabular Model metadata to Git:
 - > "Upload files" within the Web UI, then clone to a new local repository (*note: doesn't support Folder upload*)
 - > Clone the repository, add the files locally, commit and push
 - > Init local repository, add remote, add files, commit and push
 - > Use Git commandline or Visual Studio ("Open folder") if preferred



Create a new repository

Type
Git

Repository name *
TabularDevOps

Add a README to describe your repository

Add a .gitignore:
VisualStudio

Create Cancel

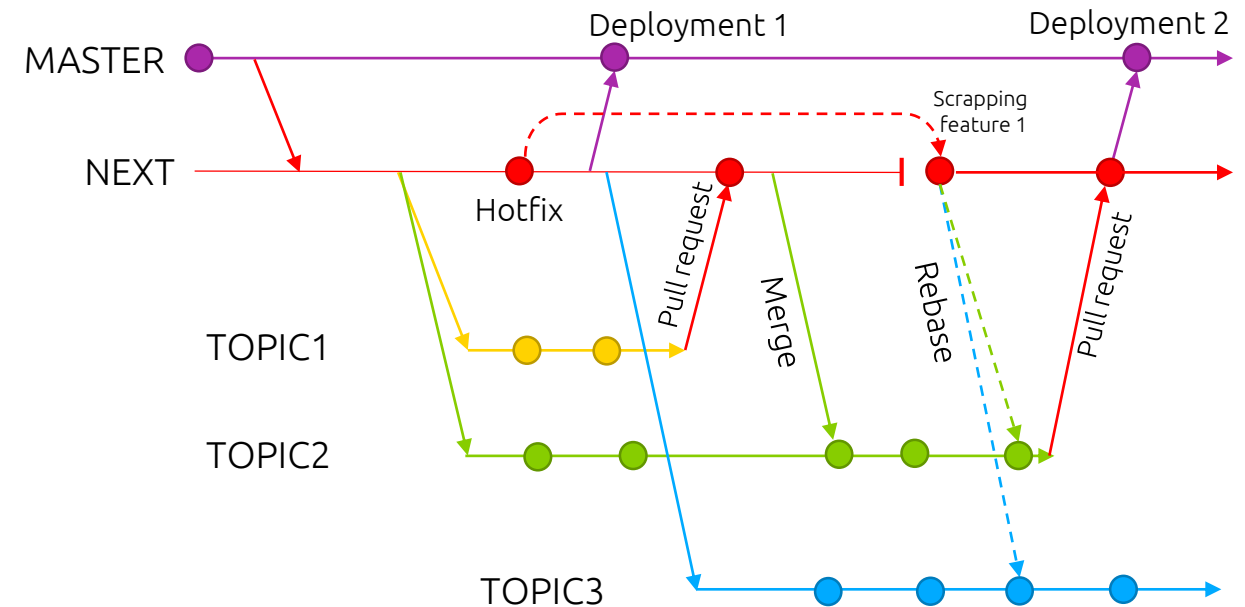
Step 3) Determine branching strategy

> <https://git-scm.com/book/en/v1/Git-Branching-Branching-Workflows>

> One possible approach:

- > MASTER = Production DB
- > NEXT = Test DB
- > TOPIC1 = New feature 1
- > TOPIC2 = New feature 2
- > ...

> Each topic = Separate workspace DB



Step 4) Create build pipelines

- › Download and configure build agent if requiring access to on-prem resources
 - › (Otherwise, may use Hosted Agent pools)
 - › Build agent SA needs admin access to SSAS instance (unless you specify credentials when connecting)
- › Build pipelines are extremely flexible! They can do ANYTHING!
 - › *...but with great power, comes great responsibility! Just because you could doesn't mean you should...*



Daily workflow

Typical daily workflow (after feature branch creation)

- › Merge from TEST to your Feature branch and deploy to DEV DB
- › Open your DEV DB in Tabular Editor and work work work...
- › Hit CTRL+S to save to the DEV DB, and test your changes in Excel / Power BI / SSMS
- › Periodically use “Save to Folder...” to save your work into the current working directory
 - › Consider committing to Git as well (frequent commits are a GOOD thing!)
 - › Might as well push to Azure DevOps, so that you don't lose any work if your laptop is stolen/goes into perpetual BSOD/takes a brick to the face/etc... ☹
- › When the feature is done, pull request into the TEST branch

Dealing with conflicts

- › May occur when merging from TEST into your current Feature branch
 - › I.e. someone else changed something related to what you're currently working on
- › Easiest to do within Visual Studio



Advanced patterns

Master Model Pattern

- › Useful when maintaining 2+ models with a reasonable amount of functional overlap, such as:
 - › Shared dimensions
 - › “Process once”-scenarios
 - › Repeated complexity
- › Helps consolidate your code base
- › <https://Github.com/otykier/TabularEditor/wiki/Master-model-pattern>

Automated deployment

- › Create “Release definition” based on a Build definition
- › Define stages, for example: Test, Pre-prod, Prod
- › For each stage, define deployment tasks (similar to build pipeline)
- › Use variables!
 - › Can be set to “sensitive” for connection strings, passwords, etc.

Continuous integration

- › Set up triggers on build- and release pipelines for end-to-end automation
- › For example:
 - › After every commit, run BPA, SchemaCheck and deploy to build server
 - › Run unit tests
 - › If all succeeds, deploy to test server and inform stakeholder
 - › If stakeholder approves, deploy to prod

Conclusion

- › You **need** Tabular Editor!
- › You **need** Git
- › You **need** Azure DevOps
- › Command-line scripts are oldschool cool
- › Next steps:
 - › Unit testing using PowerShell or NBI
 - › CI/CD for your Tabular data source
 - › Blog series coming soon:
<https://tabulareditor.github.io/2019/02/20/DevOps1.html>



Thank you!



FEEDBACK FORMS

PLEASE FILL OUT AND PASS TO YOUR ROOM
HELPER BEFORE YOU LEAVE THE SESSION



sqlbits

SPEAKEASY