Babelfish Compass: User Guide

Document history:

1.1, Nov-2021: added user-definable overrides, example for **-pgimport**, Mac/Linux support

1.0, Oct-2021: first version

Contents

What Is Babelfish Compass?

The Babelfish Compass tool (short for "**COMP**atibility **ASS**essment") analyzes SQL/DDL code for one or more Microsoft SQL Server databases to identify the SQL features which are not compatible with Babelfish for PostgreSQL.

Users of SQL Server can use Babelfish Compass to analyze the SQL/DDL code for their current SQL Server-based applications for compatibility with Babelfish. The purpose of such analysis is to inform a first Go/No Go decision about whether it makes sense -or not- to consider starting a migration project from SQL Server to Babelfish. For this purpose, Babelfish Compass produces an assessment report which lists -in great detail- all SQL features found in the SQL/DDL code, and whether or not these are supported by the latest version of Babelfish.

A new version of Babelfish Compass will be available as part of every Babelfish release containing new or changed functionality.

Note that Babelfish Compass is a stand-alone, on-premises tool. While Babelfish Compass is part of the Babelfish product, it is technically separate from Babelfish itself as well as from the Babelfish code, and is located in a separate GitHub repository.

Installing Babelfish Compass

Requirements

Before installing Babelfish Compass, the following are required:

- The Java Runtime Environment (JRE) is required to run Babelfish Compass. The Java JRE version must be 8 or higher (64-bit version).
- Babelfish Compass produces compatibility assessment reports in HTML format. For viewing the HTML output, it is recommended to use a recent release of the Google Chrome or Mozilla Firefox browsers.
- On Mac/Linux, you need to be able to run a **bash** script (e.g. with **#!/bin/bash**).

Downloading Babelfish Compass

Babelfish Compass is available as open-source at https://github.com/babelfish-for-postgresql/babelfish_compass

A binary version can be downloaded from here:
https://github.com/babelfish-for-postgresql/babelfish_compass/releases/latest ; pick the most recent **BabelfishCompass_**<version>**.zip** file.
The installation instructions below are based on this download.

Installation

Babelfish Compass is distributed as an "executable JAR", which requires no CLASSPATH settings. The only environmental requirement is that the Java JRE is in the PATH.

Installation steps on Windows:

1. Download the **BabelfishCompass.zip** file (see previous section)
2. Unzip this file so that the contents are placed in an installation directory of choice; the rest of this document will assume **C:\BabelfishCompass**
    a. In case a previous installation is already present in your installation directory, you can overwrite this (but it is recommended to make a backup copy first)
3. Installation is complete.

Installation steps on Mac/Linux:

1. Download the **BabelfishCompass.zip** file (see previous section)
2. Unzip this file so that the contents are placed in a directory of choice, for example **/home/***username***/BabelfishCompass .**
    a. Do not install Babelfish Compass into **/home/***username***/BabelfishCompassReports**, since this is where the generated reports will be located, which should be kept separate
    b. In case a previous installation is already present in your installation directory, you can overwrite this (but it is recommended to make a backup copy first)
3. Verify the shell script **BabelfishCompass.sh** is executable by running **./BabelfishCompass.sh**
   If it is not executable, run: **chmod +x BabelfishCompass.sh**
4. Installation is complete.

Running Babelfish Compass (Windows)

On Windows, Babelfish Compass is invoked by opening a **cmd** command prompt (a.k.a. "DOS box") in the directory where Babelfish Compass is installed. This document assumes it is installed in **C:\BabelfishCompass**.

At the command prompt, run **BabelfishCompass.bat** with various additional command-line options as needed (see below in this document).
To see online help information on the various command options, specify **-help**:

C:\BabelfishCompass> BabelfishCompass -help

Babelfish Compass usage typically starts with creating an assessment report file. The assessment report output file provides a detailed summary of the supported and unsupported SQL features in Babelfish for the analyzed SQL Server script(s).
In the simplest, and likely most common case, a single SQL/DDL script is analyzed. This requires specifying a report name and an input file, for example:

C:\BabelfishCompass> BabelfishCompass MyFirstReport C:\temp\AnyCompany.sql

This creates an assessment report named **MyFirstReport**, containing the analysis for SQL/DDL script **AnyCompany.sql**.
When the report is created, BabelfishCompass will automatically:

1. Open an explorer window in the directory where the report files are stored
2. Open the generated assessment report in the default browser

The full pathname of the report file is also printed to **stdout**.

Many additional command-line options can be used, such as processing multiple input scripts (for one application or multiple applications), generating more detailed output reports. See below for details.

Running Babelfish Compass (Mac/Linux)

On Mac/Linux, Babelfish Compass is invoked by opening a **bash** command prompt session in the directory where Babelfish Compass is installed.
At the bash command prompt, run **BabelfishCompass.sh:**
$ ./BabelfishCompass.sh -help

The automatic opening of the generated report in the browser, as on Windows, may not currently work on Mac/Linux. In this case, the user should explicitly open the generated report in a browser themselves.

Otherwise, Babelfish Compass is operated as on Windows. For further details, please see the rest of this User Guide, which assumes a Windows environment.

Reports, applications & input files

The Babelfish Compass tool is based on the concept of a report – the user must specify a report name which can be chosen freely. A report is the result of analyzing one or more SQL/DDL scripts. In the simplest case, a single SQL/DDL script is analyzed. Multiple input scripts, as well as combining analysis for multiple applications, are also supported.

Each input script is associated with an application name. By default, the application name is taken from the input script file name, e.g. a script named **Accounts.sql** is assumed to be for application **Accounts**. The application name can also be explicitly specified with the **-appname** flag.
A report can cover multiple input scripts for the same application, as well as multiple scripts for different applications. Examples:

-- single input file for application Accounts:
BabelfishCompass MyReport C:\temp\Accounts.sql

-- single input file for application Sales
BabelfishCompass MyReport C:\temp\ddl.20210913.sql -appname Sales

-- multiple input files for application Sales
BabelfishCompass MyReport C:\temp\ddl.20210913*.sql -appname Sales

-- multiple input files for applications Accounts, Sales and HR
BabelfishCompass MyReport C:\temp\Accounts.sql C:\temp\Sales.sql C:\temp\HR.sql

When a report is created for multiple applications, the assessment report can optionally indicate which applications contribute to a particular line item. To indicate this, use **-reportoption apps**. The report will contain lines like the following. This means 45 cases of the **SOUNDEX()** built-in function were found, in three applications as indicated:

SOUNDEX() : 45 #apps=3: Accounts(16), Support(20), HR(9)

Report directory location

A report is by default created in the following location:

- **C:\Users***username***\Documents\BabelfishCompass** (on Windows)
- **/Users/***username***/BabelfishCompassReports** (on Mac)
- **/home/***username***/BabelfishCompassReports** (on Linux)

A report is created as a directory underneath this location, e.g. for report MyReport , the directory **C:\Users***username***\Documents\BabelfishCompass\MyReport** will be created on Windows; the actual report is an **.html** file in this directory.

Technically, the location of the report root directory is determined by the value of **System.getProperty("user.home")** in Java. The locations shown above are typical defaults. It is currently not possible specify a different, user-defined location.

Specifying the Babelfish version

By default, Babelfish Compass delivers a compatibility assessment for the most recent version of Babelfish, as indicated in the **BabelfishFeatures.cfg** file. It is possible to perform the assessment for an earlier version of Babelfish by specifying the older version, e.g. **-babelfish-version 1.0.3**
The initial GA version of Babelfish is version 1.0.0. Since no older version exists at the time of the initial release, this option will only be available with later Babelfish releases.

Command-line options

To display all command-line options, run **BabelfishCompass -help**. All command-line options are optional.

- **-version**: displays the version of the Babelfish Compass tool
- **-explain**: displays some high-level guidance on how to use the Babelfish Compass tool
- **-encoding <encoding>:** specifies the encoding of the input files, in case these are not ASCII or the default encoding (this default is shown by **-help**).
  When specifying **-encoding**, this encoding is applied to all input files. To process multiple input files with different encodings, import these separately (with **-add**) with the correct encoding for each input file.
  Unicode-formatted files with BOM bits are automatically detected and processed accordingly, so -encoding does not need to be specified.
  To see the supported encodings, run **-encoding help**
- **-babelfish-version <version>**: performs analysis for an older BBF version. See section 'Specifying the Babelfish version'
- **-add**: import an additional SQL/DDL script to an existing report; perform analysis and generate a report
- **-replace**: replace an already-imported SQL/DDL script in an existing report; perform analysis and generate a report
- **- delete**: for an already existing report, delete all files first before recreating it
- **-noreport**: perform analysis, but do not generate a report. This can be useful when multiple files are imported; without **-noreport**, a report would be generated after every imported file. To generate a report after importing all files, use **-reportonly**.
- **-reportfile**: specifies the filename for the report. This does not affect the directory where the report files are located. See the examples below.
- **-importonly**: import the SQL/DDL script, but do not perform analysis or generate a report. This can be useful when importing multiple files, as the analysis would otherwise be performed after every imported file.

- **-analyze**: performs analysis on imported files, and generates a report. This can be used after importing files with **-importonly**, or to re-run analysis on imported files in an earlier report (for example, when re-running the analysis when a later version of Babelfish has become available)

- (v.1.1)
  -nooverride: do not use classification/report group overrides from **BabelfishCompassUser.cfg**

- **-list**: displays the files/applications that have been imported for a report

- **-reportonly**: generate a report for already-imported and analyzed SQL/DDL scripts. The report name should be specified, and no input files can be specified. This option is useful to generate additional detailed assessment reports, for example with a cross-reference or additional filtering (see **-reportoption**).

- **-reportoption** <options>: specifies options for generating the final assessment report. Different options can be specified in a comma-separated list (without spaces), and/or by using multiple **-reportoption** flags. The cross-reference is not generated by default, as this potentially makes the assessment report very long.
  Possible options are:

    - **xref** or **xref=all**: generates two cross-references for all items that are marked as "not supported" or "review". One cross-reference is ordered by SQL feature, the other by objects for which such items were detected.
    - **xref=feature** or **xref=object** generates only the cross-reference by feature, or by object, respectively.
    - **status=**<status>: with **xref**, specifies the categories for which the cross-reference should be generated. Without this option, a cross-reference is generated only for items marked as "not supported" or "review". To generate a cross-reference for a different category, specify (for example) **status=supported** or **status=ignored**. With **status=all**, a cross-reference for all items is generated.
      Note that using this option can result in a longer assessment report.
    - **detail**: with **xref**, generates additional detail for a reported item. For example, when reporting an object which cannot be created, specifying **detail** will include the name of the object. The report may get significantly longer as a result.

- **filter=**<string>: with **xref**, only include items which match the specified string(case-insensitive). This can be useful when the generated cross-reference is very long, for example to cross-reference only specific items of interest. Note that the Summary section is not affected by this option.
- **linenrs=**<number>: with **xref**, define the maximum number of line numbers mentioned in the cross-reference before suppressing the rest and adding "**+** *NNN* **more**". By default, the maximum number is 10.
- **notabs**: with **xref**, the hyperlinks to the original SQL source code will open in the same browser window instead of in a new tab. By default, a hyperlink opens in a new tab. NB: For large SQL source files, it may take some time before the browser displays the desired line. If this takes too long, it is also possible to manually access the corresponding flat text file (same filename, but with a **.dat** suffix instead of **.html**).
- **apps**: when a report covers multiple applications, this option will show the which applications contribute to a particular line item in the Summary section. For example, the following means that 45 cases of the **SOUNDEX()** built-in function were found, in three applications as indicated:

  SOUNDEX() : 45 #apps=3: Accounts(16), Support(20), HR(9)
- **batchnr**: with **xref**, displays the location of an item as a combination of the batch number in the file, the starting line number of the batch in the source file, and the line number in the batch. By default, the location is shown as the line number in the source file.

- **-quotedid** {**on**|**off**}: sets QUOTED_IDENTIFIER at the start of each SQL/DDL script. Default = ON
- **-pgimport "**pg-connection-attributes**"**: Creates a database table **BBFCompass** in a PostgreSQL database, and loads all captured items into the table. This table can then be accessed with SQL queries for further processing (see section "Using -pgimport").
  The PostgreSQL connection attributes are specified as a comma-separated list as follows: **host,port,username,password,dbname** . The import is performed through a script created in the **captured** subdirectory. This script uses the PostgreSQL **psql** utility, which must be installed on your system, and in the PATH.
  Note that the password is not saved anywhere and not written to any file (also not to temporary files).
- **-pgimportappend**: with **-pgimport**, appends to an already-existing PostgreSQL table. Without this option, **-pgimport** will drop the table if it exists, before recreating it.

Examples

Generate a default report without cross-reference for application Sales:

BabelfishCompass MyReport C:\temp\Sales.sql

Generate a default report without cross-reference for application Sales,
deleting the report first if it already existed:

BabelfishCompass MyReport C:\temp\Sales.sql -delete

Generate a report for applications Accounts and Sales, with a cross-reference for all categories, including additional detail, and allowing up to 100 line numbers to be enumerated in the cross-reference:

BabelfishCompass MyReport2 C:\temp\account*.sql -appname Accounts -add -noreport

BabelfishCompass MyReport2 C:\temp\sales.sql -add -noreport

BabelfishCompass MyReport2 -reportoptions xref,status=all,detail,linenrs=100

Display all files/applications imported for MyReport2:

BabelfishCompass MyReport2 -list

Re-run analysis for an existing report, but specifically for Babelfish version 1.3.0 (assuming the latest version of Babelfish is later than 1.3.0):

BabelfishCompass MyReport3 -analyze -babelfish-version 1.3.0

Import all captured items into a PostgreSQL database table:

BabelfishCompass MyReport3 -pgimport "mybighost.anycompany.com,5432,bob,B!gbob72,mydb"

Generate a cross-reference report named : **C:\...\BabelfishCompass\MyReport4\MyApp.xref.html .**
(without **-reportfile**, the report file name would be something like **C:\...\BabelfishCompass\MyReport4\report-MyReport4-2021-Sep-13-21.22.23.html**):

BabelfishCompass MyReport4 C:\temp\MyApp.sql -reportfile MyApp.xref -reportoption xref

File handling

An assessment report is an HTML file located in the report directory:

- Windows: **%USERPROFILE%\BabelfishCompass**<report-name>

A flat text version of the report is available in the same directory as the HTML file; this text version is named identically, but ends in **.txt** instead of **.html**.
If desired, the report file itself may be renamed.

The report directory contains multiple subdirectories as described below. Users of Babelfish Compass should not rename or edit the files in these subdirectories, as future invocations of Babelfish Compass for this report may no longer work correctly (or at all):

- **imported**: contains a copy of the original SQL/DDL input scripts. These are stored to allow re-running the analysis at a later time, for example for a newer version of Babelfish. In case the original input files were using a specific encoding, the files in the imported directory are in UTF8 format.
  For each imported file, an HTML version is also located in this directory. When generating a cross-reference in the assessment report, hyperlinks are generated to the actual line in the original document where the SQL feature was found.
- **imported\sym**: files in this directory contain symbol table information, for internal use.
- **captured**: files in this directory contain items that were captured during analysis. These are SQL features and options, which are reflected in the assessment report. The files in this directory can be imported into a PostgreSQL database table using the **-pgimport** option
- **log**: contains the session log file for each invocation of Babelfish Compass.
- **errorbatches**: this directory is created only when syntax errors were found in the imported SQL/DDL scripts. In this case, the input batches with the errors are saved in a file so that the user has access to this information. If desired, the user can rename or delete these files as they are not used as input for any further processing steps.

The BabelfishFeatures.cfg file

The compatibility assessment by the Babelfish Compass tool is driven by the file **BabelfishFeatures.cfg**, which is located in the Babelfish Compass installation directory. This file contains definitions of features that are (not) supported in a specific Babelfish version.

For each Babelfish release containing changes in functionality, a new version of the **BabelfishFeatures.cfg** will also be released. When Babelfish Compass is already installed, the existing version of **BabelfishFeatures.cfg** should be replaced (overwritten) by the newer version of this file.

Users of Babelfish Compass should not edit, modify or rename the **BabelfishFeatures.cfg** file: Babelfish Compass will detect this and terminate immediately.

SQL feature classifications

The general principle behind **BabelfishFeatures.cfg** is that features which are not listed in this file are supported by Babelfish. Features that are not supported may fall in either of these categories:

- **Not Supported** : the feature is currently not supported by Babelfish.
- **Review Semantics** : the feature involves aspects which cannot be addressed by Babelfish, but requires the user to determine whether or not it requires changes to be made as part of the migration process.
- **Review Performance** : the feature involves a performance-related aspect in SQL Server, and therefore the user should review this to determine whether the performance aspect may be relevant when running on Babelfish.
- **Review Manually** : the feature cannot de assessed by Babelfish Compass, but needs to be reviewed by the user. Example: **SET LANGUAGE @v** : whether **@v** contains a Babelfish-supported language name cannot be determined by Babelfish Compass.
- **Ignored** : the feature is currently ignored by Babelfish

Example: BabelfishFeatures.cfg

This denotes that ALTER VIEW is not supported, and is reported in a group named 'Views':

```
[ALTER VIEW]
rule=create_or_alter_view
report_group=Views
```

This denotes that the only supported option for FETCH is FETCH NEXT; any FETCH options are reported in a group named 'Cursors':

```
[FETCH cursor]
rule=fetch_cursor
list=NEXT,PRIOR,FIRST,LAST,ABSOLUTE,RELATIVE
supported-1.0.0=NEXT
report_group=Cursors
```

For more information about the contents of **BabelfishFeatures.cfg**, see the file header.

The BabelfishCompassUser.cfg file (classification overrides)

As described in the previous section, the file **BabelfishFeatures.cfg** defines which features are (not) supported in a particular version of Babelfish.
In v1.1 of Babelfish Compass, for SQL features that are not supported, it is possible for the user to override the classification defined by **BabelfishFeatures.cfg.** For this purpose, Babelfish Compass generates a file named **BabelfishCompassUser.cfg,** which is located in the report root directory (see section "Report location" above), e.g.
C:\Users*username***\Documents\BabelfishCompass\BabelfishCompassUser.cfg** . This file can be edited by the user (unlike **BabelfishCompassUser.cfg**, which should not be modified by the user). Also, **BabelfishCompassUser.cfg** is not overwritten when installing a new version of Babelfish Compass, as opposed to **BabelfishFeatures.cfg**, which will always be replaced in a new version of Babelfish Compass**.**

It is not required to use **BabelfishCompassUser.cfg**, and it is not recommended for new users of Babelfish Compass. However, experienced users of Babelfish Compass may want to be able to tailor their assessment reports by putting more -or less focus- on specific SQL features, and this is what **BabelfishCompassUser.cfg** allows you to do.

BabelfishCompassUser.cfg contains all sections that are present in **BabelfishFeatures.cfg**, for example **[Datatypes]** or **[Built-in functions]**. The user should not modify these section headers, but can add certain items to a section, as described below.
Note that any modifications made by the user to **BabelfishCompassUser.cfg** will not be saved or stored by Babelfish Compass. The user should ensure that **BabelfishCompassUser.cfg** is properly backed up.

Babelfish Compass will **BabelfishCompassUser.cfg** when it does not exist. In case new sections have been defined in **BabelfishFeatures.cfg,** which are not yet in **BabelfishCompassUser.cfg**, then these sections will be appended. If the user manually deletes sections from **BabelfishCompassUser.cfg**, those section will be appended again the next time Babelfish Compass runs.

Note:

- User-defined overrides are applied during analysis, and any overridden values are recorded in the captured items; the assessment report is generated from these captured items. When only generating a report (e.g. with **-reportonly**), no overrides will be applied.
  When the user has modified override entries in **BabelfishCompassUser.cfg** and wants to apply this to a report, the **-analyze** flag should be used.
- When a user-defined override is applied, the captured items will reflect the values after the overrides have been applied; the original values have been lost. This means that it is not possible to determine for individual captured items whether an override was applied (for example, after using **-pgimport**).

Example: overriding default classification and reporting group

By default, CLUSTERED indexes and constraints are classified as 'Review Semantics' by Babelfish Compass. But in case you decide you don't care about those aspects, and you want these to be ignored in the assessment report, the classification can be overridden in **BabelfishCompassUser.cfg** by adding **default_classification=Ignored** :

```
[CLUSTERED index]
default_classification=Ignored
```

Likewise, the FORMAT() and STR() functions are not supported in Babelfish Compass v.1.0.0, and will be reported accordingly. However, in case you want these functions to be classified as 'Review Manually' and reported under 'Formatting functions', then add the following lines to the section **[Built-in functions]** in **BabelfishCompassUser.cfg** :

```
[Built-in functions]
default_classification-ReviewManually=FORMAT,STR
report_group-Formatting functions=FORMAT,STR
```

Note that these changes only affect how SQL features are classified in the Babelfish Compass report: there is no impact on how Babelfish itself processes the SQL features for which you changed the classification.

For more information about possible modifications that the user can make to **BabelfishCompassUser.cfg**, see the file header.

Using -pgimport

As described earlier, the **-pgimport** flag lets you load all captured items into a PostgreSQL table. From here, you can perform customized additional operations on this data. Before you can use **-pgimport**, the PostgreSQL utility **psql** needs to be installed on your system, and needs to be in the PATH.

Examples of what you may be able to do with **-pgimport**:

- Run SQL queries to find objects with a complex combination of attributes. For example: find all SQL functions with at least two parameters, including a MONEY-type parameter, a SMALLDATETIME result type, and a table variable operation in the function body.
- The Babelfish Compass assessment report deliberately does not report any 'compatibility percentage', because it is difficult to define such a number in a meaningful way. A simple way to calculate such a percentage would be to take the ratio of non-supported features vs. supported features. However, some unsupported features may be very difficult to work around while other may be easy. Yet, they would both weigh equally heavy in such a calculation.
  Users of Babelfish Compass can decide for themselves how they would want to calculate a compatibility percentage. For example, a user could write a SQL-based application that assigns different weights to different non-supported features, thus calculating a more realistic compatibility percentage on the basis of the captured items that were loaded with **-pgimport**.
  NB: any such calculations will be the exclusive responsibility of the Babelfish Compass user.
- When a migration opportunity is discussed, a key question to is to estimate the time and cost of performing a migration. While this question is realistic, the Babelfish Compass tool does not attempt to make any estimates with respect to the amount of time or effort it may require to address the non-supported issues that were identified. The reason is that the actual effort required will be highly dependent on skills and experience of the individuals doing the actual work (picture a team of seasoned DBAs with decades of database experience vs. a team of newly arrived university graduates). Since it is not realistic to generalize such effort estimates, Babelfish Compass does not attempt this.
  However, users of Babelfish Compass could try to build such functionality themselves on the basis of the captured items that were loaded with **-pgimport**. Imagine an experienced team of migration experts who have collected detailed data points from their past migration projects: such a team might be able to quantify the effort required for the non-supported items in the Babelfish Compass assessment report, specifically aimed at their own team with their specific experience. The **-pgimport** function makes it possible to build an application using the imported items for making effort estimates.
  NB: any such estimates will be the exclusive responsibility of the Babelfish Compass user.

Schema for imported items

When using the **-pgimport** flag, a PostgreSQL table is created as follows:

```
CREATE TABLE BBFCompass(
     babelfish_version VARCHAR(20) NOT NULL,
     date_imported TIMESTAMP NOT NULL,
     item VARCHAR(200) NOT NULL,
     itemDetail VARCHAR(200) NOT NULL,
     reportGroup VARCHAR(50) NOT NULL,
     status VARCHAR(20) NOT NULL,
     lineNr INT NOT NULL,
     appName VARCHAR(100) NOT NULL,
     srcFile VARCHAR(200) NOT NULL,
     batchNrinFile INT NOT NULL,
     batchLineInFile INT NOT NULL,
     context VARCHAR(200) NOT NULL,
     subcontext VARCHAR(200) NOT NULL,
     misc VARCHAR(20) NOT NULL
);
```

The various columns represent the following:

- babelfish_version : Babelfish version for which analysis was performed
- date_imported : date/time of running -pgimport
- item : line item as shown in the report
- itemDetail : additional info for a line item
- reportGroup : report group as show in the report
- status : classification of the item, e.g. SUPPORTED, NOTSUPPORTED, etc.
- lineNr : line number of the item in the T-SQL batch
- appName : application name
- srcFile : SQL source file name
- batchNrinFile : batch no. of T-SQL batch in SQL source file
- batchLineInFile : line number in file of start of batch
- context : name of object, or 'T-SQL batch'
- subcontext : (optional) name of table in object
- misc : not for customer use; ignore

Example query

Using the schema above, this is an example of a query against the imported items:

"find all SQL functions with at least two parameters, including a MONEY-type parameter, a SMALLDATETIME result type, and a table variable operation in the function body"

```
select distinct context from BBFCompass
-- filter on table variable operation:
where item like '% @tableVariable'
-- filter on function with >= 2 parameters:
and context in (
      select context from BBFCompass
      where context like 'FUNCTION %'
        and item like '% parameter'
      group by context
      having count(*) >= 2)
-- filter on MONEY-type parameter:
and context in (
      select context from BBFCompass
      where context like 'FUNCTION %'
      and item like 'MONEY %function parameter%')
-- filter on function result type:
and context in (
      select context from BBFCompass
      where context like 'FUNCTION %'
        and item like 'SMALLDATETIME %scalar function result%')
```

NB: On large tables, performance may benefit from adding indexes to one or more columns of this table. This is left to the user to explore.

Security

The Babelfish Compass tool is a stand-alone, on-premises program which does not store any confidential or sensitive information: all information stored is derived from the SQL/DDL scripts which the user provided as input.

The Babelfish Compass tool operates offline and does not perform any network access (with the exception of the **-pgimport** option, see below).
The Babelfish Compass tool does not "phone home" and makes no network connections invisible to the user. For example, it does not perform a "check for updates": installing an update must be performed manually by the user.

The -pgimport option

The **-pgimport** option is the only function where network access takes place, by connecting to a PostgreSQL instance and loading captured items into a database table.
Technically, Babelfish Compass creates files **pg_import.bat** (on Mac/Linux, **pg_import.sh**) and **pg_import.psql** in the **captured** directory. The **pg_import.bat**/**pg_import.sh** file executes **pg_import.psql**, which runs a CREATE TABLE and a COPY statement in PostgreSQL.
Babelfish Compass executes this function by spawning a subprocess to run **pg_import.bat**/**pg_import.sh**.
To make a connection to the PostgreSQL instance, the user must specify connection attributes on the Babelfish Compass command line, including the PostgreSQL username and password. These connection attributes are not written to any file, but are supplied as environment variables in the short-lived spawned subprocess. These environment variables are not accessible from outside the spawned subprocess.
Note that the connection attributes may be accessible through the command-line history in the command-line session that runs Babelfish Compass.
As for the uploaded captured items, it is assumed that the user owns the PostgreSQL instance and is responsible for granting access to the uploaded data.

Using Babelfish Compass in PostgreSQL Migrations

Users of SQL Server can use Babelfish Compass to analyze the SQL/DDL code for their current SQL Server-based applications for compatibility with Babelfish. The purpose of such analysis is to inform a Go/No Go decision about whether it makes sense -or not- to consider starting a migration project from SQL Server to Babelfish. For this purpose, Babelfish Compass produces an assessment report which lists -in great detail- all SQL features found in the SQL/DDL code, and whether or not these are supported by the latest version of Babelfish.

On a high level, the sequence of steps to be taken is as follows:

1. The application owner identifies the SQL Server databases required for the application that is considered for migration to Babelfish. The application owner is recommended to ensure there are no legal restrictions with respect to migrating the application in question.
2. Reverse-engineer the SQL Server database(s) in question with SQL Server Management Studio (SSMS). This is done in the SSMS object explorer by right-clicking a database. Then select **Tasks ➔ Generate Scripts**, and follow the dialog (make sure to turn on triggers, collations, logins, owners and permissions (these turned off in SSMS by default), by clicking the 'Advanced' button and turning on the respective options).
    - Babelfish Compass requires its input scripts to be syntactically valid T-SQL, using **go** as a batch delimiter (i.e. **sqlcmd**-style scripts). Some tools may be able to reverse-engineer, but not do this correctly or completely, or not generate the required batch delimiters (like DBeaver). Therefore, it is recommended to use SSMS for generating a DDL script of your database(s).
3. SSMS produces a DDL/SQL script as output. Use this script (one or more) as input for Babelfish Compass to generate an assessment report (see instructions and examples earlier in this User Guide).
4. Optionally, generate additional cross-reference reports to obtain additional details about the unsupported features.
5. Discuss the results of the Babelfish Compass assessment with the application owner and interpret the findings in the context of the application to be migrated. In these discussions, it may be possible to descope the migration by identifying outdated or redundant parts of the application which do not need to be migrated.
6. The owner of the application to be migrated needs to take a decision whether, given the assessment results that show the unsupported SQL features in the SQL/DDL code, it is opportune to start a migration project to Babelfish. If the current version of Babelfish is

deemed to be insufficiently compatible with the application in question, it is recommended to re-run the analysis when future releases of Babelfish are available which will provide more functionality.

7. When proceeding, modify the SQL/DDL scripts to rewrite or remove the SQL/DDL statements that were reported as 'not supported' or requiring 'review'. Then execute the SQL/DDL script against Babelfish (with **sqlcmd**) to recreate the schema in Babelfish.
8. Finally, perform data migration, and reconfigure the client applications to connect to Babelfish.

Please keep the following in mind:

- Admittedly, the amount of detail in a Babelfish Compass assessment report can be large. When discussing the Babelfish Compass findings with the application owner, make sure to highlight the many aspects that are supported by Babelfish: experience has shown that when focusing primarily on the non-supported features, SQL Server users may easily end up with an unnecessary negative perception of Babelfish's capabilities.
- A Babelfish migration involves more than just the server-side SQL/DDL code, for example, interfaces with other system; ETL/ELT; SSIS/SSRS, replication, etc. These aspects may not be reflected in the server-side view provided by Babelfish Compass.

Troubleshooting

This section contains some troubleshooting tips. It is recommended to first read this User Guide in detail when encountering unexpected behavior.

- Syntax errors: while the SQL/DDL input scripts are reverse-engineered from existing applications and their contents are therefore assumed to contain syntactically valid SQL code, it is possible that the SQL code contains syntax errors, for example as a result of manual editing (one example: it is mandatory to terminate the MERGE statement with a semicolon, while such a terminator is optional for most other T-SQL statements).
  In case of a syntax error, this will be printed to **stdout**, and the offending batch will be logged in a file in the **errorbatches** subdirectory in the report directory. A batch containing syntax errors is not analyzed by Babelfish Compass.
  Remedy: correct any SQL syntax errors and re-run the script through the Babelfish Compass tool using the **-replace** flag.

- If syntax errors are printed which show garbage characters, it may be that the input file encoding is not correctly specified. The default encoding , and all available encodings, are displayed with the **-encoding help** option.
  Remedy: specify the correct encoding with **-encoding** on the command line.

Licensing

GitHub: https://github.com/babelfish-for-postgresql/babelfish_compass