# Babelfish Compass: User Guide

Document history:

Feb-2022: v.2022-02: new Compass version numbering
Jan-2022: added compatibility matrix with Babelfish
Dec-2021: v.1.2: added **-rewrite** option
Nov-2021: correct typo in section about **BabelfishCompassUser.cfg**; edit for grammar
Nov-2021: v.1.1: added user-definable overrides, example for **-pgimport**, Mac/Linux support
Oct-2021: v.1.0: 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.

You can use Babelfish Compass to analyze the SQL/DDL code for your current SQL Server-based applications for compatibility with Babelfish. The purpose of this analysis is to gather information so you can make a Go/No Go decision about starting a migration project from SQL Server to Babelfish. For this purpose, Babelfish Compass produces an assessment report which lists -in great detail- all of the SQL features found in your 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 each 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.

# Compatibility with Babelfish for PostgreSQL

The Babelfish Compass tool supports the following Babelfish versions.
In principle, any version of Babelfish Compass will support whichever Babelfish version the **BabelfishFeatures.cfg** file has been updated for. However, a full version of Babelfish Compass, also including fixes and enhancements, will in principle be released for every Babelfish release.

| **Babelfish Compass version** | **Babelfish version** |
|---|---|
| 2022-02 | 1.1.0, 1.0.0 |
| 1.2 | 1.1.0, 1.0.0 |
| 1.0, 1.1 | 1.0.0 |

In February 2022, Babelfish Compass changed to a different version numbering schema (YYYY-MM) to avoid confusion with Babelfish version numbers. Compass version 1.2 was followed by version 2022-02.

# Installing Babelfish Compass

## Prerequisites

Before installing Babelfish Compass, you must install a Java Runtime Environment (JRE) version 8 or higher (64-bit version).

Babelfish Compass produces compatibility assessment reports in HTML format. To view the HTML output, we recommend using a recent release of the Google Chrome or Mozilla Firefox browser.

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 an open-source project at https://github.com/babelfish-for-postgresql/babelfish_compass.

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

## Installation

Babelfish Compass is distributed as an executable JAR file, 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 as detailed in the previous section.
2. Unzip the file so that the contents are placed in your installation directory of choice; this document will assume the file resides in **C:\BabelfishCompass.**
    a. If a previous installation is already present in your installation directory, you can overwrite the installation (but we recommend you make a backup copy first).
3. Installation is complete.

Installation steps on Mac/Linux:

1. Download the **BabelfishCompass.zip** file as detailed in the previous section.
2. Unzip this file so that the contents are placed in your 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 placed, and the reports should be kept separate.

b. If a previous installation is already present in your installation directory, you can overwrite the installation (but we recommend you make a backup copy first).

3. Verify the **BabelfishCompass.sh** shell script is executable by running **./BabelfishCompass.sh** . If it is not executable, run the command: **chmod +x BabelfishCompass.sh** .
4. Installation is complete.

# Running Babelfish Compass on Windows

To run Babelfish Compass on Windows, open a **cmd** prompt (a "DOS box") and navigate to the Babelfish Compass installation directory.

Then, select the command line options that you need to include when invoking Babelfish Compass. The command line options are detailed in the Command-line options section of this guide, or you can review them on the command line by running:

**C:\BabelfishCompass> BabelfishCompass**[**.bat**] **-help**

Then, invoke **BabelfishCompass**[**.bat**] with your choice of command-line options.

Babelfish Compass usage typically starts by 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 usage case, a single SQL/DDL script is analyzed. To analyze a single script, simply specify a report name and an input file with your call to Babelfish Compass. For example:

**C:\BabelfishCompass> BabelfishCompass**[**.bat**] **MyFirstReport C:\temp\AnyCompany.sql**

This command creates an assessment report named **MyFirstReport**, containing the analysis for SQL/DDL script **AnyCompany.sql**.

When a 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.
3. Print the full pathname of the report file to **stdout**.

There are many additional command-line options you can include that support functionality to process multiple input scripts (for one application or multiple applications), generate more detailed output reports, and so on.  See Command-line options for details.

# Running Babelfish Compass (Mac/Linux)

To run Babelfish Compass on Linux, open a **bash** command prompt  and navigate to the Babelfish Compass installation directory.

Then, select the command line options that you need to include when invoking Babelfish Compass. The command line options are detailed in the Command-line options section of this guide, or you can review them on the command line by running:

```
$ ./BabelfishCompass.sh -help
```

Then, invoke **BabelfishCompass.sh** with your choice of command-line options.

Babelfish Compass usage typically starts by 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 usage case, a single SQL/DDL script is analyzed. To analyze a single script, simply specify a report name and an input file with your call to Babelfish Compass.  For example:

```
$ ./BabelfishCompass.sh  MyFirstReport  /tmp/AnyCompany.sql
```

This command creates an assessment report named **MyFirstReport**, containing the analysis for SQL/DDL script **AnyCompany.sql**.

When a report is created, BabelfishCompass will automatically:

1. Open a file browser in the directory where the report files are stored.
2. Open the generated assessment report in the default browser. Please note that on Linux, the browser will not open automatically; instead, simply open the file manually.
3. Print the full pathname of the report file to **stdout**.

There are many additional command-line options you can include that support functionality to process multiple input scripts (for one application or multiple applications), generate more detailed output reports, and so on.  See Command-line options for details.

# Reports, applications, and input files

The Babelfish Compass tool generates a report with a user-specified report name. The report is the result of analyzing one or more SQL/DDL scripts. In the simplest case, a single SQL/DDL script is

analyzed. Babelfish Compass also supports combined analysis of multiple input scripts and multiple applications.
Each input script is associated with an application name. By default, the application name is taken from the input script file name. For example, a script named **Accounts.sql** is created for an application named **Accounts**. You can specify the application name with the **-appname** flag. A report can cover multiple input scripts for the same application, as well as multiple scripts for different applications.

**Examples:**

The following command generates a report for a single input file, with an application named **Accounts**:

**BabelfishCompass MyReport C:\temp\Accounts.sql**

The following command generates a report for a single input file, with an application named **Sales**:

**BabelfishCompass MyReport C:\temp\ddl.20210913.sql -appname Sales**

The following command generates a report for multiple input files, with an application named **Sales**:

**BabelfishCompass MyReport C:\temp\ddl.20210913*.sql -appname Sales**

The following command generates a report for multiple input files, with applications named **Accounts**, **Sales** and **HR:**

**BabelfishCompass MyReport C:\temp\Accounts.sql C:\temp\Sales.sql C:\temp\HR.sql**

When you create a report for multiple applications, the assessment can optionally indicate which applications contribute to a particular line item. To include this content, specify the **-reportoption apps** option. The report will contain lines in the following format:

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

This means 45 cases of the **SOUNDEX()** built-in function were found, in three applications as indicated:

# Report directory location

By default, the Babelfish Compass report is 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 **.html** file in a directory below this location. For example, on Windows, Babelfish Compass creates a report named **MyReport.html** in the following directory: **C:\Users***username***\Documents\BabelfishCompass\MyReport**.

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 instead.

# 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. You can perform an assessment for an earlier version of Babelfish by specifying the older version with the **-babelfish-version** option; for example:

**-babelfish-version 1.3.0**

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 valid with later Babelfish releases.

# Command-line options

To display all of the command-line options, run **BabelfishCompass -help**. Note that 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, if the files are not ASCII or the default encoding (this default is shown by **-help**).
  The specified **-encoding** is applied to all input files. To process multiple input files with different encodings, import each file separately (with **-add**), specifying 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 review a list of supported encodings, run **-encoding help**

- **-babelfish-version** <*version*>: performs the analysis for an older BBF version. See Specifying the Babelfish version for more information.

- **-add**: imports an additional SQL/DDL script to an existing report, performs an analysis, and generates a report.

- **-replace**: replaces an already-imported SQL/DDL script in an existing report, performs an analysis, and generates a report

- **- delete**: deletes all the files for an already existing report before recreating it.

- **-noreport**: performs an analysis without generating a report. This is useful when multiple files are imported; without **-noreport**, a report will be generated after every imported file. To generate a report after importing all files, include the **-reportonly** option.

- **-reportfile**: specifies the filename for the report. This does not affect the directory where the report files are located. See the Examples.

- **-importonly**: imports the SQL/DDL script, but does not perform an analysis or generate a report. This can be useful when importing multiple files, as the analysis will otherwise be performed after every imported file.

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

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

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

- **-reportonly**: generates a report for already-imported and analyzed SQL/DDL scripts. Specify a report name; do not specify input files. This option is useful when generating 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. Specify different options 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.
  Warning: for large schemas, the report generated with **xref** (and even more so when combined with **status=all**), may become very large and may take longer to load in your browser. For this reason, the **xref** option is off by default, and you have to specify it explicitly with **-reportoption**.
- **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 includes 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**, defines 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**, opens the hyperlinks to the original SQL source code 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**: shows which applications contribute to a particular line item in the Summary section when a report covers multiple applications. 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 is ON
- **-pgimport "**pg-connection-attributes**"**: creates a database table named **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 Using -pgimport).
  The PostgreSQL connection attributes are specified in 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 your PATH.
  Note that the password is not saved anywhere and not written to any file (including temporary files).
- **-pgimportappend**: with **-pgimport**, appends content 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-references for an application named **Sales**:

**BabelfishCompass MyReport C:\temp\Sales.sql**

Generate a default report without cross-reference for an application named **Sales**,
deleting the report directory first if it already exists:

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

Generate a report for applications named **Accounts** and **Sales**, cross-referencing 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 and applications imported for MyReport2:

**BabelfishCompass MyReport2 -list**

Re-run an analysis for an existing report, but specifically for Babelfish version 1.3.0 (this example assumes 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-referenced report named : **C:\...\BabelfishCompass\MyReport4\MyApp.xref.html**. (without the **-reportfile** option, 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**

# Automatic rewriting of unsupported features

As of version 1.2 (or later) of Babelfish Compass, you can use the **-rewrite** option to address certain SQL features which are not supported by Babelfish, by rewriting the SQL feature in question in such a way that Babelfish is able to process. One example is the MERGE statement.

- When not specifying the **-rewrite** option, the assessment report will include a section "**Automatic SQL Rewrite Opportunities**" which lists the SQL features that could be addressed with **-rewrite**, but without actually rewriting them.
- When specifying the **-rewrite** option, Babelfish Compass creates a subdirectory **rewritten** in the report directory, containing a copy of the original SQL source file in which specific features have been rewritten (if nothing is rewritten, no copy will be created in **rewritten**).
  The assessment report will contain a section with the specific rewritten features.
  When **-reportoption xref** is used, the cross-reference links in the 'rewritten' sections point to the rewritten SQL file (instead of to the original SQL file).

In a rewritten SQL file, the bottom of the file has a list of all changes made by Babelfish Compass.

When using the **-rewrite** option, you should execute the rewritten SQL file against Babelfish instead of the original SQL file.

Note that using **-rewrite** may cause Babelfish Compass to run slower than without **-rewrite**, especially for large files in which many features are rewritten.

# File handling

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

- On 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**.

The report directory contains multiple subdirectories as described below. You 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). If the original input files used 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**: contains files with symbol table information, for internal use.
- **captured**: contains files that 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**: is a directory 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, you 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 performed 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.

Don't edit, modify, or rename the **BabelfishFeatures.cfg** file; Babelfish Compass will detect changes, 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 review 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 you should review this carefully to determine if performance may be impacted when running on Babelfish.
- **Review Manually** : the feature cannot be assessed by Babelfish Compass, but needs to be manually examined. For example: **SET LANGUAGE @v** : Babelfish Compass cannot determine if **@v** contains a Babelfish-supported language name.
- **Ignored** : the feature is currently ignored by Babelfish.

## Example: BabelfishFeatures.cfg

The following example 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
```

The following example 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 **BabelfishFeatures.cfg** file defines which features are or are not supported in a particular version of Babelfish. In version 1.1 (or later) of Babelfish Compass, for SQL features that are not supported, you can 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 Report directory location for more information. The default location of this file is **C:\Users***username***\Documents\BabelfishCompass\BabelfishCompassUser.cfg**. You can edit this file (unlike **BabelfishCompass.cfg**, which should not be modified by the user). **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**.**

Use of the **BabelfishCompassUser.cfg** file is not recommended for new users of Babelfish Compass. However, if you are an experienced user, you can use **BabelfishCompassUser.cfg** to tailor your assessment reports by putting more or less focus on specific SQL features.

**BabelfishCompassUser.cfg** contains all of the sections that are present in **BabelfishFeatures.cfg**, like **[Datatypes]** or **[Built-in functions]**. You shouldn't modify these section headers, but can add certain items to a section, as described below.
Note that any modifications made to the **BabelfishCompassUser.cfg** will not be saved or stored by Babelfish Compass. Ensure that **BabelfishCompassUser.cfg** is properly backed up.

Babelfish Compass will create the **BabelfishCompassUser.cfg** file if it does not exist. If new sections have been defined in **BabelfishFeatures.cfg,** which are not yet in **BabelfishCompassUser.cfg**, the new

sections will be appended. If you manually delete 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. If 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 version 1.0.0, and will be reported accordingly. If 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 you 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 psql client 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.
  You can decide how to calculate a viable compatibility percentage for your evaluation. For example, you 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**.
  Note: any such calculations are the exclusive responsibility of the Babelfish Compass user.
- When a migration opportunity is discussed, a key question 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, you could try to build such functionality yourself 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.
  Note: any such estimates are the exclusive responsibility of the Babelfish Compass user.

# Schema for imported items

When you include the **-pgimport** flag, Babelfish Compass creates a PostgreSQL table with the following definition:

```
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 columns represent the following:

- babelfish_version : is the Babelfish version for which analysis was performed.
- date_imported : is the date/time that -pgimport ran.
- item : is a line item as shown in the report.
- itemDetail : is additional info about a line item.
- reportGroup : is the report group as show in the report.
- status : is the classification of the item; for example, **SUPPORTED** or **NOTSUPPORTED**.
- lineNr : is the line number of the item in the T-SQL batch.
- appName : is the application name.
- srcFile : is the SQL source file name.
- batchNrinFile : is the batch number of the T-SQL batch in SQL source file.

- batchLineInFile : is the line number in the file at the start of the batch.
- context : is the name of an object, or T-SQL batch.
- subcontext : is the (optional) name of a table in the object.
- misc : is not for customer use; ignore this field.

## Example query

You can run SQL queries against the imported items to locate specific information. For example, to find this information:

> *"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"*

…use this SQL query:

```
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%')
```

Note: On large tables, performance may benefit from adding indexes to one or more columns of this table. This is left for 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 provides 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". Updating the installation 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 to migrate to PostgreSQL

Babelfish Compass analyzes the SQL/DDL code for a SQL Server-based application for compatibility with Babelfish. The purpose of this analysis is to inform a Go/No Go decision about 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 involved in a migration 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 must 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 and selecting **Tasks ➔ Generate Scripts**, and following the dialog (making sure to turn on triggers, collations, logins, owners and permissions (turned off in SSMS by default), by clicking the **Advanced** button and turning on the respective options).
    - Babelfish Compass requires 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 don't do this correctly or completely, or don't generate the required batch delimiters (like DBeaver). Therefore, we recommend using SSMS to generate a DDL script of the database(s).
3. SSMS produces a DDL/SQL script as output. Use this script (or scripts) 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 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. Use the assessment results that show the unsupported SQL features in the SQL/DDL code, to decide if 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, we

recommended you re-run the analysis when future releases of Babelfish are available which will provide more functionality.

7. If proceeding with a migration, modify the SQL/DDL scripts to rewrite or remove the SQL/DDL statements that are reported as **not supported** or **requiring review**. Then, invoke the SQL/DDL script against Babelfish (with **sqlcmd**) to recreate the schema in Babelfish.
8. Finally, perform a 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 an 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. If you encounter unexpected behavior by Babelfish Compass, we recommend you first read this User Guide in detail.

- Syntax errors: while the SQL/DDL input scripts are reverse-engineered from existing applications and their contents are assumed to contain syntactically valid SQL code, it is possible that the SQL code contains syntax errors. A syntax error might be the result of manual editing or inconsistencies. For example, MERGE statements must be terminated 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 of 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 that 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