Babelfish Compass: User Guide

Document history:

Jul-2024	v.2024-07: added **-anon** option
Apr-2024	v.2024-04: small enhancements, incl. for **-rewrite**
Nov-2023	v.2023-11: added sections on scaling and advanced features; additional functionality for effort estimation
Oct-2023	v.2023-10: added object dependency tracking with **-pgimport**
Aug-2023	v.2023-08: additional functionality for effort estimation; complexity defaults
Jun-2023	v.2023-06: added automatic DDL generation; expanded scope of **-optimistic** option
Mar-2023	v.2023-03: added predefined 'optimistic' user-defined **.cfg** file and **-optimistic** option
Dec-2022	v.2022-12: added executive summary, analysis of dynamic SQL, reporting of complexity score for not-supported items, **-userconfigfile** option
Oct-2022	v.2022-10: added support for processing queries captured through extended events
Sep-2022	v.2022-09: generating **.csv** file for complexity/effort estimation
Jun-2022	v.2022-06-a: added **-importfmt** option to process captured query files; various report enhancements
Jun-2022	v.2022-06: added **-pgimporttable** option
Apr-2022	v.2022-04: added automatic check for new Compass version and **-noupdatechk** option
Mar-2022	v.2022-03: added **-recursive**, **-include**, **-exclude** options
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

Document location

This document is located at https://raw.githubusercontent.com/babelfish-for-postgresql/babelfish_compass/main/BabelfishCompass_UserGuide.pdf .

Contents

Licensing 3
What Is Babelfish Compass? 4
Compatibility with Babelfish for PostgreSQL 5
Installing Babelfish Compass 6
Prerequisites 6
Downloading Babelfish Compass 6
Installation 6
Running Babelfish Compass on Windows 7
Running Babelfish Compass (Mac/Linux) 8
Reports, applications, and input files 9
Report root directory location 9
Specifying the Babelfish version 10
Command-line options 10
Examples 15
Automatic rewriting of unsupported features 16
Files & Directories for a Report 18
The BabelfishFeatures.cfg file 19
SQL feature classifications 19
Example: BabelfishFeatures.cfg 20
The BabelfishCompassUser.cfg file (classification overrides) 21
Example: overriding default classification and reporting group 22
Predefined 'Optimistic' .cfg file 22
User-defined estimates & .csv file 24
'Flat' format for .csv file 24
Default complexity score values 25
Complexity score estimates 25
Complexity score defaults 26
Effort estimates 26
Real-life effort estimate in executive summary 27
Examples 28
Effort estimate defaults 28
Uploading details into PostgreSQL with -pgimport 29
Schema for imported items 30
Example queries 31
Processing captured SQL queries 33
SQL Server Profiler 33
SQL Server Extended Events 33
Examples 34
Automatic DDL generation 35
Command-line options 35
How it works 36
Generating only DDL 36
Example 37
Security 39
The -pgimport option 39
The -sqlpasswd option 39
The -anon option 40

Automatic update check 40
Scaling Compass: Analyzing Multiple Applications 41
Scaling Compass: Analyzing Many Applications 42
Improvement 1: Generate DDL in batch 42
Improvement 2: Capture 1-line metrics per report 43
Advanced Compass Usage 44
Manually copying imported files 44
A single Compass report for very large amounts of SQL 44
Locating items in very large SQL scripts when cross-ref links are slow 45
Using Babelfish Compass to migrate to PostgreSQL 46
Troubleshooting 48
Example of BabelfishCompassItemID.csv 50

Licensing

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

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 published for every Babelfish release with added support for new T-SQL features.

Babelfish Compass version	**Supported Babelfish versions** (includes versions supported by earlier releases, per below, as well as patch releases)
2024-10	4.3.x (PG 16.4)
2024-07	4.2.x (PG 16.3)
2024-04	4.1.x (PG 16.2), 3.5.x (PG 15.6)
2024-02	4.0.x (PG 16.1)
2023-12[-a]	3.4.x (PG 15.5)
2023-10, 2023-11	3.3.x (PG 15.4)
2023-06, 2023-08	3.2.x (PG 15.3)
2023-03[-a]	3.1.x (PG 15.2)
2022-12	2.3.x (PG 14.6)
2022-09, 2022-10, 2022-11	2.2.x (PG 14.5)
2022-07, 2022-06[-a]	2.1.x (PG 14.3/14.4)
2022-03, 2022-04	1.2.x (PG 13.6)
2022-02	same as below
1.2 (see note on version numbering below)	1.1.x (PG 13.5)
1.0, 1.1	1.0.x (PG 13.4)

In February 2022, Babelfish Compass changed to a different version numbering schema (YYYY-MM) to avoid confusion with Babelfish version numbers. Consequently, Compass version 1.2 was followed by version 2022-02.
Note that there are no Babelfish version 2.0.x and 3.0.x.

Minor versions not listed here, such as 1.4, 1.5 etc. are patch releases on top of the latest minor release listed (e.g. 1.2).

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. Do not install Babelfish Compass into **C:\Users***username***\Documents\BabelfishCompass** since this is where the generated reports will be placed, and this location should be kept separate from the installation directory; Babelfish will not run when installed in this location.
    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. 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** (Linux) or **/Users/***username***/BabelfishCompass** (Mac)
    a. Do not install Babelfish Compass into **/home/***username***/BabelfishCompassReports** (Linux) or **/Users/***username***/BabelfishCompassReports** (Mac)**,** since this is where the generated reports will be placed, and this location should be kept separate from the installation directory; Babelfish will not run when installed in this location.
    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 root directory location

By default, the Babelfish Compass report is created in the following location (the 'Compass report root' directory):

- **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 2.2.0

The initial GA version of Babelfish was version 1.0.0. This is the oldest version you can specify.
When the patch version (the third part in the version number) is omitted, '**.0**' is assumed: specifying **2.2** is equivalent to **2.2.0** .

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).

- **-userconfigfile** filename : as of v.2022-12, this specifies the user-defined **.cfg** file to be used (default = **BabelfishCompassUser.cfg**).

- **-optimistic** : as of v.2023-03, this is shorthand for specifying **-userconfigfile BabelfishCompassUser.Optimistic.cfg** (see here) plus **-rewrite**.

- **-sqlendpoint** , **-sqllogin**, **-sqlpasswd**, **-sqldblist** : see Automatic DDL generation

- **-nooverride**: do not use classification/report group overrides from the user-defined **.cfg** file.

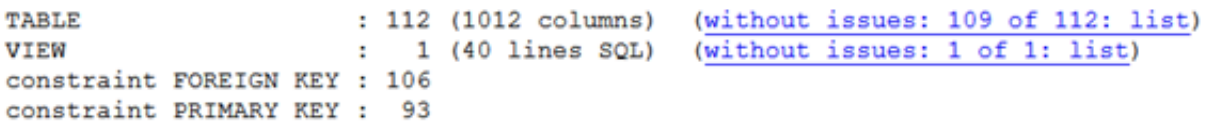
- **-noreportcomplexity**: as of v.2022-12, estimated complexity for not-supported items will not be included in the Compass report.

- **-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**.
    In addition (v.2022-07 or later) object names are listed for which issues (or no issues) were identified by Compass. These lists can be reached by clicking on the "without issues" link in the Object Count section (without **xref**, this link is not present):

  - **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.
- **hints**: (Compass v.2022-07 and later) lists all popup hints in the SQL Summary section at the end of the report (so that they can be read without requiring mouse action)

- **-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 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). By default, this table is named **public.BBFCompass**, but a different name can be specified with **-pgimporttable**.
  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.
- **-pgimporttable**: with **-pgimport**, specifies the name of the table to import the data into.
- **-rewrite**: for certain T-SQL constructs that are currently unsupported by Babelfish, performs automatic rewriting of the applicable syntax with T-SQL features supported by Babelfish (see Automatic rewriting of unsupported features).
- **-exclude** <*list*> : specifies a comma-separated list of file type suffixes to be excluded. By default, a series of file types are excluded (unless overridden with **-include**); to display these, use **-help exclude**.
- **-include** <*list*> : specifies a comma-separated list of file type suffixes to be included; only the filetypes specified with **-include** will be processed.
- **-recursive** : any subsequent directory names are processed recursively, using all files in the directory tree as input. Both **-include** and **-exclude** (if specified) are applied to any files found. With **-recursive**, it is recommended to specify **-appname** as well, otherwise each input file will be assumed to represent a different application.
- **-noupdatechk** : do not perform a check for a newer version of Babelfish Compass
- **-importfmt** <*format*> : process an XML file from SQL Server Profiler with captured SQL queries (see Processing captured SQL queries)
- **-nodedup** : with **-importfmt**, do not perform de-duplication of captured SQL
- **-csvformat** { **default** | **flat** } : as of v.2023-08, with flat, generates the **.csv** file in a flat format rather than the default 'structured' format (see User-defined estimates in .csv file)
- **-anon**: as of v.2024-07, generates a Compass reports with anonymized customer details; see The -anon option

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 1.5.0 (this example assumes the latest version of Babelfish is later than 1.5.0):

BabelfishCompass MyReport3 -analyze -babelfish-version 1.5.0

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

Generate a combined report for applications **Sales** and two applications **Finance** and **Inventory**, each of which consists of a directory tree containing **.sql** files on multiple levels, and perform automatic rewriting where possible:

BabelfishCompass MyReport5 -importonly C:\temp\Sales.sql

BabelfishCompass MyReport5 -add -importonly -appname Finance -recursive C:\Finance\install

BabelfishCompass MyReport5 -add -importonly -appname Inventory -recursive C:\Inventory\install

BabelfishCompass MyReport5 -analyze -rewrite -reportoption apps

Automatic rewriting of unsupported features

As of version 1.2 of Babelfish Compass, you can use the **-rewrite** option to address certain SQL features which are not currently supported by Babelfish, by rewriting the SQL feature in question in such a way that Babelfish is able to process it. 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 contains 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.

Notes:

- Using **-rewrite** may cause Babelfish Compass to run slower than without **-rewrite**, especially for large files in which many features are rewritten. For very large input files, it may therefore be practical to first run an analysis without **-rewrite**; when the Compass report indicates that rewrite opportunities were identified, then re-run Compass with the **-analyze -rewrite** flags.
- For dynamic SQL queries which contain unsupported-but-rewritable SQL features, no rewrite is performed. The SQL feature will be reported as 'Not Supported'.

Files & Directories for a Report

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.
  These files have cryptic-looking names like **SalesDDL.sql.bbf~imported.SalesApp**: here, **SalesDDL.sql** is the name of the original input file and **SalesApp** is the application name for the application**. bbf~imported** is added by Compass. Do not rename these files!
  For each imported file, an HTML version is also located in the **imported\html** 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 Compass-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. When using the **-pgimport** option, the files in this directory are imported into a PG table.
- **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.
- **rewritten**: contains rewritten input files as a result of using the **-rewrite** option. Only input files where actual rewriting was performed, will be present here.

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 as part of Babelfish Compass. When Babelfish Compass is already installed, the existing version of **BabelfishFeatures.cfg** should be replaced (overwritten) by the newer version of this file.

BabelfishFeatures.cfg should be treated as a read-only file: do not edit, modify, or rename the **BabelfishFeatures.cfg**; 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's 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. 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 root directory location for more information. The default location of this file is **C:\Users***username***\Documents\BabelfishCompass\BabelfishCompassUser.cfg** (on Windows). 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**.**

In v.2022-12, the user-defined **.cfg** file to be used can be specified with **-userconfigfile**; the default remains **BabelfishCompassUser.cfg**.

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.

The user-defined **.cfg** file allows you to:

- Override the classification of a not-supported SQL feature (e.g. 'Ignored' instead of 'Not Supported')
- Override the default complexity estimate (see next section)
- Define a user-defined effort estimate (see next section)

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's header.

Predefined 'Optimistic' .cfg file

As of v.2023-03, a predefined file named **BabelfishCompassUser.Optimistic.cfg** is included with Babelfish Compass. After installing the new Babelfish Compass version, this file is copied from the installation directory to **C:\Users***username***\Documents\BabelfishCompass** (on Windows) the next time Compass runs.
When specifying **-userconfigfile BabelfishCompassUser.Optimistic.cfg** , various T-SQL features which are unsupported by Babelfish will be reclassified as 'Ignored', thus removing them from the list of 'Not

Supported' features. This concerns T-SQL features that are unlikely to affect the actual application functionality and have a make-or-break effect on the success of migration the application, like ALTER DATABASE, or functions like FILEGROUP_NAME().
The reason for providing this **.cfg** file is that users are sometimes scared away from Babelfish by the initial Compass report which contains some of these 'false positive' T-SQL features. By removing these from the 'Not Supported' list, the Compass reports will present a more realistic -or, if you will, a more optimistic- view of how well the application qualifies for migration to Babelfish.
This predefined 'optimistic' **.cfg** file is intended primarily for such first impressions when no deep dive on the actual T-SQL feature in the application is performed. Once the actual migration is undertaken, there is no reason to use this **.cfg** file since all actual SQL aspects in the application will need to be handled regardless.

Compass users can modify **BabelfishCompassUser.Optimistic.cfg** as any other user-defined **.cfg** file (see above) but should be aware that each new version of Compass may include a new copy of this file, which will overwrite the existing copy. Therefore, if users want to customize **BabelfishCompassUser.Optimistic.cfg**, they should rename this file and apply the customizations to the renamed copy – and use that copy with the **-userconfigfile** flag.

When using the **-optimistic** flag, this is equivalent to specifying **-userconfigfile BabelfishCompassUser.Optimistic.cfg** plus specifiying **-rewrite**.

User-defined estimates & .csv file

The features in this section are aimed at advanced and experienced Compass users.

As of Compass version 2022-12, Compass supports a complexity estimate for not-supported items, as well as a user-defined effort estimate. A complexity estimate can be LOW, MEDIUM or HIGH reflecting a rough complexity estimate for resolving the item in question. The Compass report includes the complexity score for each non-supported item, displayed between square brackets:

```
BINARY_CHECKSUM() [medium] : 264
```

Compass defines a default complexity score, which can be overridden by the user (described below). In addition, Compass users can define an effort estimate in terms of minutes or hours, reflecting the amount of time it may take to address a particular non-supported item.

Compass generates a **.csv** file with the same filename as the report file, to assist advanced Compass users in qualifying/quantifying the migration work required to address not-supported items (as reported by Compass).
The **.csv** file contains the complexity score (either Compass default or the user-defined override) as well as any user-specified effort estimates.
The **.csv** file is intended to be imported into a spreadsheet, and Compass user should add their own formulas to the spreadsheet for performing calculations.
By specifying the **-noreportcomplexity** flag, the complexity scores will not be included in the Compass report; the **.csv** file containing the complexity scores will however always be generated.

'Flat' format for .csv file

As of version 2023-08, the **.csv** file can be generated in a 'flat' format (by using flag **-csvformat flat**) which may be more suitable for automated processing that the default 'structured' format.

As of version 2023-11, the **.csv** file contains an additional **ItemID** column allowing each item to be uniquely numbered. These numbers must be defined by the user: if a file named **BabelfishCompassItemID.csv** exists (in the Compass report root directory) , then the ItemID definition in that file will be picked up. When the file does not exist or when no itemID is specified for an item, a value of **-1** is assigned. A different file name can be specified with flag **-csvitemidfile** .
The file **BabelfishCompassItemID.csv** must be defined by the user and must be semicolon-delimited. A line must follow this layout (also see Example of BabelfishCompassItemID.csv)

itemID; itemDescription ; hint (optional)

Here, **itemID** is a number that will be assigned to the item with the text in the second field, using the steps described below. **itemDescription** is the actual text of an item as it is shown in the Compass

report. The **hint** field is optional but allows specifying free text with solution hints that will copied into the final Compass-generated **.csv** file, overriding any default Compass hint, if present.

To match an **itemID** to an **itemDescription** in the report, the following steps are applied. When a match is found, processing stops for the item in question:

1. When an **itemDescription** contains one of the 3-char pattern strings **\d+** or **\w+**, the **itemID** is applied if the string and pattern match the start of the item being reported. These patterns are interpreted as standard POSIX regular expression patterns; other patterns are not supported and will be treated as literal text.
2. When an entry is identical to the item being reported, it is applied.
3. When an entry matches the start of the item being reported, it is applied.
4. If no matching value is found, **-1** is used.

Given this order of processing, the most specific **itemDescription** in the file should come before the more generic ones (like those with a pattern or only matching the start).
When matching, commas and semicolons are ignored (note that semicolons cannot be used anyway as the file must be semicolon-separated) and whitespace is collapsed.
For a template for **BabelfishCompassItemID.csv** , see Example of BabelfishCompassItemID.csv .

Default complexity score values

The default, Compass-provided classifications LOW, MEDIUM or HIGH are intended as a rough, non-quantative indication of the level of effort required to resolve the non-supported items:

- LOW: fixing requires a very limited amount of effort, which is very simple and/or not dependent on how many times the item occurs. Remedies include global search-and-replace, use of an editor macro, or low-complexity manual editing etc.
- MEDIUM: fixing is possible but may require custom effort for every occurrence, and/or having to write some custom SQL code (like a SQL function or stored procedure to implement the desired functionality, and invoke it from the T-SQL code).
- HIGH: fixing may be complex, requiring potentially significant effort which may or may not be technically or economically feasible. For example, it may require rewriting/refactoring parts of the application and/or implementing parts of the functionality natively in PostgreSQL, possibly involve significant custom SQL coding.

Complexity score estimates

The Compass user can override the default complexity scores by adding their own complexity scores in the **BabelfishCompassUser.cfg** file using the **complexity_score** keys (see examples below).

It should be noted that the default Compass complexity scores are generalized and somewhat arbitrary since the experience and expertise of the team performing the migration is a much more deciding factor for the overall effort required. These Compass-provided complexity estimates should therefore be taken as very rough high-level guidance only; Compass users are urged to evaluate and adjust all values in the context of the actual customer application being analyzed.

The user-specific override values in **BabelfishCompassUser.cfg** can be LOW, MEDIUM or HIGH, but also an arbitrary number between 0 and 100 (so as to allow Compass users to use more detail).

When uploading analysis details into PG using the **-pgimport** option, the complexity score is available in column **misc**; user-defined effort estimates are uploaded as of version 2023-06.

Complexity score defaults

As of v.2023-08, a default complexity can optionally be defined in the **BabelfishCompassUser.cfg** file for items with a particular status (except for status = 'Supported'). This default is applied when no complexity has been specified for an item in neither the **BabelfishFeatures.cfg** file nor the **BabelfishCompassUser.cfg** file. If no default is defined for a particular status value, MEDIUM will be assumed.
The complexity defaults must be specified in a section **[Complexity Score Defaults]** which must occur towards the top of the in the **BabelfishCompassUser.cfg** file before any of the regular sections.
Example:

```
[Complexity Score Defaults]   # these complexity values are arbitrary examples; do not copy!
complexity_score_default_NotSupported=high
complexity_score_default_ReviewPerformance=high
complexity_score_default_ReviewSemantics=low
complexity_score_default_ReviewManually=low
complexity_score_default_Ignored=low
```

Effort estimates

It is also possible to define your own effort estimates, expressed in minutes, hours or days, in the **BabelfishCompassUser.cfg** file using the **effort_estimate** keys (see examples below). These values are not shown in the generated report, but only in the generated **.csv** file.
As of release 2023-08, two values can be specified for the effort estimate, one for 'scaling' and one for 'one-time learning curve'. Here, 'scaling' is the estimated effort per occurrence of a particular item, and 'one-time learning curve' is an effort estimate that reflects the one-time effort that needs to be spent for devise a solution for a particular item. As an example, there could be a on-time learning curve effort estimate of 1 hour, and an additional scaling effort estimate of 5 minutes per occurrence.
The effort estimates are specified as *<number><unit>*:*<number><unit>*, where the value before the '**:**'

is the 'scaling' effort and behind the '**:**' is the 'one-time learning curve' effort. Either value is optional. When the colon id omitted, the value represents the 'scaling' effort.
Examples (these all represent the same values: 10 minutes for each occurrence of an item, and 2 hours for the one-time learning curve for the type of item):

- **10m:2h**
- **10 mins : 2 hrs**
- **10 minute : 2 hours**

When user-defined effort estimates are used, these values will show up in additional 'Effort' columns in the **.csv** file: the first column is a textual representation of the user-configured value (e.g. '**5 minutes**' or '**1 hour**' or '**1 day**'), and the second is the corresponding number of minutes, for calculation purposes (e.g. **5** , **60**, **480** minutes, respectively).

Prior to 2023-08, only one effort estimate value could be specified, which represented 'scaling' effort.

When uploading analysis details into PG using the **-pgimport** option, as of v.2023-06, user-defined effort estimates for 'scaling' are available in column **misc2**; as of v.2023-08, user-defined effort estimates for 'one-time learning curve' are available in column **misc3**.

Real-life effort estimate in executive summary

As of release 2023-11, if user-defined effort estimates have been specified, the executive summary at the top of the Compass report will contain a summary of the total estimated effort for functional SQL migration, like this:

User-defined SQL migration effort estimate (excl. data migration, tuning, testing, etc.): 5 weeks (5 days/week, 8 hours/day)
(based on user-defined estimates in MyCompassEffortEstimates.cfg)

The calculation of weeks/days is intended to provide a rough real-life, order-of-magnitude indication of the level of human effort required. Please note that this number is derived from the user's own effort estimate definitions. Compass is not providing such estimates itself. Interpreting and validating these effort numbers are the exclusive responsibility of the Compass user.
The final number of weeks/days shown here depends on how many hours are counted per day (default=8), and days per week (default=5). Different values may however be desired to reflect some real-life considerations about time effectiveness. Such values can be defined by adding these lines to the user-defined **.cfg** file:

effort_estimate_default_Hours_Per_Day=6
effort_estimate_default_Days_Per_Week=4

Note that this definition of hours/day is used only for calculating the final total effort in the executive summary section. When defining an effort estimate for a particular item as '1 day' (as described in the rest of this section), that means 8 hours or 240 minutes.
When the number of days/weeks calculated as above is less than 2, it is rounded upwards to avoid showing an unrealistic small number.

Examples

The precise way of specifying these values is explained a bit more in the header of the **BabelfishCompassUser.cfg** file. The sections in this file correspond to the same section in **BabelfishFeatures.cfg** (which you cannot edit yourself), and in most cases correspond quite obviously to the grouping of reported items in the Compass report.
(NB. the actual values shown below are chosen arbitrarily and should not be used as guidance)

```
[Built-in functions]
complexity_score-HIGH=SOUNDEX    # complexity = HIGH for this function, if unsupported
complexity_score=LOW             # complexity = LOW for any other unsupported function
complexity_score-60=COL_LENGTH   # complexity = 60 for this function, if unsupported


[Cursor options]
effort_estimate-4hours=SCROLL,FOR_UPDATE     # 4 hours for these options, if unsupported
effort_estimate=1hour:1day                   # 1 hour/1 day for other unsupp'd options
```

Effort estimate defaults

As of release 2023-08, default effort estimates can optionally be defined in the **BabelfishCompassUser.cfg** file for items with a particular status (except for status = 'Supported'). These defaults are defined by the lines **effort_estimate_default_**{**NotSupported**|**Review**…**Ignored**} as shown below, and are applied when no effort estimate has been specified for an item in neither the **BabelfishFeatures.cfg** file nor the **BabelfishCompassUser.cfg** file. In case no effort estimate default is defined, then an effort estimate default for the complexity of the item is used, as defined below by lines **effort_estimate_default_**{**low**|**high**} . If the latter is not defined either, zero effort is assumed. The effort estimate defaults must be specified in a section **[Effort Estimate Defaults]** which must occur towards the top of the in the **BabelfishCompassUser.cfg** file before any of the regular sections. Example:

```
[Effort Estimate Defaults]   # these effort estimate values are arbitrary examples; do not copy!
effort_estimate_default_high=1h:4h
effort_estimate_default_medium=10m:30m
```

effort_estimate_default_low=1m
effort_estimate_default_NotSupported=30m:1h
effort_estimate_default_ReviewPerformance=1h:2h
effort_estimate_default_ReviewSemantics=15m
effort_estimate_default_ReviewManually=5m
effort_estimate_default_Ignored=1m

Uploading details into PostgreSQL with -pgimport

The **-pgimport** flag lets you load all captured items into a PostgreSQL table. From there, 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 your session's PATH.
By default, data is imported into a table named **public.BBFCompass**, but a different name can be specified with the **-pgimporttable** option.

Note: when using the **-pgimport** flag, any user-defined complexity estimates or effort estimates in the user-defined **.cfg** file will only be included in the uploaded data when the user-defined **.cfg** file is specified together with **-pgimport** (for the "optimistic" **.cfg** file, you can specify **-optimistic** instead).

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(300) 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,
    misc2 BIGINT NOT NULL,  -- as of v.2203-06
    misc3 BIGINT NOT NULL  -- as of v.2203-08
    );
```

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 : as of 2022-12: complexity score; in previous versions: not used.
- misc2 : added in 2023-06: 'scaling' effort estimate in minutes, if defined.
- misc3 : added in 2023-08: 'one-time learning curve' effort estimate in minutes, if defined.

Example queries

You can run SQL queries against the imported items to derive information on a more detailed level than can be presented in the Compass report Some examples are shown below.
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.

Example 1: complex filtering

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%')
```

Example 2: object dependencies

As of release 2023-10, it is possible to extract object dependencies from the uploaded data (for reports generated by previous Compass releases, data for object dependencies is incomplete). Use this query:

```
select * from (
      select 'SELECT' as DMLStatement, itemDetail as objectReferenced,
      'SELECT' as accesstype, context from public.BBFCompass where item like
      'SELECT FROM @tableVariable%'
   union all
      select reportgroup as DMLStatement, itemDetail as objectReferenced,
      item as accesstype, context from public.BBFCompass where
      status='OBJECTREFERENCE' and reportgroup <> 'DDL'
   union all
      select substring(item from E'^(.+?)\\W') as DMLStatement, itemDetail
      as objectReferenced, 'SELECT' as accesstype, context from
      public.BBFCompass where item similar to '(INSERT|UPDATE|DELETE|MERGE)
      @tableVariable%' and item not like 'Cross-database%'
   union all
      select substring(item from E'^(.+?)(\\W|$)') as DMLStatement,
      itemDetail as objectReferenced, substring(item from E'^(.+?)(\\W|$)')
      as accesstype, context from public.BBFCompass where item similar to
      '(INSERT|UPDATE|DELETE|MERGE)%' and item not like '%(target)%' and
      item not like '%@tableVariable%'
   union all
      select item as DMLStatement, itemdetail, item as accesstype, context
      from public.BBFCompass where item like '%SELECT..INTO%'
   union all
      select 'INSERT..EXECUTE' as DMLStatement, itemDetail as
      objectReferenced, 'INSERT' as accesstype, context from
      public.BBFCompass where item like 'INSERT..EXECUTE%'
   union all
      select 'EXECUTE' as DMLStatement, itemDetail as objectReferenced,
      'EXECUTE' as accesstype, context from public.BBFCompass where item
      like 'EXECUTE %'
   union all
      select item as DMLStatement, itemDetail as objectReferenced, item as
      accesstype, context from public.BBFCompass where reportgroup = 'DDL'
      and item not like 'Option %' and item not like 'Constraint %' and item
```

```
      not like 'Index%' and item not like 'CREATE INDEX%' and context not
      like 'TABLE %' and subcontext not like 'TABLE %'
   union all
      select item as DMLStatement, itemDetail as objectReferenced, item as
      accesstype, context from public.BBFCompass where item in ('TRUNCATE
      TABLE') ) t
where context not like 'TABLE %'
order by context;
```

Processing captured SQL queries

Apart from server-side DDL, also client-side SQL queries should be considered during a database migration. When capturing client-side SQL queries as described below, Babelfish Compass can extract the SQL queries from the capture files and perform a Compass assessment on them.
In order to process capture files, specify the command-line option **-importfmt** *format*, as shown below.
Since captured SQL often contains many near-duplicate statements that only differ in the value of a lookup key or a constant, by default Compass de-duplicates the captured SQL prior to analysis. De-duplication is performed by masking the values of all string/numeric/hex constants.
To suppress de-duplication, specify the command-line option **-nodedup**.

The extracted and de-duplicated SQL queries/batches are saved into a file in directory **extractedSQL**: a file named **MyCapture.xml** will be saved into **extractedSQL/MyCapture.xml.extracted.sql**. This file is then used as input for the Compass analysis.

SQL Server Profiler

To capture SQL statements with SQL Server Profiler, take these steps:

1. In SQL Server Profiler, under "Trace Properties", use the **TSQL_Replay** template
2. Initiate the tracing in SQL Server Profiler
3. Run the client application against the SQL Server database
4. When done capturing the client application's SQL, save the captured results (in SQL Server Profiler) with **Save As➔Trace XML File for Replay** . This creates an XML file containing the captured SQL batches.
5. Run Babelfish Compass with the just-created XML file as input, and specify the command-line option **-importfmt MSSQLProfilerXML**
    - The extracted SQL batches are saved into a file in directory **extractedSQL**: a file named **MyCapture.xml** will be saved into **extractedSQL/MyCapture.xml.extracted.sql**

SQL Server Extended Events

To capture SQL statements with SQL Server Extended Events, take these steps:

1. Run the client application against the SQL Server database
2. Use SQL Server Extended Events to capture SQL queries
3. Extract the captured events from the **.xel** file as **.xml** files containing `<event…> … </event>` XML documents. Note that the **.xel** files cannot be processed by Compass.
4. Run Babelfish Compass with the XML file as input, and specify the command-line option **-importfmt extendedEventsXML**

Examples

BabelfishCompass MyReport C:\temp\MyProfilerCapture.xml -importfmt MSSQLProfilerXML

BabelfishCompass MyReport C:\temp\MyXECapture.xml -importfmt extendedEventsXML

Automatic DDL generation

As of Babelfish Compass release 2023-06, Compass may optionally perform the DDL generation directly for the SQL Server database(s) for which analysis needs to be performed. This means that the Compass user does not have to manually generate the DDL script using SQL Server Management Studio. This may not always be the most optimal solution (see below), but it can simplify the overall process.

Command-line options

To let Babelfish Compass perform the DDL generation, the following command-line options must be specified. Compass will then connect to the SQL Server, generate the DDL file(s), and run a Compass analysis on the generated files:

- **-sqlendpoint** : the hostname or IP address of the SQL Server; optionally, the port number can also be specified (e.g. **10.123.45.67** or **mybigbox,1433**).
- **-sqllogin** : the login name to connect to the SQL Server; this login should typically be a member of the **sysadmin** role so as to have permission to generate the DDL.
- **-sqlpasswd** : the corresponding password.
- **-sqldblist** : optional; when omitted or when 'all' is specified, DDL is generated for all user databases in the server; alternatively, a comma-separated list of database names can be specified.

Notes:

- While generating DDL, Compass creates a network connection to the SQL Server specified, through a Powershell script (see below).
- When one of options **-sqlendpoint** , **-sqllogin** or **-sqlpasswd** is specified, the others must be specified as well.
- These options cannot be combined with specifying input files; also, this cannot be used to add DDL files to an existing report with **-add** or **-replace**, so you must either create a new report or use the **-delete** flag. Other Compass options such as **-rewrite** , **-optimistic** and **-reportoption** can be specified as usual.
- One DDL file is generated per T-SQL user database. When more than one database is processed, the flag **-reportoption apps** is automatically applied so that the Compass report shows how the T-SQL features are distributed over the different databases.
- The DDL files are generated into a directory named similar to **CompassAutoDDL-2023-Jun-07-13.22.51** under your session's **%TEMP%** location (on Windows) or **/tmp** (Mac/Linux). The directory is reported by Compass on the console output. The directory is not deleted afterwards so the original DDL files are retained.

- The speed of generating the DDL is highly dependent on the network proximity to the SQL Server since this involves many client-server roundtrips. When a few thousand objects exist in the database, it may take minutes to hours to complete, depending on the network proximity. You can tell that DDL generation has completed when the message '**DDL generated in** *<directory>*' occurs, and Compass starts its analysis of the generated DDL.
  If it takes an unreasonably long time to generate the DDL, then you might want to consider to use SQL Server Management Studio instead.
- Even with optimal network connectivity, generating the DDL will likely take more time than analyzing it by Compass.
- On Windows, if the password contains a **^** character (a.k.a. 'caret', 'circumflex', or SHIFT-6), enclose the password in double quotes since this is an escape character in Windows .bat and Powershell.

How it works

Babelfish Compass uses .Net SQL Management Objects (SMO) to perform the DDL generation. Compass invokes a Powershell script named **SMO_DDL.ps1**, which is located in the Babelfish installation directory. This script uses SMO and connects to the SQL Server specified by the user, requiring a network connection.

On Windows, Powershell is available by default. On Linux and Mac, Powershell first needs to be installed by the user before running Compass (see https://learn.microsoft.com/en-us/powershell/scripting/install/installing-powershell-on-linux) .
When Powershell script **SMO_DDL.ps1** is invoked, it will install SMO if needed (typically this only happens when it runs for the first time); this installation may take some time, and requires a network connection.
On Windows, in order to run the Powershell script, your Windows environment must have the 'Powershell Execution Policy' set to **Unrestricted**; in release 2023-08, Compass checks this and tells you whether the setting is as required.

This Powershell script, and the installation of SMO, have been tested successfully on Windows, Linux and Mac. However, in rare cases installing SMO might fail. The remedy would be to install SMO manually but that may not be trivial given that the installation steps in the script have already failed (this user guide does not provide guidance how to install SMO manually). Alternatively, use the manual approach to generate DDL via SQL Server Management Studio.

Generating only DDL

The script **SMO_DDL.ps1** can also be used stand-alone to generate DDL. This can be useful when there are many SQL Server instances and/or many databases per instance, and perhaps you want to inspect or consolidate the DDL scripts before running a Compass analysis on them. For instructions how to run the script stand-alone, see the header of the script.
Notes:

- Do not make any changes to the **SMO_DDL.ps1** script, since Compass expects this script to be available without any changes. If you want to make changes, take a copy and modify the copy.
- As above, Powershell needs to be installed and available.

Example

```
C:\BabelfishCompass> BabelfishCompass.bat SMOTest -sqlendpoint mybigbox -sqllogin
sa -sqlpasswd MyS3cret -sqldblist Sales,Ledger
Babelfish Compass v.2023-06, June 2023
Compatibility assessment tool for Babelfish for PostgreSQL
Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.

Reading BabelfishFeatures.cfg
Latest Babelfish version supported: 3.2.0: BabelfishFeatures.cfg
Reading
C:\Users\johnsmith\Documents\BabelfishCompass\BabelfishCompassUser.Optimistic.cfg

Running Powershell/SMO script to generate DDL for server 'mybigbox' into directory
C:\Users\johnsmith\AppData\Local\Temp\CompassAutoDDL-2023-Jun-07-13.22.51...
Note: run times strongly depend on network proximity to the SQL Server.
Alternatively, generate DDL manually through SQL Server Management Studio on the
SQL Server host.
You can abort by hitting CTRL-C

DDL generated in C:\Users\johnsmith\AppData\Local\Temp\CompassAutoDDL-2023-Jun-07-
13.22.51 :
 Volume in drive C is OSDisk
 Volume Serial Number is FC25-F714

 Directory of C:\Users\johnsmith\AppData\Local\Temp\CompassAutoDDL-2023-Jun-07-
13.22.51

07-Jun-2023  13:24    <DIR>          .
07-Jun-2023  13:24    <DIR>          ..
07-Jun-2023  13:24         1,959,008 Ledger_SMO_DDL_2023-Jun-07.sql
07-Jun-2023  13:23           925,052 Sales_SMO_DDL_2023-Jun-07.sql
               2 File(s)      2,884,060 bytes
               2 Dir(s)  204,604,514,304 bytes free

Creating C:\Users\johnsmith\Documents\BabelfishCompass\SMOTest

Run starting               : 07-Jun-2023 13:22:51 (Windows)
BabelfishFeatures.cfg file : v.3.2.0, Jun-2023
Target Babelfish version   : v.3.2.0 (PG 15.3)
Command line arguments     : SMOTest -sqlendpoint mybigbox -sqllogin sa -sqlpasswd
******** -sqldblist Sales,Ledger
[…rest of Compass session…]
```

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 these exceptions:

- The **-pgimport** option, which connects to a PG instance.
- The automatic DDL generation with **-sqlendpoint**, **-sqllogin**, **-sqlpasswd**, which connects to a SQL Server instance.
- The automatic check-for-updates, which connects to GitHub (see below).

Other than in these cases, the Babelfish Compass tool makes no network connections that are invisible to the user. Also, Babelfish Compass does not "phone home".

The -pgimport option

The **-pgimport** option connects to a PostgreSQL instance and loads 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.

The -sqlpasswd option

When using the **-sqlpasswd** option (see Automatic DDL generation), the specified password is passed on to the Powershell script. This script connects to the SQL Server specified with **-sqlendpoint**. The password itself is not stored or logged anywhere by Babelfish Compass. The **-sqlendpoint**

and **-sqllogin** options are included in the Compass report header under "Command line arguments". Note that the password and other connection attributes may be accessible through the command-line history in the command-line session that runs Babelfish Compass.

The -anon option

In v.2024-07, the **-anon** option anonymizes the data in the report by removing or renaming all customer-specific identifiers and names. Only the report name is kept unchanged.
Customer-specified identifiers are either replaced by the word **anonymized** or by **procedureN**, **functionN**, **dbN**, **appN** (etc), where **N**=1,2,3,…
This option can be used to address potential customer concerns about customer-specific metadata being shared outside the customer organization. By specifying **-anon**, Compass reports can be shared without disclosing any customer-specific details.

When specifying **-anon**, the option **-reportoption xref** cannot be used.

Automatic update check

Starting with Babelfish Compass version 2022-04, Compass checks whether a more recent version of itself is available at GitHub by making a REST call to GitHub repository. If so, it will print a message to inform the user, but not take any further action: the user must still download and install the update manually.
No information about these REST calls is collected or stored, other than by GitHub's default behavior.

To suppress the update check, specify the **-noupdatechk** option.

Scaling Compass: Analyzing Multiple Applications

In most cases where Babelfish is considered, the focus is on a single specific application only.
In case there are have multiple applications, the default approach would be to run Compass for each application and produce one Compass report for each application.
In some cases it may be more convenient to generate a single combined Compass report for multiple applications. The key to generating such a report is to use the flag **-reportoption apps** (see Command-line options).
For each item in the Compass report, this option will report the various applications in which the item was found. The following shows how 45 calls of the function SOUNDEX() occur in three different applications:

SOUNDEX() : 45 #apps=3: Product(16), Sales(20), HR(9)

Assuming there is one DDL script for each application, there are multiple ways to generate such a combined Compass report:

- Enumerate all DDL scripts to be processed:
  BabelfishCompass MyReport C:\tmp\Product.sql C:\tmp\Sales.sql C:\tmp\HR.sql -reportoption apps
- If the input scripts are in the sae location, use a wildcard:
  BabelfishCompass MyReport C:\tmp*.sql -reportoption apps

When running the above without **-reportoption apps**, the generated Compass report will be identical, but it will not show the breakdown of each item across the various application. Note that **-reportoption apps** is silently ignored when there is only one application in the report.

Scaling Compass: Analyzing Many Applications

Imagine a large SQL Server customer which is running thousands of SQL Server databases/applications, and a top-10 list of candidate applications must be identified for a Babelfish PoC. Let's say a candidate app should have a smaller number of lines of SQL code in procedures/functions/triggers/views, the fewest number of unsupported features, and no high-complexity unsupported features at all.
How would you approach this?

In its simplest, and also its most laborious form, you could take these steps:

1. For each of those thousands SQL Server databases, run SSMS to generate the DDL
2. For each DDL script, run Babelfish Compass to produce a Compass report
3. Examine each Compass report to identify the best candidate. The Executive Summary at the top should contain the high-level metrics allowing you to make a first selection, e.g.:

```
-------------------------------------------------------------------------------
--- Executive Summary for Babelfish v.3.3.0 -----------------------------------
-------------------------------------------------------------------------------
Total #lines of SQL/DDL: 117821      Total SQL features: 70894
#Procedures/functions/triggers/views: 1160    #Tables: 1003

SQL features not supported by Babelfish (total/unique): 631/32
Estimated complexity of not-supported features (total/unique): low:29/2 medium:469/10 high:133/20
```

The problem is that each of these steps must be done as many times as there are SQL Server databases, which makes this a non-feasible approach when the numbers are large.
Fortunately, Compass offers various features to make this task more manageable.

Improvement 1: Generate DDL in batch

Using the **-sqlendpoint** option of Compass (and **-sqllogin**/**-sqlpasswd**), the DDL for a database can be generated by Compass directly, without having to go through SSMS. Assuming you have a list of all SQL Server databases and the required login/password, you can now write a script that generates the DDL for each database, and generates a Compass report for it. This script can run unattended as a batch job, without requiring further manual action.
Note that using **-sqlendpoint** works best when Compass is located close to the SQL Server from a network proximity perspective, since the process requires many client/server round trips.

Using unattended DDL generation is clearly a step forward, but we still need to dig through thousands of Compass reports to identify candidate applications.
To optimize, you could consider using tools like **grep**, **awk** or **perl** to extract the numbers from the Executive Summary in the Compass reports. But there is a better way!

NB. When using **-sqlendpoint**, the generated DDL script is placed in the directory indicated by **%TEMP%** (Windows) or in **/tmp** (Linux/Mac), as shown in the Compass report header. You can copy the DDL script from this location if desired. A copy of the processed input files is always located in the **imported** directory for the report.

Improvement 2: Capture 1-line metrics per report

As of Compass release 2023-11, the metrics in the Executive Summary are printed to **stdout** on a single line like this (shown wrapped around here because the line exceeds the page width but it is all printed as a single line):

CompassExecutiveSummary:3.3.0 report:MyBigApp linesSQL:117821 featuresSQL:70894/920 sqlObjects:1160 tables:1003 featuresNotSupported:631/32 complexityLow:29/2 complexityMedium:469/10 complexityHigh:133/20

When scripting the Compass analysis of all those databases as a batch job as described above, you should also the output of the batch job in a file. You can then search for **CompassExecutiveSummary** to get 1 line for each Compass report. When importing these lines into a spreadsheet, you can easily sort the reports on a metric of choice and generate a top-X of Babelfish PoC candidates.
(if you did not capture the batch job output, you can still find these lines by searching through the session log files in the **log** subdirectory for each Compass report).

Advanced Compass Usage

This section describes some ways to use Compass that most users will never need, but may still be useful at some point.

Manually copying imported files

Each Compass report consists of a set of files and directories under the Compass reports root directory. These files and directories are self-contained, platform-independent and transportable: you can copy the directory for a report to a different host (into the Compass reports root on that host) and run Compass against it. This also works between Windows / Linux /Mac.
The reason why you can move a report around is that a copy of all input files is saved in the **imported** directory for the report (see Files & Directories for a Report)
When re-running analysis with the **-analyze** flag, the files in **imported** are re-read as input, so the original DDL scripts are not accessed.
Technically, it is possible to manually combine multiple reports into one:

1. Copy one or more files named ***bbf~imported*** into the **imported** directory of another report. Note that the copied files should not be renamed.
2. Re-run Compass for the target report with the **-analyze** flag: this will take the files in the **imported** directory as input and reprocess them. The resulting report is the same as when the original input files would have been specified.

A single Compass report for very large amounts of SQL

Compass scales quite well until a few million lines of SQL code per report. When the amount of SQL code becomes much larger, processing time may get dramatically worse (depending on the resources in your system, this could happen quicker or later).
Let's say a large SQL Server customer is running thousands of SQL Server databases/applications, and you want a single Compass report covering all of these applications.
If you tried to process all DDL scripts in a single Compass operation, this would likely be too slow and might seem to never complete. The same would happen when copying all DDL scripts from each imported directory into a single Compass report and running **-analyze** (as described in Manually copying imported files).
In such a case, the processing can be split up in an analysis part for each application, and a final reporting step for all of the analyzed applications together, as follows:

- Run Compass for each application or database (use **-analyze** if the report already exists) and add the (undocumented) flag **-mergereport MyBigReport** . Do this for all applications.
- The **-mergereport** flag will copy all files for that report into a separate report directory called **MyBigReport**

- When all applications have been analyzed, generate the final collective report as follows: **BabelfishCompass MyBigReport -reportoption apps**

NB. Running **BabelfishCompass MyBigReport -analyze** will likely exhibit the slow-processing behaviour which we are trying to avoid by following the steps above.

Locating items in very large SQL scripts when cross-ref links are slow

Compass can generate a cross-reference to the original SQL code with the flag **-reportoption xref** (see Command-line options). This creates hyperlinks in the Compass report that lead to the location of the item in the original SQL script file.

If the original SQL script is very large (e.g. > 500,000 lines) your browser may take considerable time after clicking a cross-ref hyperlink (depending on your browser and resources available). If this takes too long, there is a shortcut to finding the location by directly looking into the file in the **captured** directory. Such files are named similar to **captured.SalesDDL.sql.bbf~captured.SalesApp** , where **SalesDDL.sql** is the original SQL script filename and **SalesApp** is the application name. These files contain the information that was extracted from the original SQL code by Compass, and this is what the final report is generated from.

Let's say a case of a dynamically created cursor is reported by Compass, and you want to fund its location. You search in the **captured.*** file for **'Dynamically created cursor'** and you find the following line, which is semicolon-separated:

Dynamically created cursor;PrdCurs;Cursors;NOTSUPPORTED;91;MyBigApp;c:\\temp\\MyBigApp.sql; 5564;565164;PROCEDURE dbo.p_R3285_2;;;~;

Without getting into too much detail, what we see from this line is:

- This feature occurs in a procedure named **dbo.p_R3285_2** (this could be enough of a pointer already) and the cursor is named **PrdCurs**
- The batch in which this feature occurs starts at line 565164 in the SQL source file (it's batch #5564, but that information is probably not very helpful)
- The feature occurs at (or around) line 91 in the batch; so that means it's at line 565164 + 91 – 1 = 565254 in the original file (c:\temp\MyBigApp.sql) .
- A copy of the original SQL file is in the **imported** directory. In this case it would be named **MyBigApp.sql.bbf~imported.MyBigAppApp .** Note that this file has one additional line inserted at the top of the file, so you should look for line 565255.

NB. When looking in the **captured.*** files, ignore any lines that have XREFONLY, OBJECTCOUNTONLY, or OBJECTREFERENCE in the 4th field (just don't ask – but the Compass code is all on Github if you really want).

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 in T-SQL, 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 in the input script and re-process 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.

- Memory: In case Compass runs out of memory, there will be a Java stack trace starting with a line like this:

  ```
  Exception in thread "main" java.lang.OutOfMemoryError : Java heap space
  ```

  By default, Compass is allowed a maximum of 12GB. This can be increased by editing **BabelfishCompass.bat** or **BabelfishCompass.sh** and modifying the following line (located towards the end of the file):

  ```
  java -server -Xmx12g -enableassertions -jar compass.jar %*
  ```

  Here, change **12g** to something bigger, for example **20g**. Obviously, the host needs to have this amount of memory available otherwise Compass will still run out of memory.

Example of BabelfishCompassItemID.csv

See 'Flat' format for .csv file for details.

```
# BabelfishCompassItemID.csv : item IDs for Compass .csv file
#
# This file must be located in the report root folder, i.e.
# %USERPROFILE%\BabelfishCompass on Windows.
# This file is applied only when flag '-csvformat flat' is specified.
# The filename can be overridden with flag -cvsitemidfile
# When the file is present if will be used to supply values for the .csv file;
# if not present the .csv is generated without item ID values.
# See the Compass User Guide for details about how to use this file.
#
# File layout: .csv with semicolon as field separator
#   - field 1 = item ID
#   - field 2 = item description
#   - field 3 = item hint (optional)
#

# NB: the lines below are examples! Do not copy these but define your own.
1233;ALTER ROLE db_datareader ADD MEMBER
1234;ALTER ROLE db_owner ADD MEMBER; This overrides any hint by Compass
1246;Option ALLOW_PAGE_LOCKS=ON constraint PRIMARY KEY in CREATE TABLE
1247;Option ALLOW_PAGE_LOCKS=ON Index in CREATE INDEX
9000;Number of procedure parameters (115) exceeds 100
9009;Number of procedure parameters (\d+) exceeds 100   # this matches any
number of parameters; if this comes before the previous line, the line with 115
would never be matched
9002;GRANT INSERT # This matches any item starting with GRANT INSERT
9003;GRANT;       # This matches any item starting with GRANT, except GRANT
INSERT which is matched by the previous line
#
# end of file
#
```