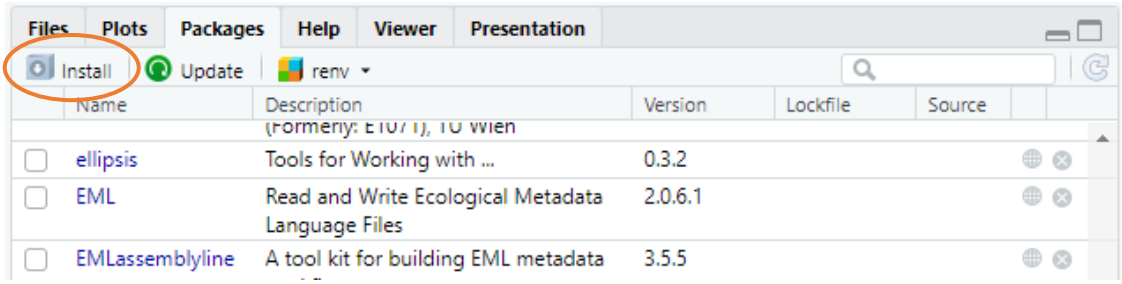
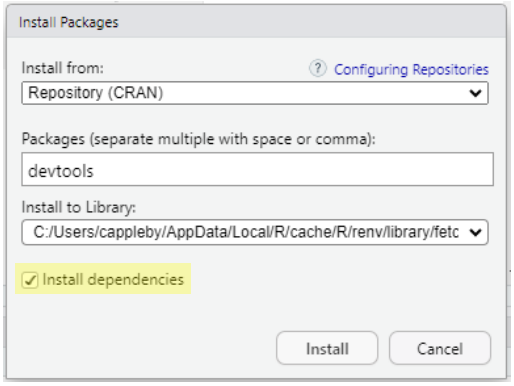
fetchsqlserver

fetchsqlserver is R package helps with data package publication to the DataStore. There are functions to fetch data tables from SQL Server and create EML metadata, which includes attribute tables, categorical variable tables, base EML information, and NPS-specific metadata. This document explains installing **fetchsqlserver**, setting up data tables queries, and creating the files needed by the package to create metadata. The package includes an R markdown template that walks through the creation of a data package.

Getting Started

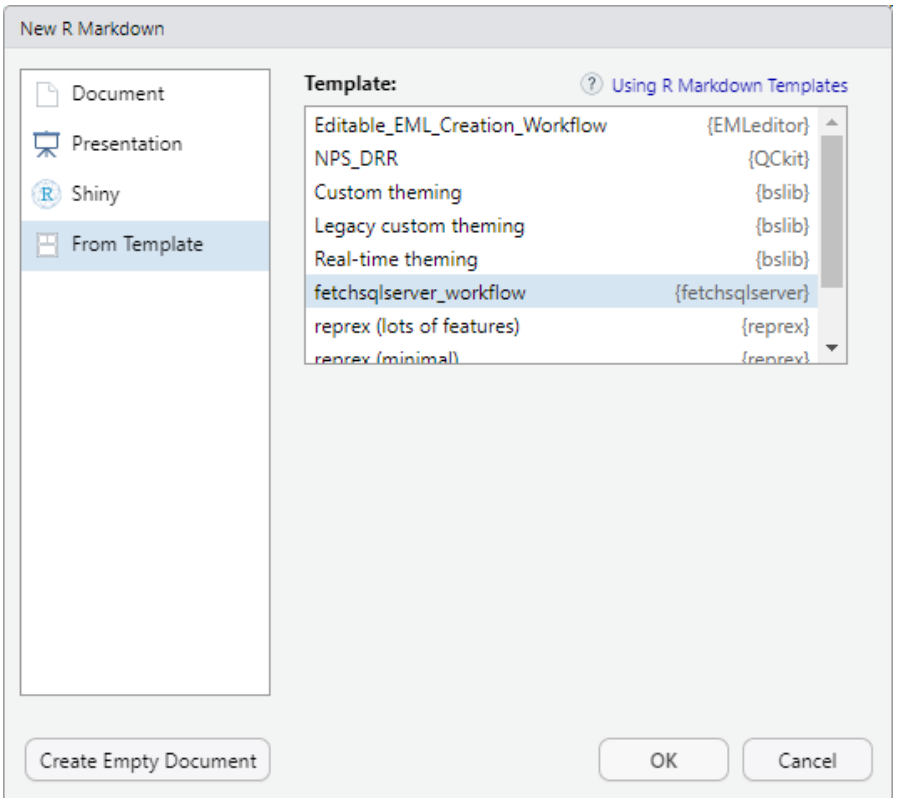
1. Open RStudio, go to **File > New Project...** Starting a project in a new directory is recommended as this allows you to select the project type. For **Project Type**, select **New Project** or **Quarto Project**, whichever you prefer. If you are uncertain, stick with **New Project**.
2. In the lower right-hand corner, you should see a window with multiple tabs. Click on Packages and ensure **devtools** package is installed. If **devtools** is not installed, click the install button in the top left corner of the window to install it. Ensure **Install dependencies** is checked. Alternatively, you can run **install.packages('devtools')** in the Console or the Markdown file.

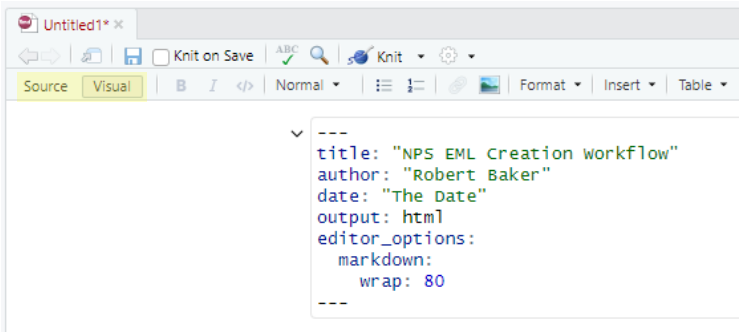




3. Install **fetchsqlserver** and **NPSdataverse** libraries by running the following code either in the Console or in the Markdown file:
   devtools::install_github("NPS-NCCN/fetchsqlserver_package")
   devtools::install_github("nationalparkservice/NPSdataverse")
4. Once **fetchsqlserver** is installed, you can open an R Markdown template that walks through the creation of the NCCN Water Quality data package using fetchsqlserver. Much of the information

found in the Data Package Creation section is also in the template. To open the template, select **File > New File > R Markdown...** In the window that opens, select **From Template** in the left panel, and select **fetchsqlserver_workflow** under the list of templates in the right panel, and click **OK**. The template opens in *Source* form; switching it to *Visual* form makes it easier to read.





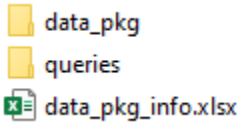
Preparation

Folder Structure

Many of the fetchsqlserver functions look for necessary files in a specific directory where your queries are saved and where templates will be saved, which is referred to as the working directory. You can set the working directory by running **set_pkg_wd()**, which allows you to select a folder using File Explorer.

The folder path is saved as a hidden global variable. If you clear all object from your workspace, including hidden objects, you will need to run this function again.

Two folders are needed inside the working directory. As mentioned in the Data Table Queries section, the folder name for the queries must be *queries*. The other folder, named *data_pkg* in the example below, holds everything needed to create a data package. This is where the metadata files need to be saved. This folder is where the data tables, attribute tables, categorical variable tables, and other metadata files will be saved.



Data Package Information

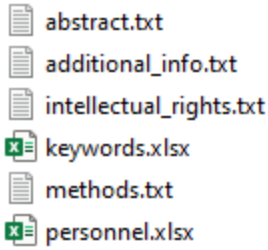
To make metadata creation easier, a spreadsheet, **data_pkg_info.xlsx**, is used to capture the information needed for **create_EML_metadata()**. Additionally, the SQL Server name and project database name are specified in the spreadsheet and used by **fetchsqlserver** to read the queries and data tables. To save the template, run **save_data_info_template()** in the console or a markdown document. The spreadsheet is saved in the working directory. In the **overall** sheet, there are notes in each column header cell describing what information to fill out for each column.

The **custom_units** sheet already contains examples of custom units. Any *numeric* data type in a data table must have a unit associated with it (tinyint, smallint, int, bigint, float, decimal, numeric, real). The units_list sheet contains the available units, but a more user-friendly version (and more up-to-date version) can be found by running **EMLassemblyline::view_unit_dictionary()**. This allows you to more easily search the unit dictionary. Any units that cannot be found in the unit dictionary need to be added as custom units. There is no need to delete the custom units exampls unless you feel the need to.

Once **data_pkg_info.xlsx** is filled out, run **read_data_pkg_info()**. This function saves all the information from the spreadsheet into a hidden variable in the R project, and it must be run before fetching the data tables.

Metadata Files

The **fetchsqlserver** package needs certain files to use throughout the data package creation process. You need one folder to save the data package files, and you can name it whatever you like. This folder holds all the files needed for data package creation, including data tables and metadata files. Many of the necessary files are generated by fetchsqlserver; however, the following files need to be created/populated by the user:



You can manually create these files, or you can use the **fetchsqlserver** function **save_metadata_templates()** to save blank templates and fill them out. When **save_metadata_templates()** is run, the function has the user select the folder to save the files to. The selected folder should be where the data tables, attribute tables, categorical variable tables, and other metadata files will be saved (the folder named *data_pkg* in the Folder Structure section). These are the same files that are saved by **EMLassemblyline::template_core_metadata()**, but with a couple of exceptions. The keywords and personnel files saved by **EMLassemblyline** are text files/TSVs (tab-separated value files), and **fetchsqlserver** saves them as Excel spreadsheets. We recommend using **fetchsqlserver** to save the templates since those are the file formats used by the package. If you are unsure about how to create an R project or install **fetchsqlserver**, see the Getting Started section.

Do not change and of the file names!

Populating Templates

Below are descriptions of the different metadata templates. For the most part, the descriptions were copied from the EMLeditor template, the NPS Data Publication Best Practices SharePoint, and the Ecological Metadata website.

Abstract

Your abstract will be forwarded to *data.gov*, DataCite, Google dataset search, etc., so it is worth some time to carefully consider what is relevant and important information for an abstract. Abstracts must be greater than 20 words. Good abstracts tend to be 250 words or less. You may consider including the following information: The premise for the data collection (why was it done?), why is it important, a brief overview of relevant methods, and a brief explanation of what data are included such as the period of time, location(s), and type of data collected. Keep in mind that if you have lengthy descriptions of methods, provenance, data QA/QC, etc., it may be better to expand upon these topics in a Data Release Report or similar document uploaded separately to DataStore.

Additional Info

Any additional information relevant to the understanding of the dataset.

Intellectual Rights

Currently **save_metadata_templates()** inserts a Creative Common 0 license. The CC0 license is updated during the metadata creation process depending on the user selections when running **gather_nps_info()**. There is no need to edit this .txt file.

Keywords

Using precise keywords chosen from a controlled vocabulary improves data discoverability. A controlled vocabulary is a standardized, organized arrangement of terms that provide a consistent way to describe

data. References with keywords drawn from a controlled vocabulary are amenable to indexing which facilitates content retrieval through browsing and searching. For example, if all 32 data managers apply the keyword “precipitation” to data packages that include measurements of water deposition, then a data user can find all references by searching for that one term, rather than searching for rain, rainfall, hail, and snow, as well as precipitation.

Methods

Describes the methods followed in the creation of the dataset, including description of field, laboratory and processing steps, sampling methods and units, quality control procedures. Used to describe the actual procedures that are used in the creation or the subsequent processing of a dataset.

Personnel

You must list at least one person with a "creator" role and one person with a "contact" role (you can list the same person twice for both.). Creators will be authors and included in the data package citation. You do not need to list anyone as the "PI" and can list as many additional people with custom roles as desired (e.g. "field technician", "laboratory assistant"). Each individual must have a surName. The electornicMailAddress is an email and the userId is the person's ORCID (if they have one). To add a park unit to the list of personnel, use the park code for the surName, leave electronicMailAddress and userId blank, and fill in the rest as you would for any other personnel. List the ORCID as just the 16-digit string, do not include the https://orcid.org prefix (e.g. xxxx-xxxx-xxxx-xxxx). You can leave the projectTitle, fundingAgency, and fundingNumber blank.

Data Table Queries

The package uses SQL queries to fetch the data tables from SQL Server. **You must save the query files in a folder named “queries”**, and only queries should be saved in this folder. You can save the queries as .sql or .txt files. **The queries must follow a specific format** for the package to generate the necessary metadata needed for publication to the DataStore.

- The first line must be SELECT and nothing else.
- Each field must be on its own line, and it must start with the table alias.
- Field/column names can have aliases.
- Each join must be on its own line.

```
SELECT
l.[ParkCode],
l.[LocationName],
e.[StartDate] AS EventDate,
e.[StartTime],
cf.[ConstrainingFeature],
lcf.ConstrainingFeatureDesc
FROM [data].[ConstrainingFeatures] AS cf
LEFT JOIN [data].[Events] AS e ON cf.EventID = e.EventID
LEFT JOIN [data].[Locations] AS l ON e.LocationID = l.LocationID
LEFT JOIN [lookup].ConstrainingFeature AS lcf ON cf.ConstrainingFeature = lcf.ConstrainingFeature
ORDER BY EventDate
```

The query file name is used to fetch the table descriptions and is the name used to save the data table CSV. For example, if the query file is `ConstrainingFeatures.sql`, `fetchsqlserver` will fetch the table

description for **ConstrainingFeatures** and use that as the data file description when creating the metadata. When the data table is saved as a CSV, the file name will be **NTWK_ProjectName_ContrainingFeatures.csv** (ex: **NCCN_WaterQuality_ConstrainingFeatures.CSV**).

You can also use any table name you want, whether the data table exists in the database or not, and fill in or update the data file description later. For instance, you have a query file named **SamplingEventsAndConditions.sql** because you want the table name to be **SamplingEventsAndConditions**. No such table exists in the database, so the data file description is left blank when table descriptions are fetched. After the data package is saved, the **data_names_descriptions.xlsx**, which contains the data file names and descriptions, can be opened and updated before running **create_EML_metadata()**.

Creating the Necessary Files

The **fetchsqlserver_workflow** R Markdown template walks through the workflow of the NCCN Water Quality data package creation. The workflow utilizes many of the functions in fetchsqlserver, and it reveals a lot of the nuances involved with data package creation using fetchsqlserver. It is highly recommended to look through the

Fetching from SQL Server

The function **fetch_sql_server_data()** retrieves data tables, data table names and descriptions, and data table attributes from the SQL Server database. The data tables are fetched using the queries described above. Additionally, **data_pkg_info.xlsx** needs to be filled out by the user prior to running **fetch_sql_server_data()**. See the Data Package Information section for more details.

A large list is returned by **fetch_sql_server_data()**, which will be referred to as the **data_list**, and it contains three items:

- A list containing data tables in data frames.
- A list containing the data table attributes (attribute tables). Each table is formatted so that it can be used to create EML metadata with the exception of two columns - RefSchemaName and RefTableName. Only fields with a foreign key relationship with have these columns filled in. These are used to create categorical variable tables later on.
- A data frame containing the data table name and description with the column headers **dataFile** and **dataFileDescriptions**. This is used during EML metadata creation.

Additionally, two CSVs are saved to the working directory when the attribute tables are fetched - **catvar_list.csv** and **field_unit_dict.csv**. **catvar_list.csv** contains the schema and table names for any data table attributes with foreign keys. The CSV also contains a Code and Description column for the lookup/ref table column names associated with the Code and Description. These values are filled out automatically with the first column of the lookup/ref table as the Code, and the second column as the Description. The user must verify these are accurate prior to running **create_catvar_tbls()**.

field_unit_dict.csv contains a list of fields that have an EML class of *numeric* (tinyint, smallint, int, bigint, float, decimal, numeric, real). The unit for each field must be filled out by the user prior to running **add_units()**. The unit must be in the unit dictionary, which can be found by running **EMLassemblyline::view_unit_dictionary()**, or in the third sheet of **data_pkg_info.xlsx**. It is easier to

search the EMLassemblyline dictionary. If the unit is not in the unit dictionary, it must be in the custom_units sheet of data_pkg_info.xlsx.

Data Wrangling and Flattening

There are four functions you can use to interact with the data_list returned by fetch_sql_server_data():

- get_tbl_as_df(table_name, table_type = "data", data_list) pulls the desired data frame from the data_list. The table_type can either be "data", which is the default, or "attr".
- rearrange_attr_tbl(data_df, attr_df) rearranges the attribute table rows to match the data table column order. Any attribute table rows that do not match a data table column are not kept. This is useful when you add rows to the data table, and therefore attribute table, so you only have to organize the data table columns. However, rearrange_attr_tbl() is run in save_data_pkg() before actually saving the files. As a result, it is not necessary to run this function during data flattening and wrangling unless desired.
- update_data_list(table_name, data_list, data_df = NULL, attr_df = NULL) replaces a data frame(s) in the data_list. You can replace just the data table, just the attribute table, or both.
- remove_from_data_list(table_name, data_list) removes the data table and attribute table from the data_list and removes from the table names and descriptions data frame.

When wrangling and flattening data, there are a few things to keep in mind. If you add a field to a data table, you must also add that field to the associated attribute table. If the field is *numeric*, you must also add the field and unit to field_unit_dict.csv. For example, if the user adds Longitude and Latitude fields, they must add both fields to the attribute table, along with attribute definitions, and the data type. The EML classes for the new columns can be added to the attribute table data frame instead of the SQL Server data type. If you feel more comfortable adding the data type instead of EML class, any SQL Server data types are converted to EML classes when save_data_pkg() is run. Since both fields are *numeric*, Longitude and Latitude must be added to the field_unit_dict.csv along with the unit degree.

If you extract data tables and/or attribute tables from the data_list for wrangling/flattening, you need to update the tables in the data_list. Data package creation tends to be an iterative process, especially if it's the first data package for a protocol. As a result, it is recommended to save the return of update_data_list() as a new data_list in case you need to go back to the original data tables and attribute tables.

Attribute Tables

The attribute tables need to be properly formatted for EMLassemblyline to read them for EML metadata creation. To do that, the following functions should be run:

- add_units(data_list) adds units for *numeric* EML classes using field_unit_dict.csv.
- missing_values_all_tbls(data_list) adds generic missing value information to all tables using information from data_pkg_info.xlsx. Or you can use missing_values(data_df, attr_df = NULL)

to add generic missing value information to an individual attribute table (pass in an attribute table along with the data table) or get a list of columns from the attribute table that have missing data (only pass in a data table). If there is already existing missing value information present, it is not replaced. This allows the user to fill in custom missing value information for certain fields before adding the generic missing value information to the remaining fields.

Categorical Variable Tables

Catvar tables describes the categorical variables of a data table with the *code* (value you see in a data table) and *definition* (what the code means or represents), which is more commonly referred to as the *description*. These tables are necessary to create EML metadata. The attribute tables in the **data_list** have two columns, *RefSchemaName* and *RefTableName*, which are used to identify attributes with catvars. Additionaly, values from **catvar_list.csv** is used to query the database to populate the *code* and *definition* columns of the catvar tables.

The catvar tables are created using **create_catvar_tbls(data_list)**. A list containing the **data_list**, a vector containing the catvar table names, and a list of catvar tables is returned. As a result, it is best to save the return of **create_catvar_tbls()** as a new variable and extract the items afterwards. An example can be found in the fetchsqlserver workflow template (instructions for opening the template can be found in the Getting Started section).

Create the Data Package and EML Metadata

The 'base' EML consists of:

- The overall package details, metadata file name, package title, data collection status, data table files, names and descriptions, and temporal information, which is read from **data_pkg_info.xlsx** with **read_data_pkg_info()** or **read_overall_sheet()** and collected with **gather_eml_info()**.
- Core metadata information, abstract, methods, additional info, personnel, and keywords files, whose templates can be saved using **save_core_metadata()** (see Metadata Files and Populating Templates sections). **These files must be filled out prior to running create_eml_metadata(eml_info)**.
- The information in the attribute and catvar tables, which are saved when you run **save_data_pkg(data_list, catvar_list)**.
- Geographic and taxonomic coverage, which are created with **gather_eml_info()**.
- Custom units, if applicable, **read_data_pkg_info()** or **read_cust_units_sheet()** saves this file.

Save the Data Package

When **save_data_pkg(data_list, catvar_list)** is run, the data tables are saved as CSVs, the attribute and catvar tables are saved as TSVs, and the table descriptions data frame is saved as an XLSX in the data package folder (the folder named *data_pkg* in the Folder Structure section). The function automatically

adds the network code and project name from **data_pkg_info.xlsx** to the beginning of all the data table names. Example: *NCCN_WaterQuality_ChannelConstraints.csv*.

Creating Base EML

The base EML metadata information is put together using **gather_eml_info()**. Different data will be pulled from files, and the user is asked for various inputs. The function writes the necessary files to create the EML metadata (*geographic_coverage.txt* and *taxonomic_coverage.txt*) to the data package folder. The output is a hash dictionary that stores all the base EML information.

The next step is to create the base EML metadata. **create_EML_metadata(eml_info)** ensures all necessary core metadata files are present, converts *keywords.xlsx* and *personnel.xlsx* to TSVs, reads the information stored in the **eml_info** hash dictionary, and passes the information to **EMLassemblyline::make_eml()**, which creates an **EML_object**.

Add NPS-specific Metadata

NPS-specific metadata includes park units, producing units, intellectual rights, CUI, and creating a DataStore reference, which is put together using **gather_nps_info(eml_info)**. The **eml_info** object is passed into the function so the NPS-specific metadata user selections can be saved as well. The updated **eml_info** is passed into **create_nps_eml(eml_info)** to create complete metadata.

Save EML to an XML File

As long as the EML is valid, the EML needs to be saved as an XML file using **write_eml_xml()** (which will check the validity of the EML as well). The data package folder should contain the data tables, attribute tables, catvar tables, and metadata files. During the data package creation process you were likely prompted to select the data package folder; that is where **write_eml_xml()** will look for the files and save the XML file to. Many messages are produced when the EML is checked. If your schema is valid, you should see something like this:

—— Reading metadata ——

—— Checking metadata compliance ——

✓ Your metadata is schema valid.

There is also the option to update the **eml_info** object with the DataStore Reference ID. If there was not an existing DataStore reference for this data package, then the Reference ID was not saved in the **eml_info** object. To save it, pass in the **eml_info** object and save it as a variable. Otherwise, simply run **write_eml_xml(my_metadata2)** without passing in the **eml_info** object. Example XML file name: *NCCN_WaterQuality_DataPackage_2010-2021_metadata.xml*.

Upload Data Package to the DataStore

Before uploading the data package to the DataStore, congruence checks need to be ran using **DPchecker::run_congruence_checks(data_pkg_folder)**. The folder path to the data package needs to be provided.

If everything checked out, you should be ready to upload your data package! It's recommended to use **EMLeditor::upload_data_package(data_pkg_folder)** to accomplish this. The function automatically checks the DOI and uploads to the correct reference on DataStore. All of the files for the data package need to be in the same folder, there can be only one *.xml* file (ending in "*_metadata.xml*"). Each individual file should be < 32Mb. If you have files > 32Mb, you will need to upload them manually using the web interface on DataStore.