CSSP Desktop

Components (.dll, .exe)

CSSPCultureServices – used to hold all the text information in (fr and en)

CSSPCultureServices.Tests – used to test the CSSPCultureServices

CSSPEnums – used to hold all the Enums and related Enums functions

CSSPEnums.Tests – used to test the CSSPEnums

CSSPModels – used to hold all data types and db contexts

CSSPModels.Tests – used to test the CSSPModels

CSSPServices – used to interact with dbs and also hold all required services used in other apps and web apis

CSSPServices.Tests – used to test the CSSPServices

CSSPWebAPIs – is the main interface between client apps and web apis

CSSPWebAPIs.Tests – used to test the CSSPWebAPIs

CSSPDesktop – is the actual local application

It can be installed using <https://csspdesktopinstall.azurewebsites.net/>

CSSPDesktopServices – is the service dll for the CSSPDesktop application

CSSPDesktopServices.Tests – used to test the CSSPDesktopServices

CSSPDesktopInstall – used to package CSSPDesktop for easy transfer to <https://csspdesktopinstall.azurewebsites.net/>

GenerateAllGzFiles – used to generate all the JSON files stored on azure (<https://portal.azure.com/#home>) or local (C:\CSSPDesktop\csspjson\)

PolSourceGroupingExcelFileReadServices – used to read the C:\CSSPTools\src\assets\PolSourceGrouping.xlsm file. This dll is temporary until a more appropriate solution is found which will use the CSSPDB to hold all the Groupings information.

PolSourceGroupingExcelFileReadServices.Tests – used to test PolSourceGroupingExcelFileReadServices

Azure Components (<https://portal.azure.com/#home>)

App Services

* CSSPDesktopInstall – used to help users with the installation and updates of CSSPDesktop
* CSSPWebAPIs – used to access CSSPDB (now temporary called CSSPDB2)
* CSSPLabSheet – used with the Desktop application (CSSPWQInputTool) (this tool will be replaced by the new CSSPDesktop)

Databases

* CSSP – used with CSSPLabSheet (app services) and CSSPWQInputTool (desktop client) (temporary, will be removed in the future once CSSPDesktop is completed)
* CSSPDB2 – used with CSSPWebAPIs (app services) and CSSPDesktop (desktop client). This DB will be renamed to CSSPDB in the future version

Storage

* csspstorage – used to store static information that could have been pre-generated or uploaded
  + csspapps – used to store apps related information
    - csspclient.zip -- Angular client app (this file should be unzipped under C:\CSSPDesktop\csspwebapis\csspclient\)
    - csspwebapis.zip -- local version of CSSPWebAPIs (this file should be unzipped under C:\CSSPDesktop\csspwebapis\)
    - cssphelpdocs.zip – HTML help docs (this file should be unzipped under C:\CSSPDesktop\cssphelpdocs\)
  + csspjson – used to store all the pre-generated compressed JSON files. These files will be regenerated when required by the CSSPWebAPIs server version hosted by Azure CSSPWebAPIs app service. All compressed JSON files (in total ~ 24000 files) are stored using various names like shown below where nnn is the CSSPDB TVItems unique number (TVItemID):
    - WebArea\_nnn.gz (Area related information)
    - WebClimateDataValue\_nnn.gz (Climate Data Value for a particular Climate Site)
    - WebClimateSite\_nnn.gz (Climate Site under a province)
    - WebContact.gz (Contact related information)
    - WebCountry\_nnn.gz (Country related information)
    - WebDrogueRun\_nnn.gz (Drogue runs for a particular subsector)
    - WebHydrometricDataValue\_nnn.gz (Hydrometric Data Value for a particular Hydrometric Site)
    - WebHydrometricSite\_nnn.gz (Hydrometric Site under a province)
    - WebMikeScenario\_nnn.gz (Mike Scenario related information)
    - WebMunicipalities\_nnn.gz (Municipalities under a particular province)
    - WebMunicipality\_nnn.gz (Municipality related information)
    - WebMWQMRun\_nnn.gz (MWQM Runs for a particular subsector)
    - WebMWQMSample\_nnn\_nnn2\_nnn3.gz (MWQM samples for a particular subsector between years “nnn2” and “nnn3” inclusively.
    - WebMWQMSite\_nnn.gz (MWQM Sites for a particular subsector)
    - WebPolSourceGrouping.gz (Groupings related information use for pollution source site information and issues identification)
    - WebPolSourceSite\_nnn.gz (Pollution source site related information for a particular subsector)
    - WebProvince\_nnn.gz (Province related information)
    - WebReportType.gz (Report type related information)
    - WebRoot.gz (Root related information)
    - WebSamplingPlan\_nnn.gz (Sampling plan related information)
    - WebSector\_nnn.gz (Sector related information)
    - WebSubsector\_nnn.gz (Subsector related information)
    - WebTideLocation.gz (Tide location related information)
  + csspfiles – used to store all the CSSP files (.docx, .xlsx, .gif, .png, .dfsu, etc…)
    - All files are stored with the TVItemID as the subdirectory. So for example all files stored under the C:\CSSPDesktop\csspfiles\1\ are related to Root while subdirectory ending with …\5\ are all related to Canada and so on. File name under a particular needs to have unique names.

Local Components (C:\CSSPDesktop\)

Directories

* csspfiles – used to hold a copy of the Azure storage csspfiles (the content of this directory is managed by CSSPWebAPIs local version and the use of SQLite database CSSPDBFilesManagement.db)
* csspjson – used to hold a copy of the Azure storage csspjson (the content of this directory is managed by CSSPWebAPIs local version)
* csspwebapis – used to hold the required files (csspclient and CSSPWebAPIs.exe and related/required files). The CSSPDesktop tool will run the CSSPWebAPIs from this location. The Angular (.js) and Index.html is also stored within this subdirectory under csspwebapis/csspclient
* cssplocaldatabases – used to hold local database which are used by CSSPDesktop and CSSPWebAPIs. All the databases listed below will be created the first time CSSPDesktop is install and verified after each update. If the structure of the CSSPDB change then it could be that the CSSPDBLocal.db needs to be recreated. The users will be asked to sync all their changes to Azure before updating is permitted.
  + CSSPDBFilesManagement.db (SQLite) – used to help manage the sync of local and server csspfiles
  + CSSPDBLocal.db (SQLite) – used to store CSSPDB changes locally until the user decides to upload all its local changes to Azure. This is comparable to CSSPWQInputTool and CSSPPolSouceInputTool local storage (.txt files) but in a db like format. This database has identical tables and fields as Azure CSSPDB2 (to be renamed CSSPDB in the future).
  + CSSPDBLogin.db (SQLite) – used to store local Login information and user preferences. When the users is not connected to the internet, they still require to identify themselves through login. When not connected to the internet, it will use this database to login. When connected to the internet it will use the Azure CSSPDB(2) database. The first time the users install the CSSPDesktop application, they will be asked to login with the Azure DB. A local copy of the information required is then stored locally for later use.

Certificate for windows install

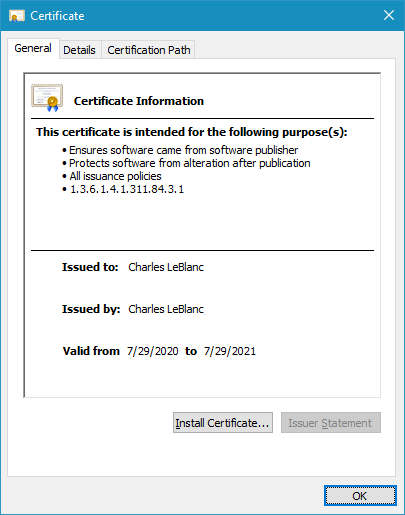
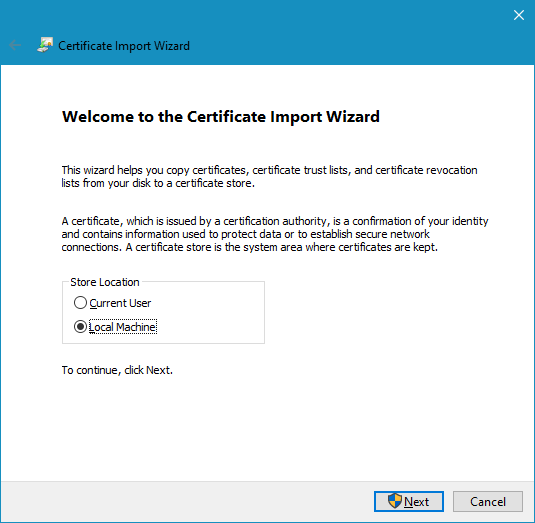
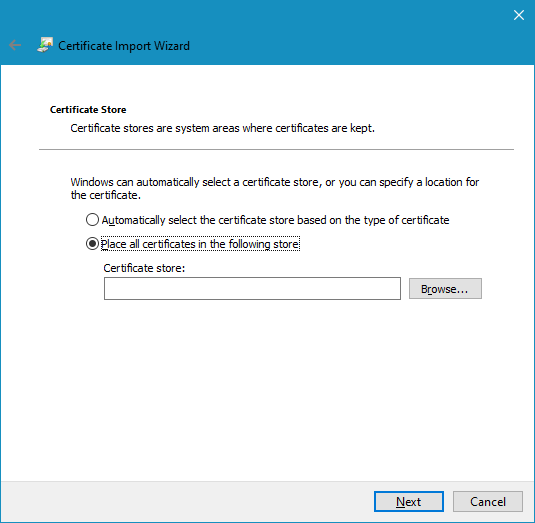
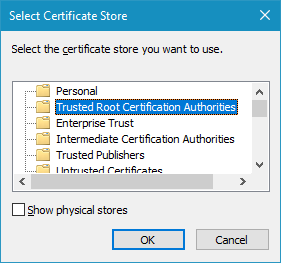
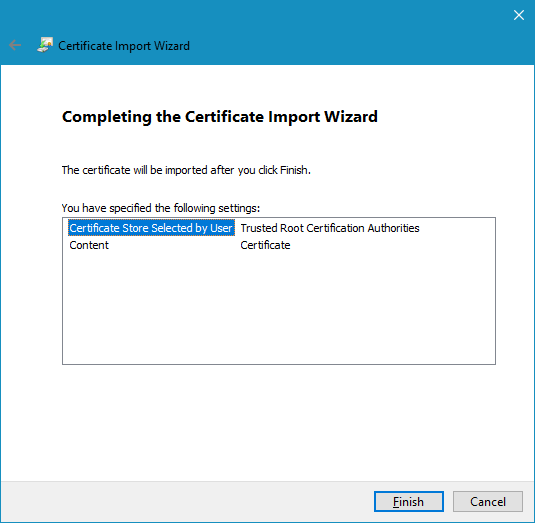
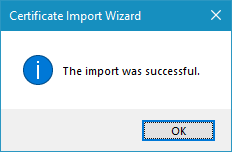
C:\CSSPTools\src\assets\CharlesLeBlanc\_CSSPDesktop.cer

**NOTE:** New software installed on Windows 10 now requires a trusted certificate from the person/company wanting to install their application.

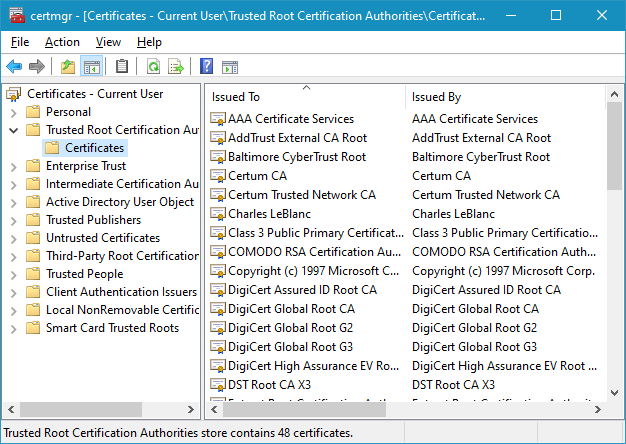
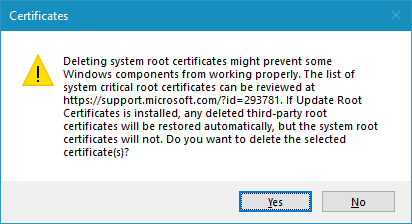
Usually, companies would get these trusted certificates from certificate providers (specialized companies responsible to verify your identity). Like opening an account at a bank. To save 100 $ / year all users will have to manually install a temporary certificate (CharlesLeBlanc\_CSSPDesktop.cer)… if you trust me.

Before the users can install the CSSPDesktop application on their computers they will have to install a certificate (CharlesLeBlanc\_CSSPDesktop.cer. Here is how.

**Installing a Certificate:**

1. Copy the attached file CharlesLeBlanc\_CSSPDesktop.docx on your desktop
2. Rename it to “CharlesLeBlanc\_CSSPDesktop.cer” or “anything\_else.cer”. (extension has to be .cer)
3. Double click on the CharlesLeBlanc\_CSSPDesktop.cer file
   1. 
4. Click on the Install Certificate button
   1. 
5. Select Local Machine and click Next
   1. You might be asked to give access to the application via a User Account Control dialog box
   2. Click Yes
   3. 
6. Select Place all certificates in the following store
7. Click Browse…
   1. 
8. Select Trusted Root Certification Authorities … and click OK
9. Click Next
   1. 
10. Click Finish
    1. 
11. Click OK

**Uninstalling the Certificate:**

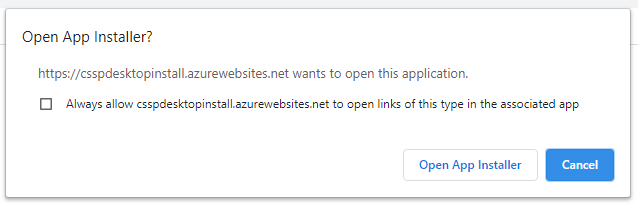
1. Press the windows button or start writing Certificate on the bottom left search panel
2. You should see “Manage User Certificates”. Clicking on it should open.
   1. 
3. Go under “Trusted Root Certification Authorities” -- > “Certificates”
4. Find “Charles LeBlanc” and click on it. Then you can press “Delete” or “Right Mouse click” and Delete.
   1. 
5. Click Yes

**How to install CSSP Desktop**

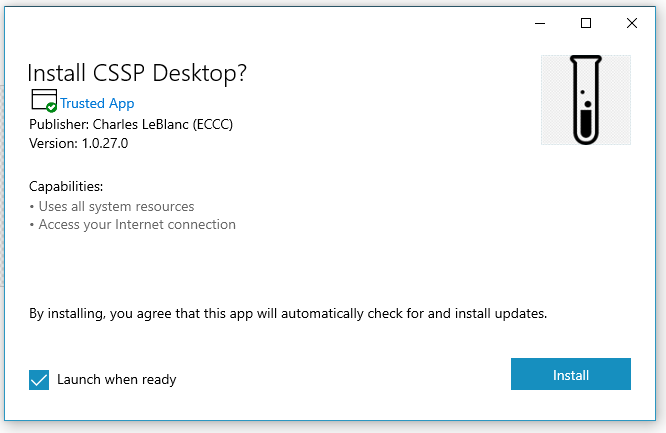
Installing CSSP Desktop is a lot easier than previous CSSP tools once Charles LeBlanc Certificate is properly setup on your computer. The user only needs to load the web site: <https://csspdesktopinstall.azurewebsites.net/>. This will show a web page:



Clicking on the “Get the app >” will start the installation process.



Clicking on “Open App Installer”

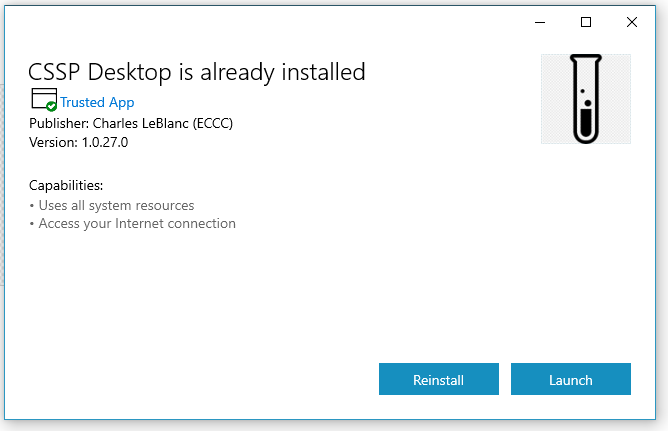


If you don’t see the text “Trusted App” then you must install Charles LeBlanc certificate.

Clicking on “Install” will download and install CSSPDesktop.

To keep the application icon on the bottom task bar just “right click” and select “pin to taskbar”.

Re-accessing the web site <https://csspdesktopinstall.azurewebsites.net/> will and following the same steps will give you:



**Installing Updates CSSP Desktop**

Now that CSSP Desktop is installed, you will not have to worry about updates. Updates will be automatically done. When you run the CSSP Desktop application, it will check to see if updates are available (internet connection required). If updates are available then the next time CSSP Desktop runs it will automatically download and install the updates.

**Uninstalling CSSP Desktop**

Uninstalling CSSP Desktop is as easy as clicking on the Windows button or search area (bottom left of windows) and type CSSP Desktop. This should show the CSSP Desktop app in the best match list. From there you can right button on it and do a lot of stuff, including uninstall.

**Running CSSP Desktop**

Running CSSP Desktop is as easy as clicking on the icon located on the task bar or clicking on the Windows button or search area (bottom left of windows) and type CSSP Desktop. This should show the CSSP Desktop app in the best match list. From there you can directly press [ENTER] or click on CSSP Desktop.

**First Time Running CSSP Desktop**

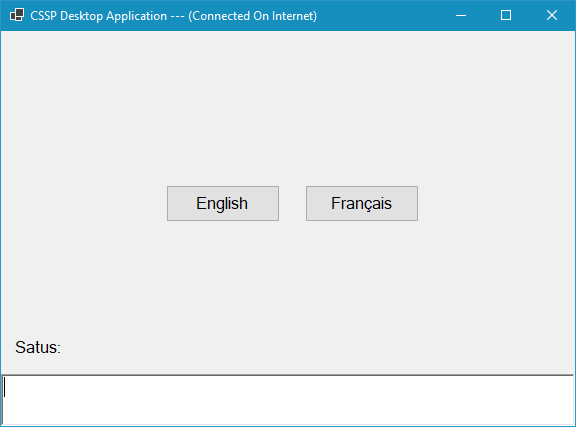
The first time CSSP Desktop runs on your computer it might take some time as it will be setting everything up for it to properly work on your computer. This includes:

* Downloading and unzipping csspclient.zip to C:\CSSPDesktop\csspwebapis\csspclient\
* Downloading and unzipping csspwebapis.zip to C:\CSSPDesktop\csspwebapis\)
* Downloading and unzipping cssphelpdocs.zip to C:\CSSPDesktop\cssphelpdocs\)
* Creating an empty copy of C:\CSSPDesktop\cssphelpdocs\CSSPDBLocal.db
  + Is a local copy of the structure of CSSPDB database located on Azure
  + It is used to hold changed information which need to be synced with the Azure DB
  + It also permits the users to run CSSP Desktop without internet connection for a while and then sync the information back to Azure
* Creating an empty copy of C:\CSSPDesktop\cssphelpdocs\CSSPDBFilesManagement.db
  + Is used by CSSPWebAPIs to help manage and sync the local CSSP files (csspfiles) and Azure CSSP files (csspfiles)
* Creating an empty copy of C:\CSSPDesktop\cssphelpdocs\CSSPDBLogin.db
  + It is used to hold the required information to login without internet connection
  + It also stores all the user preferences for later uses
* You will be asked to Login using your CSSPWebTools credentials.
  + This is a one-time thing necessary for being able to login without internet connection

**How CSSP Desktop Works**

When installing and running CSSP Desktop for the first time, there are many things being setup on the user computer as explained in previous sections. This section will explain in more detail what each part of CSSP Desktop does.

Starting CSSP Desktop



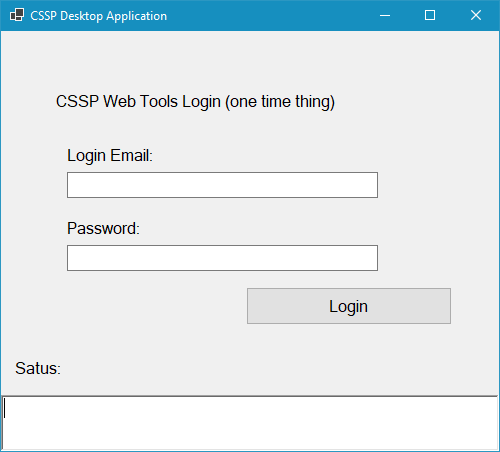
**Title bar**: Shows the name of the application and will indicate if the user has internet connection or not.

**Main Window**: Various things will appear on this main part. The image above shows the language selection page.

**Status bar**: Shows basic help and status of what the application is trying to do

**Detail status section**: The bottom section is where a log of the activities will appear as well as errors when they occur

The first page you will see when running CSSP Desktop is the language selection page. Clicking on a language will determine the language used for CSSP Desktop as well as the User Interface (discussed later).



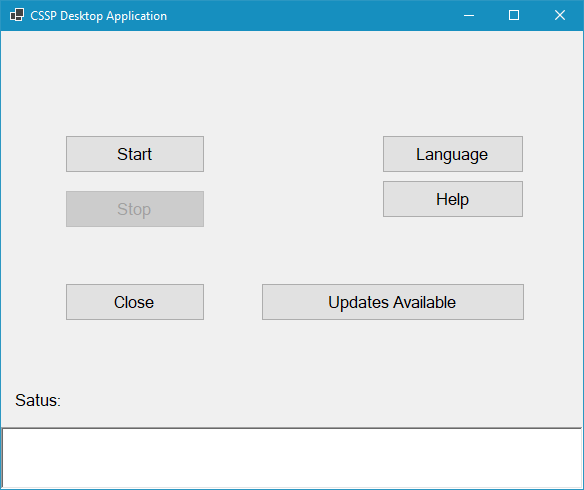
The login page is composed of:

**Login Box**: You need to provide the same Login Email used in the old CSSPWebTools

**Password**: You need to provide the same Password used in the old CSSPWebTools

**Login Button**: Click on the Login button to login.

Note: This is a one-time login thing. You need to be connected to the internet as your login information will be compared with the one used in CSSPWebTools.



The central page is composed of:

**Start Button**: This button will start both a local CSSPWebAPIs and the web user interface (UI).

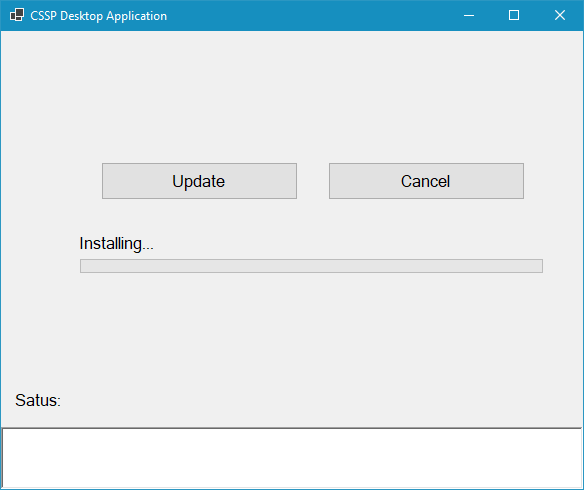
**Stop Button**: This button will close both the local CSSPWebAPIs and the web user interface if opened

**Close**: This button will close the CSSP Desktop application

**Language**: This button will reopen the first language page so you can select the preferred language to use

**Help**: This button opens a detailed help with many how to and SOPs

**Updates Available**: This button will send you to another page to help you update the web component of the CSSP Desktop (CSSPWebAPIs and web UI). Everytime you run CSSP Desktop and you are connected to the internet, the application will check on Azure to see if new updates are available for the web component of the application. If no update is available then the button will be disabled.



The update page is composed of:

**Update Button**: This button will start installing the updates on your local machine

**Cancel**: This button cancels this page and sends you back to your central page

**Installing Progress Bar**: Will track the progress of the installation of the updates

**Detail Status Section**: Will show details of what is being done in the background. If for whatever reason they are errors during the update, this will also be the place to show them.

**Purpose and details of CSSP Web APIs**

This CSSPWebAPIs mini web server located at C:\CSSPDesktop\csspwebapis\csspwebapis.exe is automatically started when the Start button is clicked.

The main purpose of this local mini web server is to be able to provide information requested from the user interface (Local Web Site). It’s also the brain of all commands coming from the local web site. It is responsible in managing all request and knowing when information need to be pulled down from Azure or use the local information stored in C:\CSSPDesktop\csspjson and C:\CSSPDesktop\csspfiles.

It’s also responsible in orchestrating and sending specific tasks to external software (single purpose application) which most of them will be run locally.

* TaskRunnerCSV – used to generate .csv files
* TaskRunnerDOCX – used to generate .docx files (Word Documents)
* TaskRunnerKML – used to generate .kml or .kmz files (Google Earth)
* TaskRunnerMIKE – used to read, parse and interpret .m21fm, .m3fm, .dfs0, dfs1 and .dfsu
* TaskRunnerTXT – used to generate .txt files
* TaskRunnerXLSX – used to generate .xlsx files (Excel Documents)
* TaskRunnerJSON – used to generate .json files (compressed JSON files could also be .zip or .gz

**How CSSPWebAPIs works**

CSSPWebAPIs was developed using asp.net core (Microsoft). It is built with both the local version and the server version (Azure) within the same code.

CSSPWebAPIs (local version) is responsible to send JSON and other files (.png, .jpg, .docx, .xlsx, .txt, etc..) type information to the user interface (web site). It first starts by downloading required JSON and/or other files from Azure and storing them on the local computer, if these files does not already exist on the local computer. It only downloads and store information needed and gradually. It then read the local version of the files and send the information to the web site that requested it.

Note: For the first-time use, you will need internet connection to at least start downloading required files to view them on the web site (UI).

When users make changes on the web site or uploads a new file, the local server will first upload or change the information locally using local version of the database and store the file under C:\CSSPDesktop\csspfiles. Once in a while, when the user has internet connections, the changed and/or new local information and files are sent to Azure in the background. Reports of the background upload/sync will be available for users to see and analyze. Giving them the opportunity to reload the information.

Note: Regarding updating the web component of CSSP Desktop. The user will be asked to make sure all changed information (stored locally) be sent the Azure before being able to accept/do the update. It could happen that some of the local database needs to change in structure and emptying them before changing their structure is important so no information is lost.

Example of logic when the web site requests a web page.

With internet connection:

* Web site request a web page
* CSSPWebAPIs accept the request
* It collects all the .json file required to fulfill the request (could be more than one .json files)
* It then checks all required .json files in the database CSSPDBFilesManagement.db to see if these files are already stored locally or not and if they need an update using Microsoft Azure Store ETag,
* For the .json files not stored locally yet, these files are downloaded from the Azure Store csspjson and stored locally under C:\CSSPDesktop\csspjson\
* They are then read, parsed and stored in an in-memory database using the same structure as CSSPDB
* The JSON information is then read from the in-memory database (might use LINQ to only send partial/filtered information to the client)
* The same process is done for normal files like (.xlsx, .docx, .dfsu etc)
* The name of these files to be downloaded are within the .json files
* The files are then downloaded from Azure Store csspfiles and stored under C:\CSSPDesktop\csspfiles\

Example of logic when information is changed or added from the web site (client UI)

* When user change or add new stuff using the web site, the information is temporary stored locally using C:\CSSPDesktop\cssplocaldatabases\CSSPDBLocal.db (SQLite)
* CSSPDBLocal.db has the same database structure as CSSPDB located on Azure
* Changed information is stored in the database with the same identifiers used in CSSPDB (Azure)
* New information however is stored in the database with very high values as identifiers
* The local CSSPDBLocal does not have any relationship between tables in order to be able to use the correct identifies without uploading all necessary tables and information within the local database (CSSPDBLocal)
* CSSPWebAPIs has to maintain all these relationships itself.
* When .json files are parsed the information is placed into an in-memory database with the same structure as the CSSPDB (Azure) and CSSPDBLocal.
* This in-memory database can then be use to verify all necessary relationships in order to accept the changes or additions to the CSSPDB/CSSPDBLocal
* Once the information is changed the in-memory has to reflect the new changes or additions.
* In-memory database has to be reloaded every time CSSPDesktop is restarted

Example of logic when change or new information is sent to Azure

* User uses the upload to Azure page to manage when to upload
* Information is gathered and sent to Azure parts at a time
* CSSPWebAPIs (server version) receives the information from the user, this could take many back and forth (sending multiple .json file and other types of files (.xlsx, .docx, .dfsu…)
* It then stores the work job id in a table within the CSSPDB database located on the server
* Once all information has been uploaded, parsed, entered in the CSSPDB database and all the files (images, documents etc…) are also uploaded to the server and stored properly using CSSPWebAPIs server
* It then need to recreate all the .json for which some information has changed.
* It then sends the status of the upload to the client in order to inform the client if everything was ok or indicate where some error occurred so the user can fix it
* If everything is ok and all the changed .json file are recreated, an ok message is sent to the CSSPWebAPIs (client)
* The CSSPWebAPIs (client) then needs to remove all information from the C:\CSSPDesktop\cssplocaldatabases\CSSPDBLocal.db that was properly sent and stored in CSSPDB Azure

Purpose of each CSSP Desktop databases

* CSSPDB (Azure SQL Server) is the master database and is considered the correct and final storage of information for the CSSP Desktop
  + It is used to pre-generate .json files which are stored on Azure Store csspjson
  + Every time these are changes made to CSSPDB, some of these .json file might have to be regenerated
  + It is also used to generate other types of files which will be used for Arc GIS and other tools
* C:\Desktop\cssplocaldatabases\CSSPDBLocal.db (local) is the main local database which has the same structure as the CSSPDB (Azure)
  + It stores the user changes or additions temporary until the user decides to upload the changes to Azure
  + It also allows the user to work offline with the .json files (the user has to make sure they download all necessary information before however)
* C:\Desktop\cssplocaldatabases\CSSPDBFilesManagement.db (local) is use to manage the downloaded files
  + Keeps the names, date created and Etag for each file downloaded from Azure csspjson and csspfiles
  + It can then verify the Etag to see if the file has changed on Azure since the last download
* C:\Desktop\cssplocaldatabases\CSSPDBLogin.db (local) is use to store relevant information for the user to be able to login without being connected to the internet
  + Used mainly to be able to login to the web site (client UI) when the user is not connected to the internet
  + It also saves some basic user preferences
* In-Memory database (local) is use for quick access to information which the user might request via the web site (client UI)
  + It’s main purpose is for speed
  + It’s also used to make sure the user changes and additions are properly verified with relational tables
  + It’s very much dynamic following the users needs and location on the web site
  + It only store what is needed and removes the rest, so if a user goes to another subsector it will remove the information from the previous subsector and add the information from the new subsector

Advantage and purpose of using Angular Client

* Angular framework, developed by Google, has the advantage of pre-compiling all javascript and html template into some single javascript files which can and is downloaded only once.
* CSSPDesktop is responsible to download/update the csspclient (Angular) into C:\CSSPDesktop\csspwebapis\csspclient\
* Once downloaded, it does not have to be downloaded again until a new update is available
* This pre-download and pre-compile save a lot of unnecessary downloads from Azure
* Adding to this that most of the information from CSSPDB is pre-generated into .json file which are compressed at > 95%.
* These .json files are also just downloaded once and stored on the user computer under C:\CSSPDesktop\csspjson\
* Files like (Excel, Word, etc) are also only downloaded once and stored on the user computer C:\CSSPDesktop\csspfiles\
* These are only re-downloaded when they change on Azure

Purpose of storing CSSPDB (Azure) content in pre-generated .json files

* In the old CSSPWebTools every time the user requested a web page the information had to be taken from the CSSPDB using SQL Server, Filtered and interpolated into something like a template which created information of .html type (web page). Some of these web pages could start to be big (> 1 MB). So every time one user would hit that page the all of these process had to be done.
* With the new way of doing things, having .json files pre-generated and zipped (often zipping at >95%), the information being transferred is reduced considerably
* It also allow the user to work offline (no internet connection)