Bringing back C:\ file access to Business Central

With the move to cloud and universal code, Microsoft for obvious reasons has removed access to local files from AL. The alternative is to use cloud storage like OneDrive, SharePoint, Azure Blob storage, or Dropbox.

However, there is nothing which can replace the simplicity of accessing local files, and using cloud storage comes with a cost if you have many and/or large files. Also, there is still a lot of legacy software that does not support writing to or reading from cloud storage.

To mitigate this, ForNAV has created the ForNAV File Service solution that comes with the ForNAV Customizable Report Pack, which has the benefits of being very easy to set up and also has a rich file API.

Using the ForNAV File Service API is as simple as the old Business Central file API:

```
procedure WriteMyFile()
var
  FileService: Codeunit "ForNAV File Service";
begin
  FileService.WriteText('DEMOFILES:\test.txt', 'Hello world ' +
        Format(CurrentDateTime), "ForNAV Encoding"::"utf-8", true);
  Message('The file was written.');
end;
```

The API is heavily inspired by the .net file API, which supports almost any file/directory operation and it does not have the limitations that the old Business Central API had.
Directory API:

- Test if exist, Creating, Copying, Moving, Scanning, and Deleting directories.
- Move files from one directory to another.
- Automatic creation of sub-paths when needed.

File API:

- Complete set of CRUD methods.
- Text file API with support for ReadAllText, ReadLine, and all Windows codepages and encodings.
- Reading and writing file BOMs.
- Advanced error handling.
- Batch operations.

The ForNAV File Service is easy to install, can be set up on multiple windows machines and is designed for security:

- Only uses outgoing HTTPS (on port 443) and does not require ingoing HTTPS, which means no firewall issues!
- Data only passes through the servers hosting Business Central and the File Service.
- File/Directory access is configured so it is restricted to parts of a drive by creating path aliases.
- Error messages are written to the Windows event log.

Getting started

To set up the File Service, you need to edit the **C:\ProgramData\ForNAV\Direct Print\Configuration\config.json** file to add aliases:

```
{
 "Version": 1,
 "Environments": [
  {
   "Name": "default",
   "MemoryLimit": 0,
   "FileShares": [
    {
     "Alias": "DEMOFILES",
     "Path": "%programdata%\\ForNAV\\Direct Print\\Demo Files"
    }
   ],
```

To use the ForNAV File Service, you need to add the dependency:

```
        { "id": "83326d6d-11f8-49fd-981a-6f266a7c8d81", "publisher": "ForNAV",
"name": "Customizable Report Pack", "version": "7.2.0.1" }
```

And now you are ready to start writing code against the API:

```
codeunit 50000 "File Service Test"
{
    procedure ScanDir()
    var
        TempFileDirectory: Record "ForNAV File Directory" temporary;
        FileService: Codeunit "ForNAV File Service";
    begin
        FileService.ScanDir('DEMOFILES:', '*.*', true, TempFileDirectory);
        if TempFileDirectory.FindFirst() then;
        TempFileDirectory.SetFilter(ShareDirectory,
TempFileDirectory.LinkDirectory);
        page.RunModal(Page::"ForNAV File Directory", TempFileDirectory);
    end;

    procedure WriteFile()
    var
        FileService: Codeunit "ForNAV File Service";
    begin
        FileService.WriteText('DEMOFILES:\test.txt', 'Hello world ' +
Format(CurrentDateTime), "ForNAV Encoding"::"utf-8", true);
```

```
        Message('The file was written.');
    end;

    procedure ReadFile()
    var
        FileService: Codeunit "ForNAV File Service";
        txt: Text;
    begin
        txt := FileService.ReadText('DEMOFILES:\test.txt', "ForNAV
Encoding"::"utf-8");
        Message(txt + '\LastError=' + FileService.LastError());
    end;

    procedure DeleteFile()
    var
        FileService: Codeunit "ForNAV File Service";
    begin
        FileService.DeleteFile('DEMOFILES:\test.txt');
        Message('File was deleted.\LastError=' + FileService.LastError());
    end;

    procedure Exists()
    var
        myFile: Codeunit "ForNAV File Service";
    begin
        message(format(myFile.FileExist('DEMOFILES:\test.txt')));
    end;

    procedure TestFiles()
    var
        TempFileDirectory: Record "ForNAV File Directory" temporary;
        FileService: Codeunit "ForNAV File Service";
        TypeHelper: Codeunit "Type Helper";
        tmpBlob: Codeunit "Temp Blob";
        writeBigText: BigText;
        readBigText: BigText;
        starttime: DateTime;
        testTime: DateTime;
        asyncTime: Duration;
        syncTime: Duration;
        linesToWrite: List of [Text];
        linesToRead: List of [Text];
        fileOutStream: OutStream;
        fileInStream: InStream;
        testPath: Text;
        fn8: Text;
        fn16: Text;
        fn16_2: Text;
        fn16_copy: Text;
        i: Integer;
        b: Boolean;
        txt: Text;
        txt2: Text;
        taskUTF8, taskUTF16, taskReadFromStream : Integer;
        taskDirExistsTrue, taskDirExistsFalse : Integer;
        taskFileExistsTrue, taskFileExistsFalse : Integer;
```

```
        taskDeleteDirectoryTrue, taskDeleteDirectoryFalse : Integer;
        taskScanDir: Integer;
        taskReadLines: Integer;
    begin
        FileService.SetTimeOut(20000);
        testTime := CurrentDateTime();
        tmpBlob.CreateOutStream(fileOutStream);
        starttime := CurrentDateTime();
        testPath := 'DEMOFILES:\filetest-async';

        FileService.SetDevice('XYZ2');

        //
        // SYNC TEST
        //
        starttime := CurrentDateTime();
        testPath := 'DEMOFILES:\filetest-sync';

        // Clean up
        FileService.SetErrorAction("ForNAV File Error Action"::Ignore);
        FileService.DeleteDirectory(testPath + '\filetest', true);

        // Test error handling
        FileService.SetErrorAction("ForNAV File Error Action"::Error);
        if TryDirectoryExists(FileService, 'INVALID_ALIAS:\') then error('An
error should have happened.');
        if not TryDirectoryExists(FileService, testPath) then error('An error
should not have happened.');

        FileService.SetErrorAction("ForNAV File Error Action"::Ignore);
        FileService.DirectoryExist('INVALID_ALIAS:\');
        if FileService.LastError() = '' then error('Last error should be
set.');
        FileService.DirectoryExist(testpath);
        if FileService.LastError() <> '' then error('Last error should not be
set.');

        // Prepare
        FileService.SetErrorAction("ForNAV File Error Action"::Error);
        if FileService.DirectoryExist(testPath + '\filetest') then
Error('Folder should not exist.');
        FileService.CreateDirectory(testPath + '\filetest');
        if not FileService.DirectoryExist(testPath + '\filetest') then
Error('Folder should exist');

        // Write from stream
        Clear(tmpBlob);
        Clear(writeBigText);
        if FileService.FileExist(testPath + '\filetest\test-write-bytes.txt')
then Error('File should not exist');
        writeBigText.AddText('This is a sample output file ' +
Format(testTime));
        tmpBlob.CreateOutStream(fileOutStream);
        writeBigText.Write(fileOutStream);
        tmpBlob.CreateInStream(fileInStream);
```

```
        FileService.WriteFromStream(testPath + '\filetest\test-write-
bytes.txt', fileInStream);
        if not FileService.FileExist(testPath + '\filetest\test-write-
bytes.txt') then Error('File should exist.');

        // Read from stream
        Clear(tmpBlob);
        FileService.SetErrorAction("ForNAV File Error Action"::Error);
        tmpBlob.CreateOutStream(fileOutStream);
        FileService.ReadFromStream(testPath + '\filetest\test-write-
bytes.txt', fileOutStream);
        tmpBlob.CreateInStream(fileInStream);
        readBigText.Read(fileInStream);
        if Format(readBigText) <> Format(writeBigText) then Error('Test
failed: Write and read bytes.');

        // Write lines
        Clear(linesToWrite);
        for i := 1 to 1000 do
            linesToWrite.Add('Line ' + Format(i));
        FileService.WriteLines(testPath + '\filetest\test-write-lines.txt',
linesToWrite, "ForNAV Encoding"::"utf-8", true);

        // Validate read lines
        linesToRead := FileService.ReadLines(testPath + '\filetest\test-write-
lines.txt', "ForNAV Encoding"::"utf-8");
        if linesToRead.Get(3) <> 'Line 3' then Error('Test failed: Reading
line does not match.');
        if FileService.GetText(linesToRead) <>
FileService.GetText(linesToWrite) then Error('Test failed: Reading line does
not match text of written lines.');

        // UTF8
        fn8 := testPath + '\filetest\textfile-utf8 æøå 你好世界.txt';
        FileService.WriteText(fn8, txt, "ForNAV Encoding"::"utf-8", true);

        // UTF16
        fn16 := testPath + '\filetest\textfile-utf16 æøå 你好世界.txt';
        fn16_2 := testPath + '\filetest\textfile-utf16 æøå 你好世界 - 2.txt';
        fn16_copy := 'subdir\textfile-utf16 æøå 你好世界 - copy %1.txt';
        FileService.WriteText(fn16, txt, "ForNAV Encoding"::"utf-16BE",
true);

        // Rename the UTF16 file
        FileService.Move(fn16, fn16_2);

        FileService.SetCurrentDir(testPath + '\filetest');
        FileService.CreateDirectory('subdir');

        // Copy the new file
        // FileService.Copy(fn16_2, StrSubstNo(fn16_copy, 1));
        for i := 1 to 5 do
            FileService.Copy(fn16_2, StrSubstNo(fn16_copy, i));

        FileService.DeleteFile(StrSubstNo(fn16_copy, 2));
```

```
        // UTF-8 read and write
        txt2 := FileService.ReadText(fn8, "ForNAV Encoding"::"utf-8");
        if txt <> txt2 then Error('Test failed: UTF8 text does not match.');
        txt2 := FileService.ReadText(StrSubstNo(fn16_copy, 1), "ForNAV
Encoding"::"utf-16BE");
        if txt <> txt2 then Error('Test failed: UTF16 text does not match.');

        FileService.Copy('subdir', 'subdir 2');

        // Validate directory scanning
        Clear(TempFileDirectory);
        TempFileDirectory.DeleteAll();
        TempFileDirectory.Reset();
        FileService.SetCurrentDir(testPath + '\filetest');
        FileService.ScanDir('.', '*.*', true, TempFileDirectory);
        TempFileDirectory.SetRange(IsDirectory, true);
        TempFileDirectory.SetRange(Name, 'subdir');
        if not TempFileDirectory.IsEmpty then begin
            if TempFileDirectory.Count <> 1 then Error('Test failed: Cannot
find subdir in scandir.');
            TempFileDirectory.SetRange(IsDirectory, false);
            TempFileDirectory.SetRange(Name, 'test-write-bytes.txt');
            if TempFileDirectory.IsEmpty() then Error('Test failed: Cannot
find file in scandir.');
        end else
            Error('Test failed: Scandir did not return any subdir.');
        syncTime := CurrentDateTime() - starttime;

        //
        // ASYNC TEST
        //

        // Clean up
        FileService.SetErrorAction("ForNAV File Error Action"::Ignore);
        FileService.DeleteDirectoryTask(testPath + '\filetest', true);

        // Prepare
        FileService.SetErrorAction("ForNAV File Error Action"::Error);
        taskDirExistsFalse := FileService.DirectoryExistTask(testPath +
'\filetest');
        FileService.CreateDirectoryTask(testPath + '\filetest');
        taskDirExistsTrue := FileService.DirectoryExistTask(testPath +
'\filetest');

        // Write from stream
        taskFileExistsFalse := FileService.FileExistTask(testPath +
'\filetest\test-write-bytes.txt');
        writeBigText.AddText('This is a sample output file ' +
Format(testTime));
        Clear(tmpBlob);
        tmpBlob.CreateOutStream(fileOutStream);
        writeBigText.Write(fileOutStream);
        tmpBlob.CreateInStream(fileInStream);
        FileService.WriteFromStreamTask(testPath + '\filetest\test-write-
bytes.txt', fileInStream);
```

```
        taskFileExistsTrue := FileService.FileExistTask(testPath +
'\filetest\test-write-bytes.txt');

        // Read from stream
        FileService.SetErrorAction("ForNAV File Error Action"::Error);
        tmpBlob.CreateOutStream(fileOutStream);
        taskReadFromStream := FileService.ReadFromStreamTask(testPath +
'\filetest\test-write-bytes.txt');

        // Write lines
        Clear(linesToWrite);
        for i := 1 to 1000 do
            linesToWrite.Add('Line ' + Format(i));
        FileService.WriteLinesTask(testPath + '\filetest\test-write-
lines.txt', linesToWrite, "ForNAV Encoding"::"utf-8", true);

        // Read lines
        taskReadLines := FileService.ReadLinesTask(testPath + '\filetest\test-
write-lines.txt', "ForNAV Encoding"::"utf-8");

        txt := 'My sample text æøå 你好世界';
        for i := 1 to 15 do
            txt := txt + TypeHelper.CRLFSeparator() + txt;

        // UTF8
        fn8 := testPath + '\filetest\textfile-utf8 æøå 你好世界.txt';
        FileService.WriteTextTask(fn8, txt, "ForNAV Encoding"::"utf-8",
true);

        // UTF16
        fn16 := testPath + '\filetest\textfile-utf16 æøå 你好世界.txt';
        fn16_2 := testPath + '\filetest\textfile-utf16 æøå 你好世界 - 2.txt';
        fn16_copy := 'subdir\textfile-utf16 æøå 你好世界 - copy %1.txt';
        FileService.WriteTextTask(fn16, txt, "ForNAV Encoding"::"utf-16BE",
true);

        // Rename the UTF16 file
        FileService.MoveTask(fn16, fn16_2);

        FileService.SetCurrentDir(testPath + '\filetest');
        FileService.CreateDirectoryTask('subdir');
        // Copy the new file
        for i := 1 to 5 do begin
            FileService.CopyTask(fn16_2, StrSubstNo(fn16_copy, i));
        end;

        FileService.DeleteFileTask(StrSubstNo(fn16_copy, 2));

        taskUTF8 := FileService.ReadTextTask(fn8, "ForNAV Encoding"::"utf-
8");
        taskUTF16 := FileService.ReadTextTask(StrSubstNo(fn16_copy, 1),
"ForNAV Encoding"::"utf-16BE");

        FileService.CopyTask('subdir', 'subdir 2');

        FileService.SetCurrentDir(testPath + '\filetest');
```

```
        taskScanDir := FileService.ScanDirTask('.', '*.*', true);

        FileService.RunTasks();

        // Validate binary write and read
        Clear(tmpBlob);
        tmpBlob.CreateOutStream(fileOutStream);
        if not FileService.GetReadFromStreamTaskResult(taskReadFromStream,
fileOutStream) then error('Error reading from stream.');
        tmpBlob.CreateInStream(fileInStream);
        readBigText.Read(fileInStream);
        if Format(readBigText) <> Format(writeBigText) then Error('Test
failed: Write and read from stream.');

        // Validate read lines
        if not FileService.GetReadLinesTaskResult(taskReadLines, linesToRead)
then error('Unable to read lines.');
        if linesToRead.Get(3) <> 'Line 3' then Error('Test failed: Reading
line does not match.');

        // Validate UTF-8 read and write
        if not FileService.GetReadTextTaskResult(taskUTF8, txt2) then
error('Unable to read UTF8 file.');
        if txt <> txt2 then Error('Test failed: UTF8 text does not match.');

        // Validate UTF-16BE read and write
        if not FileService.GetReadTextTaskResult(taskUTF16, txt2) then
error('Unable to read UTF16 file.');
        if txt <> txt2 then Error('Test failed: UTF16 text does not match.');

        // Validate directory exists
        if not FileService.GetDirExistsTaskResult(taskDirExistsFalse, b) then
error('Unable to check for non existing directory.');
        if b then Error('Test failed: The directory should not exist.');
        if not FileService.GetDirExistsTaskResult(taskDirExistsTrue, b) then
error('Unable to check for existing directory.');
        if not b then Error('Test failed: The directory should exist.');

        // Validate file exists
        if not FileService.GetFileExistsTaskResult(taskFileExistsFalse, b)
then error('Unable to check for non existing file.');
        if b then Error('Test failed: The file should not exist.');
        if not FileService.GetFileExistsTaskResult(taskFileExistsTrue, b) then
error('Unable to check for existing file.');
        if not b then Error('Test failed: The file should exist.');

        // Validate directory scanning
        if not FileService.GetScanDirTaskResult(taskScanDir,
TempFileDirectory) then error('Unable to get directory scanning result.');
        TempFileDirectory.SetRange(IsDirectory, true);
        TempFileDirectory.SetRange(Name, 'subdir');
        if TempFileDirectory.Count <> 1 then Error('Test failed: Cannot find
subdir in scandir.');
        TempFileDirectory.SetRange(IsDirectory, false);
        TempFileDirectory.SetRange(Name, 'test-write-bytes.txt');
```

```
        if TempFileDirectory.IsEmpty() then Error('Test failed: Cannot find
file in scandir.');
        // if not (TempFileDirectory.FullName = testPath + '\filetest\test-
write-bytes.txt') then Error('Test failed: Cannot match full name in
scandir.');

        asyncTime := CurrentDateTime() - starttime;

        Message('Async test time: %1\Sync test time: %2', asyncTime,
syncTime);
    end;

    [TryFunction]
    local procedure TryDirectoryExists(var FileService: Codeunit "ForNAV File
Service"; Directory: Text)
    begin
        FileService.DirectoryExist(Directory);
    end;
}
```