Backup Settings for Distributed Applications

By Jean-Yves Fock-Hoon, QA Manager, 4D Inc.

Technical Note 06-13

Overview

When creating a distributed application in 4D 2004, it is often desirable to be able to specify custom Backup preferences that will accompany the application.

The purpose of this Technical Note is to show how to create a custom Backup.XML file (also called the Backup project file) to be used by the 4D Backup system with distributed applications.

Building an Application

The Problem

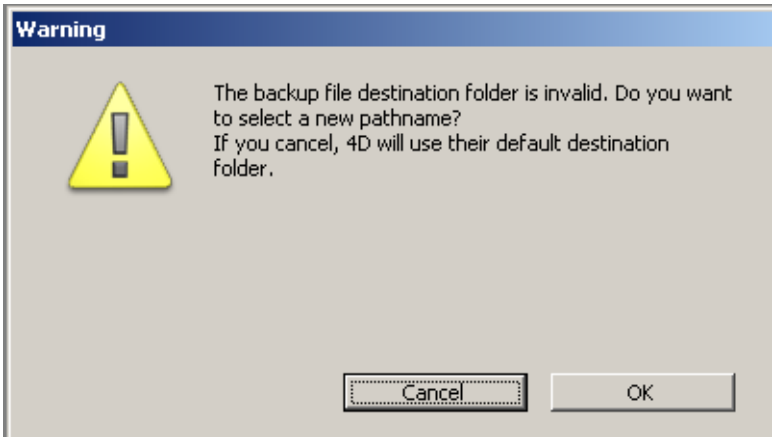
Usually, when developing a 4D database, the developer is focused on getting the main features working. Things like building the redistributable application (based on 4D Runtime or 4D Server) or testing of some basic features such as Backup or the merged application are usually the last steps. This Technical Note focuses on the often overlooked step of setting the Backup preferences for a distributed application.

Here is a typical scenario:

The database development is over. The developer tests the application and tests the Backup system. Everything seems fine. Note that by testing the Backup system a **Backup.XML** project file is created.

The developer is now ready to test the database as a merged Server application. However, once the application has been built, not all files will have been copied inside the Server folder. Some of these files are under the Preferences folder. Therefore, the developer needs to copy some files from the Preferences folder to the new server's folder. However the developer should not just copy the entire Preferences folder since it often contains unnecessary files like the xml file used to build the application.

At this point the application seems to be working on the development machine so the next step is to copy the application onto another machine and launch it. Here is where the first problem is often encountered…



A folder cannot be found. 4D asks the user to select a new folder path. What is going on here? How can this warning message be avoided?

The 4D Backup Project File

The problem comes from the Backup.XML file. In this file there is a path where all Backup files will be saved. Since the database has been moved to a new machine, that path is no longer valid. Furthermore 4D asks that a new Backup path be chosen before the database can launch because the Backup settings of the database, or even the 4D code in the database, may want to run a Backup immediately. Thus 4D needs to have a valid Backup path.

So how can the developer prevent this dialog? One solution is to not to copy the Backup.XML file when moving the database. If a Backup is requested and there is no Backup.XML file, 4D will create a new one with all the default values. At this point the Backup may be launched.

Notice that when a brand new database performs a Backup for the first time, 4D performs the Backup and creates the default Backup.XML file in the *Preferences\Backup* folder, next to the structure file. This is also true for merged applications. Given a merged 4D Server application, with no Backup project file, 4D will still be able to perform a Backup and will generate a default Backup project file for that application. In many cases allowing 4D to proceed with this default behavior is completely adequate.

However, the default values may not match the current needs of the database because the developer might need to define some of the Backup settings, such as the number of archives to keep, some extra files to be archived, some advanced settings to speed up the Backup process or, especially, the developer may have a specific Backup schedule in mind. In this situation the developer needs to provide their own predefined Backup project file. Here is the dilemma: the developer can use their own settings, but they will still see the invalid Backup path dialog since the destination path specified in the Backup.XML may no longer be valid after moving the database.

A Different Approach

Building a Backup.XML File

Given that the developer needs to specify their own values in the Backup.XML file, a better solution might be to create a custom Backup.XML file when starting the application. 4D already has built-in commands to parse and write XML documents. The developer just needs to know the XML tags that will be used. The structure of these tags and the process of generating the document are quite easy.

If the developer does not want the user to be involved in this process, they can create this document from the **On Startup** or **On Server Startup** database method of the application. When starting the database, the developer can check to see if the Backup project file exists or not. If the file does not exist, the developer may create a new XML file. The current user will not notice this operation. The Backup will be ready to be performed.

If the developer wants the user to be informed about the creation of the Backup.XML file and give them the ability to modify some settings, then a dialog could be displayed so that the user can change the settings. Based on the developer's hard-coded values and the user's last modifications, the developer can regenerate the Backup.XML file.

Note that the developer must be sure that the XML generated is well formed, e.g. that all parameters are valid or that there are no extra space characters at the end of the values. Otherwise 4D Backup will generate its own default values for each malformed object.

There is already a demonstration database that shows how to parse and modify the Backup XML document. The inconvenient part of this technique is that the XML code will have to be stored inside 4D, with a list of XML commands that will insert some namespaces, tags and values in the XML tree. If something changes one day, the developer may have to modify that code based on the way the XML is now built.

Another Concept

Another idea is to have the Backup.XML file outside of the application, in an XML file that will be used as a template. If the XML format changes, all the developer has to do is change this template XML file; there is no need to modify the 4D code, unless a new variable must be declared and initialized.

How does this work? The idea is to have a Backup.XML template file in which all of the values are defined by 4D HTML tags. These variables would be pre-defined in the 4D code. Note that the 4D code will not use any XML commands; it will just load the template and execute the **PROCESS HTML TAGS** command on it. Once the template has been processed, it can be saved as Backup.XML file in the Preferences folder for the database.

A template XML file is provided with the demonstration database. Its name is **Backup.XML2** and resides in the *Preferences**Backup* folder.

Here is a summary of the steps involved in this technique:

- After building the merged 4D Server application, the developer is supposed to copy the *Preferences**Backup* folder to the *Server* folder. Instead of copying the Backup.XML file, just copy the template, e.g. Backup.XML2.
- When launching the database, the On Server Startup method will check if a Backup.XML file exists.
- In order to get the location of the *Preferences* folder, the current technique would be to grab the *Extras* folder path (using the **Get 4D folder** command) and compute the *Preferences* folder path. Another technique would be to use the Structure path instead. But the *Extras* folder is simpler and faster.
- Once the path has been computed, a call to **Test path name** on that path can be used to tell if the Backup.XML file exists or not.
- If a Backup.XML file exists, that would mean that the Backup settings have already been set. The database is all set and can continue.
- If the file does not exist this should be a brand new installation. At this point, the developer has a few different choices on how to create the Backup.XML file:
    - The developer can make this completely invisible to the user. They can generate the Backup.XML file with their own pre-defined variables.
    - If the developer wants to give the user the ability to choose the folder right now or later, a request dialog can be used.
    - Finally, the developer might want to offer the user some direct control over the Backup settings. There is an "Advanced" Backup settings dialog provided in the example database. The developer could use the default provided dialog, or use their own dialog, with more or less options.

In the demonstration database a simple dialog is displayed where the user can choose the Backup destination folder. This dialog also contains a button that will display an Advanced Settings dialog that will covert almost all options that the user can see in the default Backup preferences dialog. Once done the PROCESS HTML TAGS command is used to process the Backup.XML2 template and save the result as the Backup.XML file in the Preferences folder. Voila!

The Backup.XML2 File

This file is just a default 4D Backup project with some modifications. Almost all values have been replaced by 4D tags such as 4DVAR with a lot of 4DIF tags for when the values can be different.

Here is the list of all variables used in that template file (and also defined in the Advanced Settings dialog) with the corresponding XML key tag.

Variable Name	Tag Name
BKP_rb_AlwaysWaitBKP	<WaitForEndOfTransaction>
BKP_NbMinWait	<Timeout>
BKP_rb_RetryNextTime	<TryBackupAtTheNextScheduledDate>
BKP_at_timeretry	<TryToBackupAfter>
BKP_cbCancelRetryBKP	<AbortIfBackupFail>
BKP_NbTries	<RetryCountBeforeAbort>
BKP_cbRestoreLastBKP	<AutomaticRestore>
BKP_cbIntegrateLastLog	<AutomaticLogIntegration>
BKP_cbStartDbAfterRestore	<AutomaticRestart>
BKP_cbIfModified	<BackupIfDataChange>
BKP_cbKeepLastBKP	<Enable>
BKP_BackupSet	<Value>
BKP_at_CompressionRate	<CompressionRate>
BKP_at_RedundancyRate	<Redundancy>
BKP_at_InterlacingRate	<Interlacing>
BKP_at_SegmentSize	<DefaultSize>
BKP_at_DelOldBKP	<EraseOldBackupBefore>
Bkp_cbStructureFile	<IncludeStructureFile>
Bkp_cbDataFile	<IncludeDataFile>
Bkp_cbAltFile	<IncludeAltStructFile>
BKP_BackupFileDest	<DestinationFolder>
BKP_SA_Attatchments	<ItemsCount>
BKP_SA_Attatchments	<Item>
BKP_rbSched_NoBKP	<Frequency>
BKP_rbSched_Hours	
BKP_rbSched_Days	
BKP_rbSched_Weeks	
BKP_rbSched_Months	
BKP_atSchedStartHours	<Hourly>
BKP_atSchedStartDay	<Daily>
BKP_SchedEveryWeek	<Weekly>
BKP_cbSchedEveryMonday	<Monday>
BKP_atSchedStartTuesday	<Tuesday>
BKP_CBSCHEDEVERYWEDNESDAY	<Wednesday>
BKP_CBSCHEDEVERYTHURSDAY	<Thursday>
BKP_CBSCHEDEVERYFRIDAY	<Friday>
BKP_CBSCHEDEVERYSATURDAY	<Saturday>
BKP_CBSCHEDEVERYSUNDAY	<Sunday>
BKP_SchedEveryMonth	<Monthly>
BKP_atSchedStartMonth	<Hour>
BKP_SchedEveryMonthDay	<Day>

Summary

This Technical Note describes how to prevent the invalid destination dialog from 4D Backup when starting a custom application. It also shows how to implement the definition of your own settings about Backup in your database.