```
    // Kill symbolic path but leave directory on disk.
    KillPath/Z tempPackagePrefsPath
End

// LoadPackagePrefs(packageName)
// Loads the data from the previously-saved package preference file,
// if it exist, into the package's data folder.
// Returns 0 if the preference file existed, -1 if it did not exist.
// In either case, this function creates the package's data folder if it
// does not already exist.
// LoadPackagePrefs does not affect any other data already in the
// package's data folder.
Function LoadPackagePrefs(packageName)
    String packageName              // NOTE: Use a distinctive package name.

    Variable result = -1
    DFREF saveDF = GetDataFolderDFR()

    NewDataFolder/O/S root:Packages      // Ensure root:Packages exists
    NewDataFolder/O/S $packageName       // Ensure package data folder exists

    // Find the disk directory in the Packages directory for this package
    String fullPath = SpecialDirPath("Packages", 0, 0, 0)
    fullPath += packageName
    GetFileFolderInfo/Q/Z fullPath
    if (V_Flag == 0)                     // Disk directory exists?
        fullPath += ":Preferences.pxp"
        GetFileFolderInfo/Q/Z fullPath
        if (V_Flag == 0)                 // Preference file exist?
            LoadData/O/R/Q fullPath      // Load the preference file.
            result = 0
        endif
    endif

    SetDataFolder saveDF
    return result
End
```

The hard part of using the experiment file for saving package preferences is not in saving or loading the package preference data but in choosing when to save and load it so that the latest preferences are always used. There is no ideal solution to this problem but here is one strategy:

1. When package preference data is needed (e.g., you are about to create your control panel and need to know the preferred coordinates), check if it exists in memory. If not load it from disk.
2. When the user does a New Experiment or quits Igor, if package preference data exists in memory, save it to disk. This requires that you create an IgorNewExperimentHook function and an IgorQuitHook function.
3. When the user opens an experiment file, if it contains package preference data, delete it and reload from disk. This requires that you create an AfterFileOpenHook function. This is necessary because the package preference data in the just opened experiment is likely to be older than the data in the package preference file.

Creating Your Own Help File

If you are an advanced user, you can create an Igor help file that extends the Igor help system. This is something you might want to do if you write a set of Igor procedures or extensions for use by your colleagues. If your procedures or extensions are generally useful, you might want to make them available to all Igor users. In either case, you can provide documentation in the form of an Igor help file.

Here are the steps for creating an Igor help file.

1. Create a formatted-text notebook.

   A good way to do this is to open the Igor Help File Template provided by WaveMetrics in the More Help Files folder. Alternatively, you can start by duplicating another WaveMetrics-supplied help file and then open it as a notebook using File→Open File→Notebook. Either way, you are starting with a notebook that contains the rulers used to format an Igor help file.
2. Choose Save Notebook As from the File menu to create a new file. Use a ".ihf" extension so that Igor will recognize it as a help file.