The use of the Igor experiment file supports storing a large amount of preference data but creates a problem of synchronizing your preference data stored in memory and your preference data stored on disk. It also leads to a proliferation of preference data stored in various experiments. You should avoid using this technique if possible.

Saving Package Preferences in a Special-Format Binary File

This approach supports preference data consisting of a collection of numeric and string data. You define a structure encapsulating your package preference data. You use the **LoadPackagePreferences** operation (page V-505) to load your data from disk and the **SavePackagePreferences** operation (page V-825) to save it to disk.

SavePackagePreferences stores data from your package's preferences data structure in memory. LoadPackagePreferences returns that data to you via the same structure.

SavePackagePreferences also creates a directory for your package preferences and stores your data in a file in that directory. Your package directory is located in the Packages directory in Igor's preferences directory. The job of storing the preferences data in the file is handled transparently which, by default, automatically flushes your data to the file when the current experiment is saved or closed and when Igor quits.

You would call LoadPackagePreferences every time you need to access your package preference data and SavePackagePreferences every time you want to change your package preference data. You pass to these operations an instance of a structure that you define.

Here are example functions from the Package Preferences Demo experiment that use the LoadPackagePreferences and SavePackagePreferences operations to implement preferences for a particular package:

```
// NOTE: The package name you choose must be distinctive!
static StrConstant kPackageName = "Acme Data Acquisition"
static StrConstant kPrefsFileName = "PanelPreferences.bin"
static Constant kPrefsVersion = 100
static Constant kPrefsRecordID = 0

Structure AcmeDataAcqPrefs
   uint32version    // Preferences structure version number. 100 means 1.00.
   double panelCoords[4]      // left, top, right, bottom
   uchar phaseLock
   uchar triggerMode
   double ampGain
   uint32 reserved[100]       // Reserved for future use
EndStructure

// DefaultPackagePrefsStruct(prefs)
// Sets prefs structure to default values.
static Function DefaultPackagePrefsStruct(prefs)
   STRUCT AcmeDataAcqPrefs &prefs

   prefs.version = kPrefsVersion

   prefs.panelCoords[0] = 5          // Left
   prefs.panelCoords[1] = 40         // Top
   prefs.panelCoords[2] = 5+190      // Right
   prefs.panelCoords[3] = 40+125     // Bottom
   prefs.phaseLock = 1
   prefs.triggerMode = 1
   prefs.ampGain = 1.0

   Variable i
   for(i=0; i<100; i+=1)
      prefs.reserved[i] = 0
   endfor
End

// SyncPackagePrefsStruct(prefs)
// Syncs package prefs structures to match state of panel.
// Call this only if the panel exists.
static Function SyncPackagePrefsStruct(prefs)
   STRUCT AcmeDataAcqPrefs &prefs

   // Panel does exists. Set prefs to match panel settings.
   prefs.version = kPrefsVersion
```

```
    GetWindow AcmeDataAcqPanel wsize
    // NewPanel uses device coordinates. We therefore need to scale from
    // points (returned by GetWindow) to device units for windows created
    // by NewPanel.
    Variable scale = ScreenResolution / 72
    prefs.panelCoords[0] = V_left * scale
    prefs.panelCoords[1] = V_top * scale
    prefs.panelCoords[2] = V_right * scale
    prefs.panelCoords[3] = V_bottom * scale

    ControlInfo /W=AcmeDataAcqPanel PhaseLock
    prefs.phaseLock = V_Value        // 0=unchecked; 1=checked

    ControlInfo /W=AcmeDataAcqPanel TriggerMode
    prefs.triggerMode = V_Value      // Menu item number starting from on

    ControlInfo /W=AcmeDataAcqPanel AmpGain
    prefs.ampGain = str2num(S_value)  // 1, 2, 5 or 10
End

// InitPackagePrefsStruct(prefs)
// Sets prefs structures to match state of panel or
// to default values if panel does not exist.
static Function InitPackagePrefsStruct(prefs)
    STRUCT AcmeDataAcqPrefs &prefs

    DoWindow AcmeDataAcqPanel
    if (V_flag == 0)
        // Panel does not exist. Set prefs struct to default.
        DefaultPackagePrefsStruct(prefs)
    else
        // Panel does exists. Sync prefs struct to match panel state.
        SyncPackagePrefsStruct(prefs)
    endif
End

static Function LoadPackagePrefs(prefs)
    STRUCT AcmeDataAcqPrefs &prefs

    // This loads preferences from disk if they exist on disk.
    LoadPackagePreferences kPackageName, kPrefsFileName, kPrefsRecordID, prefs

    // If error or prefs not found or not valid, initialize them.
    if (V_flag!=0 || V_bytesRead==0 || prefs.version!=kPrefsVersion)
        InitPackagePrefsStruct(prefs) // Set from panel if it exists or to default values.
        SavePackagePrefs(prefs)       // Create initial prefs record.
    endif
End

static Function SavePackagePrefs(prefs)
    STRUCT AcmeDataAcqPrefs &prefs

    SavePackagePreferences kPackageName, kPrefsFileName, kPrefsRecordID, prefs
End
```

NOTE: The package preferences structure, AcmeDataAcqPrefs in this case, must not use fields of type Variable, String, WAVE, NVAR, SVAR or FUNCREF because these fields refer to data that may not exist when LoadPackagePreferences is called.

The structure can use fields of type char, uchar, int16, uint16, int32, uint32, int64, uint64, float and double as well as fixed-size arrays of these types and substructures with fields of these types.

Use the reserved field to add fields to the structure in a backward-compatible fashion. For example, a subsequent version of the structure might look like this:

```
Structure AcmeDataAcqPrefs
    uint32   // Preferences structure version number. 100 means 1.00.
    double panelCoords[4]       // left, top, right, bottom
    uchar phaseLock
    uchar triggerMode
    double ampGain
```