What is a *Logical Unit of Work*

In a transactional system, there are some sets of work that you want saved either in its entirety, or not at all. For example, you don't want detail lines of an order to be successfully saved if the save of the header row containing order total information fails. You probably don't want a user's permissions deleted if the delete of the user fails. So, a logical unit of work is a set of transactions where either all are successfully applied against the database, or none have any impact on the database.

How does PFC's LUW Service work

Each LUW object keeps a list of objects that make up the logical unit of work. In strict PFC environments, all objects in this list will be self-updating objects. However, the LUW can handle certain non-PFC and non self-updating objects. The LUWs concept of save functionality is broken down into several stages:

- AcceptText
- UpdatesPending
- Validation
- UpdatePrep
- PreUpdate
- BeginTran (LUW oriented not applied to objects)
- Update
- EndTran (LUW oriented not applied to objects)
- PostUpdate

(Details of these stages are described in the table below.)

For each stage, appropriate logic is applied against each object in the list before proceeding with the next stage. If any object fails to pass one of these stages, the entire process is aborted.

For example, all objects will have to pass the AcceptText stage. If one of the objects fails the AcceptText stage, no further AcceptTexts are performed and no further stages are processed.

What are Self-Updating Objects

Simply put, self-updating objects (SUOs) are objects that know how to save (or update) themselves. They all have code that implements the Update stage of the Logical Unit of Work service, and may contain code to implement other stages. To implement functionality for each of the save process stages, the Logical Unit of Work service expects a specific API. (For details on the API, see the table below, looking at the Save Stage and SUO/Target API columns where the Target is SUO controls and containers)

PFC6 comes with several objects with the full SUO API built in. These objects are:

- all windows (descendants of pfc_w_master: including w_child, w_frame, w_main, w_popup, w_response, w_sheet)
- DataWindows (u_dw)
- DataStores (n_ds)
- tabs (u_tab)
- custom visual objects and tabpages (descendants of pfc_u_base: including u_tabpg)
- listviews (u_lv, u_lvs)
- treeviews (u_tv, u_tvs)
- datawindow updating services (n_cst_dwsrv_Linkage)

However, one is not restricted to these objects. Any powerobject that has all or part (but always at least the Update stage) of the SUO API implemented can be included in a logical unit of work. An example would be a SingleLineEdit that updates a field in the INI file. The update stage may be a call to SetProfileString ().

How does the Logical Unit of Work Service define its List of Objects

The LUW service defines a set of objects that it will work against by default. The process usually starts at the window level. The process goes through the control list, querying each object using metaclass functionality to discover which objects implement the self-updating object API. If it finds a SUO, it will add that object, plus it will query the object for its update list. The list of objects that the LUW service operates against can always be programmatically define using of_SetUpdateObjects ().

The SUO API

For an object you create yourself to be self-updating, it must comply with the SUO API if it is to work with the Logical Unit of Work service. For each stage, the object may or may not implement the corresponding function. However, the function for the Update stage is always required. The functions related to each of the object-related stages are:

Stage:	AcceptText
Prototype:	of_AcceptText (ab_FocusOnError) returns integer
Description:	This stage is analogous to its datawindow cousin. It should provide simple data validation such as data type checking, range checking, or checking against simple rules. Data should not be validated against data from other objects, as the simple validation of the other objects may not yet have been performed. Returns -1 for failures, 1 for successes.
Stage:	UpdatesPending

Prototype:	of_UpdatesPending () returns integer
Description:	Returns the number of objects with updates pending if there are updates to be done in this object; returns 0 if there are none.

Stage:	Validation
Prototype:	of_Validation () returns integer
Description:	An additional error checking step. This step may contain cross-object validation. Returns -1 for failures, 1 for successes.

Stage:	UpdatePrep
Prototype:	of_UpdatePrep () returns integer
Description:	Returns -1 for failures, 1 for successes.

Stage:	Update
Prototype:	of_Update (ab_AcceptText, ab_ResetFlag, lpo_UpdateRequestor) returns integer
Description:	**Required!!** This step is analogous to its datawindow cousin also. Returns -1 for failures, 1 for successes.

Stage:	PostUpdate
Prototype:	of_PostUpdate () returns integer
Description:	Returns -1 for failures, 1 for successes.

Save Process Overview

Save Stage:	The stage of the save process as viewed by the Logical Unit of Work service
Target:	The object (or object type) from the list of objects in the logical unit of work.
SUO/Target API:	The function call(s) against the target by the Logical Unit of Work service
Application Level Logic Location:	(SUOs only) The place defined by the standard PFC SUOs to implement application specific code for the given stage
Default Functionality:	The behaviour that the standard PFC DataWindows and DataStores implement by default in the Application Level Logic Location

Save Stage	Target	SUO/Target API	Application Level Logic Location	Default Functionality for DataWindows and DataStores
AcceptText	All SUO controls and containers	`of_AcceptText (ab_FocusOnError)`	Event pfc_AcceptText	`AcceptText()`
	All other containers	`of_AcceptText ( luo_Control.Control, ab_FocusOnError )`		

	All other DataWindows and DataStores	`If ldw_nonpfc.RowCount() + ldw_nonpfc.FilteredCount() + ldw_nonpfc.ModifiedCount() + ldw_nonpfc.DeletedCount() > 0 Then`
		
		`la_rc = ldw_NonPFC.AcceptText()`		

UpdatesPending	All SUO controls and containers	`of_UpdatesPending ()`	**DWs & DSs:** Event pfc_UpdatesPending **Windows:** Event pfc_UpdatesPendingRef	`IF of_SetUpdateable ()=TRUE AND ((ModifiedCount()+DeletedCount( )) > 0)`
	All other containers	`of_UpdatesPending ( lw_control.control )`		

	All other DataWindows and DataStores	`ls_Updatetable = ldw_nonpfc.Describe("DataWindow.Table.UpdateTable")`
		
		`If ls_updatetable = '?' or ls_updatetable = '' Then`
		
		`lb_updatespending = False`
		
		`Else`
		
		`lb_updatespending = (ldw_nonpfc.ModifiedCount() + ldw_nonpfc.DeletedCount() >= 1)`
		
		`End If`		

Validation	All SUO controls and containers with pending updates *or* all SUO controls and containers	`of_Validation ()`	Event pfc_Validation	`FindRequired ()`

Save Stage	Target	SUO/Target API	Application Level Logic Location	Default Functionality for DataWindows and DataStores
	All other [pending] SUO containers	`of_validation (lw_control.control)`		
	All other [pending] DataWindows and DataStores	`ue_validation ()`		
Exit the process if no updates pending				
UpdatePrep	Pending SUO controls and containers	`of_UpdatePrep ()`	Event pfc_UpdatePrep	None
	Other pending containers	`of_UpdatePrep ( lw_control.control )`		
	Other pending DataWindows and DataStores	`ue_updateprep ()`		
PreUpdate			Local to LUW service	
BeginTran	Transaction or transaction array	`atr_control[li_i].of_Begin()`	Local to LUW service	
Update	Pending SUO controls and containers	`of_Update (ab_AcceptText, ab_ResetFlag, lpo_UpdateRequestor)`	Event pfc_Update	`Update (TRUE, FALSE)`
	Other pending containers	`of_Update ( lw_control.control, ab_accepttext, ab_resetflag )`		
	Other pending DataWindows and DataStores	`Update (ab_accepttext, ab_resetflag)`		

Save Stage	Target	SUO/Target API	Application Level Logic Location	Default Functionality for DataWindows and DataStores

EndTran	Transaction or transaction array	`If ai_saverc > 0 Then`
		`  If atr_control[li_i].of_Commit() < 0 Then...`
		`Else`
		`  If atr_control[li_i].of_Rollback() < 0 Then...`
			Local to LUW service	

If an error occurred during the save, report the DBError and exit				
PostUpdate	Pending SUO controls and containers	`of_PostUpdate ()`	Event pfc_PostUpdate	`ResetUpdate()`
	Other pending containers	`of_PostUpdate ( lw_control.control )`		
	Other pending DataWindows and DataStores	`ResetUpdate()`		

1. Shows default functionality for DataWindows and DataStores. Containers cascade the API calls in all cases by default. Descriptions exclude functionality conditional on the Linkage or Multitable Services
2. Unless a filter has been defined through of_SetTypeToProcess (string as_ObjectType) (e.g. of_SetTypeToProcess (datawindow))
3. Windows, tabs and custom visual user objects. (Note: tabpages are viewed as custom visual user objects to PowerBuilder internally.)
4. Dependent on of_SetAlwaysValidate(boolean), which toggles all controls validated whether updates pending or not

LUW Service Developers Guide

Example task 1

Tabpage with DataWindows. No DataStores. Order of update unimportant. No save time validations.

Tabpage Constructor

```
// Instantiate the LUW service
of_SetLogicalUnitOfWork (TRUE)
```

Example task 2

Tabpage with an update that requires datawindow updates, datastore updates and embedded SQL. Save time validations are also required.

Tabpage Constructor

```
window lw_Parent

// Create datastore with either:
ids_ResponseContents = CREATE n_ds // do not CREATE datastore
ids_ResponseContents.DataObject = d_ResponseContents
// OR
ids_ResponseContents = CREATE ds_ResponseContents // where ds_ResponseContents is a descendent
of n_ds


// Instantiate the LUW service
of_SetLogicalUnitOfWork (TRUE)

// Define the order of update and/or include objects other than DataWindows
of_SetUpdateObjects ([dw_UpdateMeFirst, dw_Second, ids_ResponseContents])

// Associate DataStores with a window (required for some LUW functions)
of_GetParentWindow (lw_Parent)
ids_ResponseContents.of_SetParentWindow (lw_Parent)
```

Tabpage pfc_Update (ab_AcceptText, ab_ResetFlag) Event

override ancestor event

```
integer li_Return


// Allow ancestor event to process objects in update array and capture return value / success
indicator
li_Return = Super::Event pfc_Update (ab_AcceptText, ab_ResetFlag)

// If ancestor was successful, run embedded SQL
IF li_Return = 1 THEN
        UPDATE ... ;
        IF SQLCA.of_SQLError () <> 0 THEN
                li_Return = -1
                ...
        END IF
END IF
```

```
RETURN li_Return
```

Save time validations

Save time validations are coded in:

- datawindow controls pfc_Validation events
- datastore objects pfc_Validation events
- tabpage pfc_Validation events (e.g. cross datawindow validation)

Example task 3

Tabpage with two DataWindows. One is for display purposes only. The other should be updated to the database at save time.

Tabpage Constructor

```
// Instantiate the LUW service
of_SetLogicalUnitOfWork (TRUE)
```

dw_DisplayOnly Constructor

```
// Flag datawindow to be excluded from LUW save process
of_SetUpdateable (FALSE)
```