XFS

Prática: mensagens

O SP deve sempre notificação a conclusão da execução dos comandos WFP, a não ser que o comando tenha terminado com um erro do tipo WFS_ERR_INTERNAL_ERROR.

Essa notificação é feito por meio da publicação de uma mensagem na window que o XFS Manager passa para o SP nas chamadas das funções WFP. As mensagens podem ser do tipo “COMPLETE”, que indica que um comando foi finalizado, mas também pode ser do tipo “EVENT” para indicar que algo inesperado ocorreu. Essas mensagens têm ID definições assim no XFS:

- WFS_OPEN_COMPLETE
- WFS_CLOSE_COMPLETE
- WFS_LOCK_COMPLETE
- WFS_UNLOCK_COMPLETE
- WFS_REGISTER_COMPLETE
- WFS_DEREGISTER_COMPLETE
- WFS_GETINFO_COMPLETE
- WFS_EXECUTE_COMPLETE
- WFS_EXECUTE_EVENT
- WFS_SERVICE_EVENT
- WFS_USER_EVENT
- WFS_SYSTEM_EVENT

Para enviar essas mensagens, a função PostMessage do Win32API deve ser utilizada. O parâmetro WPARAM não é utilizado e o parâmetro LPARAM deve conter o endereço de memória para o objeto WFSRESULT.

XFS

Prática: mecanismo de notificação - eventos

O SP deve sempre gerar e notificar eventos de acordo como a especificação XFS. Sempre que esses eventos ocorrem, o SP deve notifica-los ao XFS Manager. Além do XFS Manager, as aplicações com sessão aberta junto ao SP, podem também se registrar para receber notificações da ocorrência desses eventos. Esse registro é feito com chamada as funções **WFSRegister** / **WFSAsyncRegister**, e podem ser desfeitos com as chamadas **WFSDeregister** / **WFSAsyncDeregister**. As classes de eventos e seus significados são as seguintes:

- **SERVICE_EVENTS**: o status do serviço XFS mudou. Exemplo: a impressora ficou indisponível;
- **USER_EVENTS**: o serviço XFS precisa de ação do usuário. Exemplo: abastecer de papel a impressora;
- **SYSTEM_EVENTS**: um evento de Sistema ocorreu. Exemplo: ocorreu erro de hardware, acabou espaço em disco;
- **EXECUTE_EVENTS**: esse tipo é diferente. Ele ocorre como parte da execução do comando **WFSExecute** e são enviados sempre antes do fim da execução do comando. Exemplo: necessidade de inserção do cartão na leitora para concluir a execução do comando.

As três primeiras são transmitidas para todas as janelas (window) registradas junto ao SP. A ultima (EXECUTE_EVENTS) devem ser transmitida apenas para a aplicação que executou o comando **WFSExecute**. Há também uma exceção na classe de evento **SYSTEM_EVENTS**, que é o evento **WFS_SYSE_LOCK_REQUESTED**. Esse evento, embora seja do tipo “SYSTEM”, deve ser enviado apenas para a aplicação que detém o “lock” atual do serviço.

Na documentação XFS, parte 1, tem a definição e explicação de todos os eventos comuns ao sistema XFS, ou seja, eventos do tipo “SYSTEM”. Dentro da documentação de cada classe XFS se encontra eventos dos demais tipos, seu significado e uso. Isso é encontrado no capítulo “Events”.

XFS

Prática: WFSRegister

Habilita a monitoração de eventos de um SP para uma janela da aplicação.

Parâmetros:

- **hService**: handle do SP. Se for NULL e dwEventClass = SYSTEM_EVENTS, então o XFS Manager envia seus próprios eventos SYSTEM_EVENTS para a aplicação.
- **dwEventClass**: a classe de eventos. Pode ser usado como um conjunto de máscara de bit (operador |).
- **hWndReg**: handle da janela onde a notificação deve ser postada.

O comando é cumulativo, ou seja, pode ser chamado mais de uma vez. Exemplos de chamada:

Exemplo 1 (habilitado duas classes de evento na mesma chamada):

```
hr = WFSRegister(hPassbook1, SYSTEM_EVENTS | USER_EVENTS, hWndReg1);
```

Exemplo 2 (habilitando duas classes de evento em duas chamadas):

```
hr = WFSRegister( hPassbook1, SYSTEM_EVENTS, hWndReg1);
.
.
.
hr = WFSRegister( hPassbook1, USER_EVENTS, hWndReg1);
```

XFS

Prática: habilitando notificação de eventos App

Para conseguirmos usar a notificação de eventos XFS do lado da App cliente, são necessários:

- Criar uma janela de notificações;
- Passar o endereço dessa janela de notificações para o XFS Manager (WFSRegister);
- Processar os eventos XFS que chegam;

Para estabelecer uma janela de notificações, usar:

- CreateThread: cria uma thread para o loop de mensagens;
- Criar uma Procedure CALLBACK;
- CreateWindow: cria uma janela e aponta a Procedure CALLBACK;

Fazer a chamada do WFSRegister:

- WFSRegister(hService, SYSTEM_EVENTS | USER_EVENTS | SERVICE_EVENTS | EXECUTE_EVENTS, messageWindow);

Vide exemplo:

- boolean CreateXFSMonitor();
- boolean RegisterCallback();
- LRESULT CALLBACK PostCallBack(HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam);
- DWORD FAR PASCAL ThreadMonitor(string name);

XFS

Prática: WFPRegister

Habilita a monitoração de eventos de um SP para uma janela de notificação.

Parâmetros:

- **hService**: handle do SP.
- **dwEventClass**: a classe de eventos. Pode ser usado como um conjunto de máscara de bit (operador |).
- **hWndReg**: handle da janela onde a notificação deve ser postada.
- **hWnd**: handle da janela para postar a notificação de conclusão do comando WFPRegister. A mensagem que tem que ser postada aqui é a WFS_REGISTER_COMPLETE ;
- **ReqID**: identificação única da requisição;

XFS

Prática: habilitando notificação de eventos SP

Pelo lado da App a notificação de eventos já é feita. O que precisa fazer em adicional é preparar o SP para notificar os eventos registrados para as janelas de notificação adicionais que foram registradas. A inclusão de novas janelas de notificação ocorre por meio do WFPRegister.

O que precisa para implementar o WFPRegister?

- Criar uma lista de HWND; Por enquanto iremos apenas nos concentrar em guardar essas HWND em lista, sem aplicar restrições (dwEventClass) ao qual ela está associada;
- Adicionar a HWND na lista (ignorar registro duplicados por enquanto)

Vide exemplo:

- Variável global:
  - vector<HWND>janelas = {};
- WFPRegister:
  - janelas.push_back(hWndReg);
- WFPExecute (ou qualquer outro, é apenas um exercício):
  - for (HWND it : janelas)
    - PostMessage(it, WFS_SYSTEM_EVENT, 0, (LPARAM)lpPostRes);