RÉSUMÉ DE LA SÉRIALISATION (par M.M. sept. 99)

1. Pour pouvoir faire la sérialisation facilement, il faut d'abord conserver les données dans le document. Dans la fonction `Serialize`, on peut alors écrire ou lire ces données à partir du paramètre `ar` (de type `CArchive`), un peu comme on le faisait avec les streams. Il est important d'écrire toutes les informations qui nous permettront de relire facilement les données. Par exemple, on écrira d'abord combien de données seront écrites, puis on écrira chacun des éléments de ces données.

   De façon à faire une validation simple du fichier de données, on y écrira d'abord une signature. En lecture, on essaiera d'abord de lire celle-ci et on la vérifiera : si elle n'est pas correcte, on lèvera une exception avec `AfxThrowArchiveException(CArchiveException::badIndex);` ce qui fera afficher automatiquement un message adéquat. Cette exception similaire est levée automatiquement si les lectures dans `ar` ne fonctionnent pas comme prévues (mauvais format, etc.).

   Voici un exemple qui suppose qu'on a conservé les données des types `TypeClient` et `TypeFacture` dans les vecteurs `m_clients` et `m_factures` :

```
void CFacturationDoc::Serialize(CArchive& ar)
    {
    const CString signatureOfficielle("Facturation version 1.0");

    if (ar.IsStoring())
        {
        ar << signatureOfficielle;
        ar << m_clients.size();

        for (int i= 0; i < m_clients.size(); ++i)
            {
            ar << m_clients[i].numero << m_clients[i].nom << m_clients[i].prenom
               << ... // etc. pour les autres champs
            }

        ar << m_factures.size();

        for (int i= 0; i < m_factures.size(); ++i)
            {
            ar << m_factures[i].noSerie << m_factures[i].noClient
               << ... // etc. pour les autres champs
            }
        }
    else
        {
        CString signature;
        ar >> signature;

        if (signature != signatureOfficielle)
            AfxThrowArchiveException(CArchiveException::badIndex);

        int nbClients;
        ar >> nbClients;
        m_clients.resize(nbClients);

        for (int i= 0; i < m_clients.size(); ++i)
            {
            ar >> m_clients[i].numero >> m_clients[i].nom >> m_clients[i].prenom
               >> ... // etc. pour les autres champs
            }

        int nbFactures;                 // Dans certains cas, une série de push_back (avec reserve
        ar >> nbFactures;               // dans les vecteurs) est préférable à un resize...
        m_factures.reserve(nbFactures);

        for (int i= 0; i < nbFactures; ++i)
            {
            TypeFacture facture;
            ar >> facture.noSerie >> facture.noClient
               >> ... // etc. pour les autres champs
            m_factures.push_back(facture);
            }
        }
    }
```

   On peut aussi faire une fonction membre `Archiver` dans les types de données qu'on crée, ce qui permettrait, par exemple, de faire simplement `m_clients[i].Archiver(ar);` dans la boucle `for`… Par exemple :

```
void TypeClient::Archiver(CArchive& p_ar)
    {
    if (p_ar.IsStoring())
        p_ar << numero << nom << prenom << // etc.
    else
        p_ar >> numero >> nom >> prenom << // etc.
    }
```

2. On doit ensuite surcharger le fonction `DeleteContents` de notre document, particulièrement si on fait des applications SDI, car l'objet de type document est réutilisé lorsqu'on charge un fichier du disque ou qu'on demande *Fichier-Nouveau*. Cette fonction est alors appelée et elle doit détruire les données du document, c'est-à-dire réinitialiser celui-ci. Par exemple :

```
void CFacturationDoc::DeleteContents()
    {
    m_clients.clear();
    m_factures.clear();
    // etc.
    CDocument::DeleteContents();  // N.B. Ne fait rien, mais on le laisse par principe
    }
```

3. Si on veut fournir certaines valeurs par défaut lors de *Fichier-Nouveau*, on peut mettre ces opérations dans `OnNewDocument` :

```
BOOL CLibrairieDoc::OnNewDocument()
    {
    if (!CDocument::OnNewDocument())   // Ça appelle DeleteContents...
        return FALSE;

    // TODO: add reinitialization code here
    // (SDI documents will reuse this document)

    // À ce point, le document est vide (à cause de DeleteContents), on peut y mettre
    // quelque chose si c'est ce qu'on veut lors de Fichier-Nouveau (le programme ne
    // passe pas par ici si on utilise Fichier-Ouvrir ou autre).

    m_factures.resize(1);         // N.B. POUR L'EXEMPLE SEULEMENT, N'EST PAS
    m_factures[0].noSerie= 1000;  //      VRAIMENT TRÈS LOGIQUE…

    return TRUE;
    }
```

4. Lorsque des opérations modifient les données du document, on veillera à appeler la fonction membre `SetModifiedFlag()` afin qu'un message apparaisse automatiquement si on tente de quitter le programme ou de charger un nouveau fichier avant d'avoir sauvegarder les modifications.

5. Dans la vue, il faudra surcharger la fonction `OnInitialUpdate` et probablement `OnUpdate`. La première, dans laquelle on initialise d'abord habituellement un pointeur sur le document (`m_pDoc`), est appelée lors de la création d'un «nouveau» document après *Nouveau*, *Ouvrir*, etc. On peut donc y appeler aussi des fonctions du document pour connaître certaines valeurs, qu'on voudrait faire afficher par exemple.

   Cependant, plus généralement, on fera ces opérations dans `OnUpdate`, qui est appelée une première fois lors du `OnInitialUpdate` de la classe de base qu'on aura laissé dans notre propre fonction `OnInitialUpdate`. Ensuite, `OnUpdate` sera aussi rappelée si le document appelle la fonction membre `UpdateAllViews(0)`; on peut alors utiliser le principe suivant : la vue demande des opérations au document, puis celui-ci, au besoin, avertit la vue qu'elle doit se mettre à jour (en faisant appeler `OnUpdate`).

   L'autre alternative est que la vue se mette manuellement à jour chaque fois qu'elle demande une opération au document. Dans ce cas cependant, certaines opérations sans impact sur le document pourraient obliger la vue à se rafraîchir inutilement. On peut souvent utiliser judicieusement une combinaison des deux méthodes : dans ce cas, on fait une fonction membre genre `Rafraichir()` dans notre vue et on l'appelle directement, au besoin, et aussi à partir de `OnUpdate`.

   La technique du `UpdateAllViews` et `OnUpdate` sera essentielle lorsqu'on utilisera plusieurs vues d'un même document…