complete source code of the serializer plug-in for contacts, or KABC::Addressee objects as the KDE library calls them:

```
bool SerializerPluginAddressee::deserialize( Item& item,
                                             const QByteArray& label,
                                             QIODevice& data,
                                             int version )
{
    if ( label != Item::FullPayload || version != 1 )
        return false;

    KABC::Addressee a = m_converter.parseVCard( data.readAll() );
    if ( !a.isEmpty() ) {
        item.setPayload<KABC::Addressee>( a );
    } else {
        kWarning() << "Empty addressee object!";
    }
    return true;
}

void SerializerPluginAddressee::serialize( const Item& item,
                                           const QByteArray& label,
                                           QIODevice& data,
                                           int &version )
{

    if ( label != Item::FullPayload
      || !item.hasPayload<KABC::Addressee>() )
        return;
    const KABC::Addressee a = item.payload<KABC::Addressee>();
    data.write( m_converter.createVCard( a ) );
    version = 1;
}
```

The typed payload, setPayload, and hasPayload methods of the Item class allow developers to use the native types of their data type libraries directly and easily. Interactions with the store are generally expressed as jobs, an application of the command pattern. These jobs track the lifetime of an operation, provide a cancelation point and access to error contexts, and allow progress to be tracked. The Monitor class allows a client to watch for changes to the store in the scope it is interested in, such as per mime type or per collection, or even only for certain parts of particular items. The following example from an email notification applet illustrates these concepts. In this case the payload type is a polymorphic one, encapsulated in a shared pointer:

```
Monitor *monitor = new Monitor( this );
monitor->setMimeTypeMonitored( "message/rfc822" );
monitor->itemFetchScope().fetchPayloadPart( MessagePart::Envelope );
connect( monitor, SIGNAL(itemAdded(Akonadi::Item,Akonadi::Collection)),
         SLOT(itemAdded(Akonadi::Item)) );
connect( monitor, SIGNAL(itemChanged(Akonadi::Item,QSet<QByteArray>)),
         SLOT(itemChanged(Akonadi::Item)) );

// start an initial message download for the first message to show
ItemFetchJob *fetch = new ItemFetchJob( Collection( myCollection ), this );
```


Download at WoweBook.Com