A task executing an `async` computation reacts in case of an *exception* or a *cancellation* by calling the corresponding *continuation* (determined e.g. by a `try`...`with` construct, cf. Section 3.10). A cancellation is requested (from outside the task) by setting the *cancellation token* of the execution of the computation (see example below). The cancellation token is polled regularly by the asynchronous library functions and by the member functions of the `async` computation expression. The cancellation is performed with a proper clean-up of resources as soon as the cancellation request has been discovered.

Using the library function `Async.StartWithContinuations` you may supply your own continuations when an asynchronous computation is started. This function requires three continuations among its parameters:

- Normal continuation *okCon* – invoked after normal termination.
- Exception continuation *exnCon* – invoked if an exception is raised.
- Cancellation continuation *canCon* – invoked if the computation is cancelled.

The following examples execute the computation `downloadComp` (cf. Page 318) with the continuations:

```
let okCon (s: string) = printf "Length = %d\n" (s.Length)
let exnCon _ = printf "Exception raised\n"
let canCon _ = printf "Operation cancelled\n"
```

Such an execution may terminate normally:

```
Async.StartWithContinuations
    (downloadComp "http://www.microsoft.com",
     okCon, exnCon, canCon);;
val it : unit = ()
Length = 1020
```

it may be terminated by an exception:

```
Async.StartWithContinuations
    (downloadComp "ppp",
     okCon, exnCon, canCon);;
val it : unit = ()
Exception raised
```

or it may be cancelled:

```
open System.Threading;; // CancellationTokenSource
let ts = new CancellationTokenSource()

Async.StartWithContinuations
    (downloadComp "http://www.dtu.dk",
     okCon, exnCon, canCon, ts.Token);;
val it : unit = ()

ts.Cancel();;
Operation cancelled
```