TYPE **Iterator**
ABSTRACT
Base class for coroutines used as iterators (sometimes also called generators).
The intended way of using an Iterator (*it*) is that its Run routine sets *it*'s extended internal state into a succession of configurations (representing the designed iterations), and after each configuration is set it calls *it.Yield* which transfers control back to its client.
The client may then use each configuration as desired, then will typically call *it.Next* again.
After the final configuration is reached (assuming that there are only a finite number available) the Run routine returns.

Note that *Coroutines.Transfer* is never called explicitly in either the client code, or the implementation of *it.Run*.

A typical design pattern for using an iterator is:

```
VAR
    it: Iterator;
BEGIN
    NEW(it);
    ... Set it's extended internal state to its initial configuration ...

    LOOP
        it.Next
        IF it.state = Coroutines.returned THEN EXIT END;
        ASSERT(it.state = Coroutines.suspended, 30)
        ... performed required processing with each configuration of it's internal state ...
    END;
END Iterate;
```

PROCEDURE (this: Iterator) **Next**
NEW
...

Post
current.source = this 80
this.state # trapped 81
this.state = suspended The next configuration is available
this.state = returned No further configurations are available