11 Klassenbasierte Eigenentwicklungen im Workflow

11.1 Utility-Klasse ZCL_DATE: Datumsberechnung auf Kalendern

11.1.1 Verwendung von Datumsberechnungen

Datums- und Zeitberechnung tritt an vielen Stellen im Workflow auf. Typischerweise wird sie bei der Ermittlung von Vorlage und Erledigungsfristen verwendet. Kompliziert wird es erst, wenn die Datumsberechnung auf einem Fabrikkalender ausgeführt werden soll, wenn also Wochenenden und Feiertage ausgeblendet werden sollen. Das ist häufig bei der Angabe von Fristen oder Deadlines der Fall.

11.1.2 Implementierung von ZCL_DATE

Die folgende Utility-Klasse ZCL_DATE besitzt die beiden Returningmethoden DATE() und TIME(). DATE() ermittelt ausgehend von einem Startdatum und einer Startzeit ein Ergebnisdatum durch Addition von DUR Zeiteinheiten (UNIT). Dabei kann ein Fabrikkalender angegeben werden. Außer der Dauer sind alle Parameter optional und mit Defaultwerten versehen. Gibt man nur DUR=3 an, so wird z. B. ein Datum ermittelt, das bezogen auf heute 3 Werktage auf dem Defaultfabrikkalender »01« in der Zukunft liegt.

```
************************************************************
* Parameters:
* STARTD   Importing    TYPE  SY-DATUM  Aktuelles Datum
* STARTT   Importing    TYPE  SY-UZEIT  Aktuelle Uhrzeit
* DUR      Importing    TYPE  FLOAT     Dauer
* UNIT     Importing    TYPE  MSEHI     Maßeinheit
* CAL      Importing    TYPE  WFCID     Fabrikkalender
* RV_DATE  Returning    TYPE  SYDATUM   Aktuelles Datum
************************************************************
```

Listing 11–1
Klasse ZCL_DATE, Methode DATE

```
METHOD DATE.
    DATA:
    lv_start_date TYPE sy-datum,
    lv_start_time TYPE sy-uzeit,
    lv_end_date   TYPE sy-datum,
    lv_end_time   TYPE sy-uzeit.

  lv_start_date = startd.
  lv_start_time = startt.

  CALL FUNCTION 'END_TIME_DETERMINE'
   EXPORTING
     duration                          = dur
     unit                              = unit
     FACTORY_CALENDAR                  = cal
   IMPORTING
     end_date                          = lv_end_date
     end_time                          = lv_end_time
   CHANGING
     start_date                        = lv_start_date
     start_time                        = lv_start_time
   EXCEPTIONS
     factory_calendar_not_found        = 1
     date_out_of_calendar_range        = 2
     date_not_valid                    = 3
     unit_conversion_error             = 4
     si_unit_missing                   = 5
     parameters_no_valid               = 6
     OTHERS                            = 7
          .
  IF sy-subrc = 0.
    rv_date = lv_end_date.
  else.
* MESSAGE ID SY-MSGID TYPE SY-MSGTY NUMBER SY-MSGNO
*         WITH SY-MSGV1 SY-MSGV2 SY-MSGV3 SY-MSGV4.
  ENDIF.
ENDMETHOD.
```

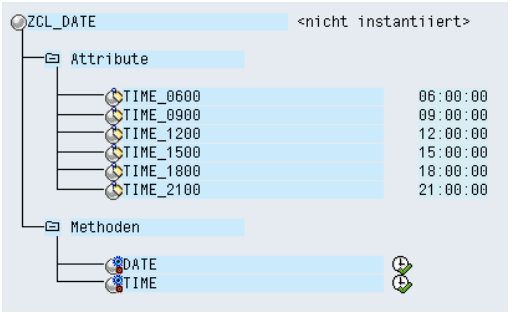
Die Methode TIME() funktioniert ganz ähnlich, liefert nur die Zeit zurück. Man beachte allerdings, dass die Zeitberechnung ebenfalls ein Startdatum und nicht nur eine Startzeit benötigt!

```
************************************************************
* Parameters:
* STARTD   Importing    TYPE  SY-DATUM  Aktuelles Datum
* STARTT   Importing    TYPE  SY-UZEIT  Aktuelle Uhrzeit
* DUR      Importing    TYPE  FLOAT     Dauer
* UNIT     Importing    TYPE  MSEHI     Maßeinheit
* CAL      Importing    TYPE  WFCID     Fabrikkalender
* RV_TIME  Returning    TYPE  SYUZEIT   Uhrzeit
************************************************************
METHOD TIME.
    DATA:
    lv_start_date TYPE sy-datum,
    lv_start_time TYPE sy-uzeit,
    lv_end_date   TYPE sy-datum,
    lv_end_time   TYPE sy-uzeit.

  lv_start_date = startd.
  lv_start_time = startt.

  CALL FUNCTION 'END_TIME_DETERMINE'
   EXPORTING
     duration                          = dur
     unit                              = unit
     FACTORY_CALENDAR                  = cal
   IMPORTING
     end_date                          = lv_end_date
     end_time                          = lv_end_time
   CHANGING
     start_date                        = lv_start_date
     start_time                        = lv_start_time
   EXCEPTIONS
     factory_calendar_not_found        = 1
     date_out_of_calendar_range        = 2
     date_not_valid                    = 3
     unit_conversion_error             = 4
     si_unit_missing                   = 5
     parameters_no_valid               = 6
     OTHERS                            = 7
          .
  IF sy-subrc = 0.
    rv_time = lv_end_time.
  else.
* MESSAGE ID SY-MSGID TYPE SY-MSGTY NUMBER SY-MSGNO
*         WITH SY-MSGV1 SY-MSGV2 SY-MSGV3 SY-MSGV4.
  ENDIF.
ENDMETHOD.
```

Listing 11–2
Klasse ZCL_DATE, Methode TIME

Um in den Fristen die Zeiten auf feste Werte eines berechneten Tages zu setzen (z.B. auf morgens 09:00), wurden entsprechende Read-only-Attribute in der Klasse angelegt.

Abb. 11–1
Zeitkonstanten der Klasse ZCL_DATE

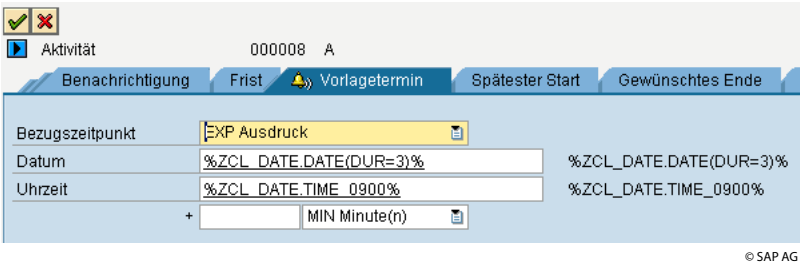


11.1.3 Verwendung von ZCL_DATE in Ausdrücken

Die Methoden DATE und TIME können, wie im vorhergehenden Kapitel bereits dargestellt, direkt in Zuweisungsausdrücken oder in Ausdrücken bei der Berechnung von Terminen (Vorlage, Frist) benutzt werden. Die optionalen Parameter können dabei weggelassen werden. Die Zeit bezieht sich immer auf den Tag, der mit DATE ermittelt wurde. Das folgende Beispiel zeigt einen Vorlagetermin mit folgenden Eigenschaften:

- Datum = SY-DATUM (Default) + 3 TAG(Default) auf Fabrikkalender »01« (Default)
- Zeit = Morgens 09:00 am berechneten Tag

Abb. 11–2
Verwendung von ZCL_DATE im Vorlagetermin



In der Abbildung 11–3 ermittelt ein Datenfluss das Containerelement &KeyDate& durch Rückrechnung von einem anderen Containerelement &EndDate&. Dieses wird als Parameter mitgegeben (STARTD = &EndDate&). Die Dauer ist negativ (DUR = -3), es wird ein von der Defaulteinstellung abweichender Fabrikkalender (CAL = »ZU«) benutzt.

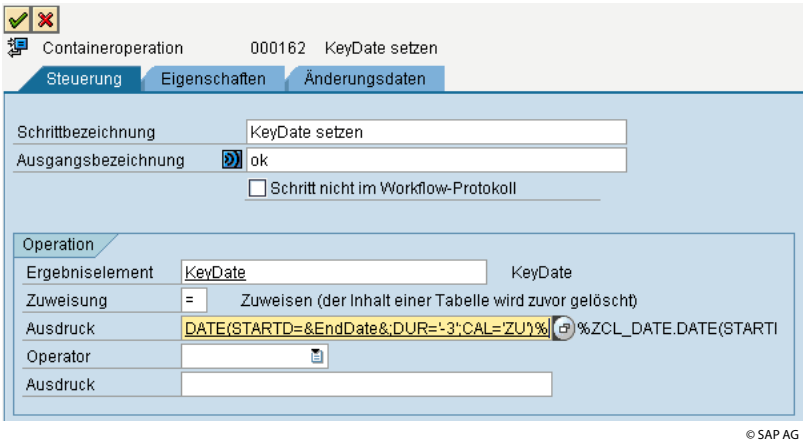


Abb. 11–3
Datumsberechnung in die Vergangenheit

Es ist erstaunlich, wie leistungsfähig diese Verwendung von statischen Returningmethoden ist. Beim Einsatz von BOR-Methoden hätte man hier extra einen Schritt einfügen müssen.

Klassenausdrücke mit % ohne IF_WORKFLOW

Verwendet man ZCL_DATE wie oben dargestellt nur in %-Ausdrücken, so ist das Interface IF_WORKFLOW nicht notwendig. Wenn man es doch implementiert, ggf. mit der im letzten Kapitel erwähnten Pseudoinstanziierung, dann kann man die Ausdrücke bequem über die F4-Hilfe vorschlagen lassen.

11.2 Utility-Klasse ZCL_COND: komplexe Bedingungen

11.2.1 Konzept komplexer Bedingungen im Workflow

Bereits die Modellierung einer einfachen *Überkreuzbedingung* zwischen Parallelzweigen (Abschnitt 10.7.2) mithilfe von lokalem Ereignis und Ereignisempfänger war sehr unhandlich. Noch komplexere Bedingungen würden entsprechend zu noch aufwendigerer Modellierung führen.

Abbildung beliebiger Bedingungen

Es fragt sich daher, ob man nicht eine allgemeine Methode zur Abbildung solcher Bedingungen finden kann, die mit der Modellierung über Vorgänger-Nachfolger-Beziehungen nicht erfassbar sind.

Eine solche Methode wird hier vorgestellt. Sie lässt sich als eine Art Meilensteinmethode verstehen, mit deren Hilfe wichtige Zustände in der Objektbearbeitung erfasst und in Workitem-Startbedingungen geprüft werden können. Folgende Bestandteile werden vorgestellt:

- Tabelle ZCONDITION, in der die für eine Objektinstanz erfüllten Bedingungen stehen.
- Eine Utility-Klasse ZCL_COND (ohne IF_WORKFLOW), die die statischen Methoden SET und CHECK für den Tabellenzugriff implementiert.