HR Query Generator

Defining Switches

Version: July 22, 2002

Table of Contents

1 Introduction

Using the tools Ad Hoc Query and SAP Query, you are able to dynamically define reports (so called Queries) and evaluate any data that is stored in infotypes. If a query is executed, data is read from the database according to the specified select-options, if used, additional fields are calculated and finally all data is printed in an output list. To implement this process, complex ABAP coding is required. Due to the flexibility of the query tools this coding cannot exist statically in the system but has to be generated at runtime of the query. This task – the generation of the coding – is done by the *query generator*.

The generated coding determines, which data is read from the database, which additional fields are calculated at which point of time and how the presentation of data from those infotypes being involed in the query will look like. Due to the fact, that infotypes might have different time constraints, there might be different demands. E.g. an infotype with time constraint 3 (there is an arbitrary set of data records with arbitrary begin- and ending dates) has to be handled differently than an infotype with time constraint 1 (for each point of time exactly one data record exists).

Therefore in the process of generating the coding, the specification of each infotype is readout and appropriate coding is generated according to the settings of the infotype. This defines a standard behavior that in most cases gives a reasonable result. But it is likely, that in some cases an infotype is handled in a way that is not favored by the user. In such cases the user is enabled to explicitly influence the process of generating coding in a way, that his needs are fulfilled.

This influence on the process of generating coding is realized via the usage of standard switches when defining an InfoSet. The following chapters will start with an explanation of the technique to define theses switches and an overview of existing switches. For a better understanding of the influences of these switches, a sketch of the generated coding is presented. Finally a detailed description of each switch and its influence on the process of the generated coding is given.

2 Defining Switches

2.1 General Technique

Defining Switches is done within the InfoSet maintenance tool (transaction SQ02). Switches are part of the InfoSet and therefore valid for all Queries, that base upon this InfoSet. Due to the fact, that manipulation or introduction of a switch might have inpact on all existing Queries, that base upon the InfoSet, any changes on switches should be done very carefully.

The definition of switches is located within the same dialog, where Coding has to be stored, that should be executed at Event DATA. You can use function ‘Goto’ -> ‘Code’ -> ‘Data’ to access this dialog. All switches are defined using a dedicated syntax:

```
NAME_OF_SWITCH = 'value'
```

Whereas NAME_OF_SWITCH has to be replaced be the name of the switch and the value has to be specified using single quotation marks. Furthermore, each line containing the definition of a switch has to start with the prefix *HR. For example the proper definition of the switch LAST_REORD_ONLY with value ‘X’ looks like:

```
*$HR$ LAST_RECORD_ONLY = 'X'
```

There exists two different types of switches: switches that are defined for a single infotype and switches that are valid for the whole InfoSet. To distinguish these types of switches, an ID has to be defined in front of the switch definition. For switches being valid for the whole InfoSet this ID is [COMMON]. For switches that are defined for a single infotype, the ID is the name of the infotype within square brackets: [Pnnnn] where nnnn has to be replaced by the infotype number. E.g. the correct and complete definition of switch LAST_RECORD_ONLY with value ‘X’ for infotype 0001 is:

```
*$HR$ [P0001]
*$HR$ LAST_RECORD_ONLY = 'X'
```

Following the ID [COMMON] or [Pnnnn] an arbitrary number of switches might be specified. Each switch definition always refers to the last specified ID. Besides the possibility of specifying a switch for a single infotype, it is as well possible to define a switch for a set of infotypes at the same time. The following notations are supported:

Specification of several infotypes

```
*$HR$ [P0000, P0004, P0006]
```

Specification of a range

```
*$HR$ [P0006 – P0009]
```

Use of templates/placeholders (‘#’ for an individual character, ‘*’ for any character string)

```
*$HR$ [P00++]
*$HR$ [P*]
*$HR$ [P0+1*]
```

The counting method can also be combined with the two other methods, which means for example that the following notation is also allowed:

```
*$HR$ [P0000, P0002 – P0005, P1*]
```

It is not necessary for all of the switches belonging to an infotype to be set together behind one and the same ID. Using ranges or placeholders enables you to specify an infotype more than once, which means that the switch definitions of an infotype are distributed. They are then combined automatically. If the same switch is used more than once for an infotype, only the last assignment is relevant. The multiple-value switches are an exception:

Multiple-value switches

Multiple-value switches store all of the values assigned to them in a list. This takes place internally using a table to which the next assigned value is appended. The switch overview shows you whether a switch is a multiple-value switch or not.

In accordance with the following statements:

```
*$HR$ [P0000]
*$HR$ PROVIDE_FIELD = 'MASSN'
*$HR$ PROVIDE_FIELD = 'MASSG'
```

the PROVIDE_FIELD switch is assigned the 'MASSN' and 'MASSG' values for infotype 0000.

2.2 Generating queries after switch changes

Changing a switch in, or adding a switch to an existing InfoSet affects all of the existing queries (that are based on this InfoSet). However, switches influence generated coding which means that changes to a switch do not actually have an effect until the query is regenerated. To ensure good performance, a query is not regenerated each time it is executed. Instead, it is only regenerated if its definition has changed which means that different output can be expected. However, changes to the InfoSet (such as changes to switches) are not recognized automatically. Query regeneration can be forced as follows: Access transaction SQ02, which enables you to maintain InfoSets, choose 'Goto' -> 'Query Directory', enter the name of the InfoSet, and choose 'Execute'. The system lists all of the queries for this InfoSet. To generate the queries, select them and then choose 'Edit' -> 'Generate Program'. Alternatively, you can access transaction SQ01, which enables you to maintain queries, and choose 'Query' -> 'More Functions' -> 'Generate Program'. Before doing so, make sure you restart transaction SQ01 so that if a buffer containing the old version of the InfoSet exists, it is deleted.

2.3 Checking the effects of a switch on a query

It is often unclear whether a switch defined in an InfoSet has an effect on query processing. If defined incorrectly or with an invalid value, it is simply ignored when generation takes place. The same applies to switches that are defined correctly but assigned to an infotype that is not used in the query. If you need to find out which switches were taken into account when query generation took place, proceed as follows: Access transaction SQ01, which enables you to maintain queries, and regenerate the query by choosing 'Query' -> 'More Functions' -> 'Generate Program'. Information on used switches is included in the generated query report as comment lines. To display it, you must first determine the name of the generated report by choosing 'Query' -> 'More Functions' -> 'Display Report Name'. This report can then be displayed in the ABAP editor (transaction SA38). To access the comment lines with the used switches, search for the text *HR-LOG*. There are three sections. In the first section, general messages are displayed. In the second and third sections, general and infotype-dependent switches that were used when the query was generated are listed with their values.

2.4 Note

When defining switches, you must proceed with extreme caution. Generated coding is extremely complex, which means you should only use switches if you are sure about how they work and the consequences they have. In the following sections, switches are described in as much detail as possible. However, many switches have such far-reaching effects when coding is generated that you need extensive knowledge of HR and ABAP programming skills to understand them properly. In cases of doubt, additional consulting may be necessary. In any event, switches should only ever be changed by an experienced administrator.

3 Overview of Switches

3.1 General Switches

General Switches are valid for the entire Query and are defined following the ID `*$HR$ [COMMON].`

Switch	Description
REPORT_CLASS	Set up the report class (PNP and PNPCE)
BL_ALLOW_DUP_LINES	Allow output of identical (duplicate) lines in the basic list
PROCESS_LOCKED_RECORDS	Process locked data records too (PNP and PNPCE)
PROC_PERNR_PARTIAL_AUT	Process persons too for whom a mere partial authorization exists (PNP und PNPCE)
PERSON_ONLY_ONCE	Process each person just once (PNPCE)

3.2 Infotype-specific Switches

Infotype-specific switches determine how an infotype is processed. They must therefore be specified following an ID that specifies one (or more) infotypes (for example, `*$HR$ [P0001]`).

Switch	Description	multiple-value
LAST_RECORD_ONLY	Process the last data record only	
PROVIDE	Merge neighboring/overlapping data records	
PROVIDE_FIELD	Relevant fields when data records are merged	X
PRIMARY_INFTY	Relationship of infotype with primary infotype (for infotype views)	
TIME_DEPENDENCE	Time dependence	
DATA_REQUIRED	Existence of data records required	
SPLIT_DATA_REQUIRED	Existence of data records required in split period	
NO_DUPLICATE_LANGU	output data records in one language only	
NO_INDIRECT_EVALUATION	no calculation of indirectly evaluated wage types	
IGNORE_WAGE_TYPE_OPERA	ignore operation indicator (for deduction wage types)	
CASE_SENSITIVE_SEL	case-sensitive selection (take upper/lowercase into account)	
ADD_FIELDS_SPLIT_DEP	HR additional fields are calculated with split dependency	
SPLIT_DEPENDENT_AF	technical name of a split-dependent HR additional field	X
SPLIT_INDEPENDENT_AF	technical name of a split-independent HR additional field	X

4 Procedure of a generated query

4.1 Sketch of generated coding

The following sketch is intended to give you a rough overview of the structure of generated coding. The terms in square brackets are events that are described in more detail in the next section. Reference is made to them in the description of switches.

[Declarations]
Declare infotypes, additional fields, etc.

[Data determination]
GET statement
Only take data in the reporting period into account
Optional: only take the last data record into account
Optional: eliminate multiple languages (delete superfluous data records)
Expand indirect valuation
Expand repeat fields

For each infotype:
[Additional calculations]
Calculate infotype-dependent HR additional fields
Read text fields

[Data compression]
Provide (eliminate splits from data records)

[Split calculation]
Calculate periods (splits)

For each calculated split:
For each infotype:
[Data record determination]
Determine infotype records to be output for this split

[Additional data calculation]
Calculate split-dependent HR additional fields
Calculate additional fields/tables/structures

[Data output]
Create output line (with all output fields)

4.2 Structure and procedure of generated coding

The following text briefly describes the structure of generated coding with reference to switches that enable you to control the procedure. For a more detailed explanation of the control options, see the description of each switch.

All of the variables that are needed to execute the program are declared at the event [declarations]. These are the used infotypes for the most part, but also additional fields, text fields, and other auxiliary variables.

The data records of requested infotypes are read one after the other for each selected person/object at the event [data determination]. Data records are only taken into account if they occur in the reporting period. The data records to be processed can be restricted further (see the LAST_RECORD_ONLY switch). If an infotype with indirect valuation exists, it is expanded. Repeat fields are also expanded at this event and written to separate data records. If the infotype is language-dependent, the redundant languages are deleted at this point (see the NO_DUPLICATE_LANGU switch).

Each infotype used in the query is then processed separately. At the [additional calculations] event, all of the (non-split-dependent) HR additional fields are calculated (see the ADD_FIELDS_SPLIT_DEP switch). Texts (if available and requested) on these additional fields and on the infotype fields are also calculated.

At the [data compression] event, data records created as a result of splits can be re-merged if their fields that are relevant to the query (see the PROVIDE_FIELD switch) are the same. This depends on the time constraint of the infotype and the setting of the PROVIDE switch.

If the reporting period is not restricted to a key date, the periods/splits that are relevant to output are calculated at the [split calculation] event on the basis of infotype data record validity (see the TIME_DEPENDENCE switch). The splits are calculated so that they always cover the reporting period.

Each calculated split is then processed separately. At the [data record determination] event, the valid data records of all (used) infotypes are determined for each split. The data records that are regarded as valid can also be determined by the TIME_DEPENDENCE switch.

At the [additional data calculation] event, HR additional fields are calculated if they were not calculated at the [additional calculations] event because of their split-dependency. All other additional fields, additional tables, and additional structures defined in the InfoSet are also calculated at this event.

Finally, data is output line by line at the [data output] event.

5 General Switches

5.1 REPORT_CLASS

Description

The report class concept, which enables you to determine the structure of the selection screen, applies to logical databases PNP and PNPCE. A report class can be assigned to each report that is based on one of these two logical databases. To maintain report classes in Customizing, choose Personnel Management -> Human Resources Information System -> Reporting -> Adjusting the Standard Selection Screen -> Create Report Categories. Each query is a generated report, which means that report classes can also be assigned for queries. A default report class is assigned in the standard system. The REPORT_CLASS switch can be used to specify a report class explicitly.

Standard system conduct

If the InfoSet is based on logical database PNP, report class `____X2001` is used. If this report class does not exist, report class `____22002` is used instead. If the InfoSet is based on logical database PNPCE, report class QUEPNPCE is used.

Values

SAP and customer-specific report classes can be used. PNP and PNPCE use different report classes, which means you must ensure that the specified report class was created for the logical database used in the InfoSet.

Note

Report classes are only supported by logical databases PNP and PNPCE. The switch cannot be used for InfoSets based on any other logical database.

Ad Hoc Query does not use the logical database selection screen to define selection conditions. Specifying a report class in Ad Hoc Query does not, therefore, have any recognizable effect.

Example

```
*$HR$ [COMMON]
*$HR$ REPORT_CLASS = '0MYREPCL'
```

5.2 BL_ALLOW_DUP_LINES

Description

The time-dependence of infotypes in particular can lead to a situation in which several data records exist for one and the same personnel/object number whose fields used in the query are identical. They therefore give rise to identical output. There is seldom any point in outputting identical lines because they do not contain any new information.

The ALLOW_DUP_LINES switch enables you to determine whether the process of outputting identical lines to the basic list is permitted or suppressed.

Standard system conduct

Identical lines are not output for a personnel/object number.

Values

'X' – Identical output lines (for a personnel/object number) are allowed

' ' – Standard: identical output lines (for a personnel/object number) are suppressed.

Note

This switch is only relevant to basic lists. The system does not attempt to find identical lines for statistics or ranked lists because data is compressed (aggregated) before it is output anyway.

Identical lines are only monitored/suppressed for data pertaining to the same personnel/object number. This means the basic list could very well contain identical lines if they originate from different personnel/object numbers.

Example

```
*$HR$ [COMMON]
*$HR$ BL_ALLOW_DUP_LINES = 'X'
```

5.3 PROCESS_LOCKED_RECORDS

Description

It is possible to lock individual data records in Personnel Administration. In the standard system, such data records are not processed by the query. If you want these data records to be processed, you must set the PROCESS_LOCKED_RECORDS switch.

Standard system conduct

Locked data records are not processed in the query.

Values

'X' – Locked data records are processed (together with data records that are not locked).

' ' – Standard: locked data records are not processed.

Note

This switch is only supported by logical databases PNP and PNPCE.

Example

```
*$HR$ [COMMON]
*$HR$ PROCESS_LOCKED_RECORDS = 'X'
```

5.4 PROC_PERNR_PARTIAL_AUT

Description

If data determination discovers that an authorization does not exist for all the data records of a personnel number, there are two ways of proceeding: Either the entire personnel number is not processed, or the personnel number is only processed with the data records for which an authorization exists. In the standard system, the entire personnel number is not processed by the query. By setting the PROC_PERNR_PARTIAL_AUT switch, you can ensure that the personnel number is processed with authorized data records.

Standard system conduct

Personnel numbers are not processed if authorization is missing for just one data record.

Values

`'X'` – All personnel numbers are processed with just those data records for which an authorization exists.

`' '` – Standard: personnel numbers are not processed at all if authorization is missing for just one data record.

Note

This switch is only supported by logical databases PNP and PNPCE.

Example

```
*$HR$ [COMMON]
*$HR$ PROC_PERNR_PARTIAL_AUT = 'X'
```

5.5 PERSON_ONLY_ONCE

Description

The concurrent employment model allows one person to have more than one personnel assignment at an enterprise. Each personnel assignment corresponds to a personnel number. The person is represented by the 'Central Person' (CP) object type, which is related to all personnel assignments/personnel numbers. If the query is used to report on data, all personnel assignments/personnel numbers (for which the selection conditions are satisfied) are included in the report in the standard system. However, if you are only interested in personal data that is identical for all of a person's personnel assignments, this data is output several times in the standard system: once for each of a person's personnel assignments. To avoid this redundant output, the PERSON_ONLY_ONCE switch exists for logical database PNPCE. If it is set, the relationship from personnel assignments to (central) persons is evaluated, and the system only outputs the data of one personnel assignment for each (central) person.

Standard system conduct

Reporting on the connection between personnel assignments/personnel numbers and (central) persons does not take place. All personnel assignments are processed independently of each other, as if they belonged to different persons.

Values

`'X'` – Each (central) person is processed just once. To be more exact: processing only takes place for the first personnel assignment/personnel number to be found for each (central) person.

`' '` – Standard: all personnel assignments/personnel numbers are reported on independently of each other.

Note

Only logical database PNPCE supports concurrent employment processing, which means this switch can only be used for PNPCE too.

From a technical point of view, personal data is stored redundantly for all personnel assignments. At this time, (virtually) no data is stored for the person (the 'Central Person' object). Therefore, data on (central) persons must be reported on via their personnel assignments.

You should only set this switch if you only want to report on personal data (which is stored redundantly for all personnel assignments). After this switch has been set, reporting takes place on just one personnel assignment of each (central) person. (The personnel assignment that is in actual fact reported on is more or less a matter of chance.) For this reason, all other personnel assignment-specific data is suppressed.

Example

```
*$HR$ [COMMON]
*$HR$ PERSON_ONLY_ONCE = 'X'
```

6 Infotype-specific Switches

6.1 LAST_RECORD_ONLY

Description

In the standard system, all of the data records in the reporting period are processed by the query. Sometimes, however, only the most recent (most current) data record in the reporting period is relevant, and only this data record needs to be output. In ABAP reports the RP_PROVIDE_FROM_LAST macro is often used to filter the last data record out of a set of data records. To achieve this result in the query, you can set the LAST_RECORD_ONLY switch.

Standard system conduct

All of the data records in the reporting period are taken into account/processed.

Values

`'X'` – Only the last data record (that is, the data record with the highest end date) in the reporting period is taken into account/processed.

`' '` – Standard: all of the data records in the reporting period are taken into account/processed.

Note

When data is determined, only the data record with the highest end date is taken into account. All other data records in the reporting period are ignored. Other checks regarding, for example, time constraints or subtypes are not performed for the data records. Processing continues as if this one data record were the only data record to exist. In particular, all of the selection conditions are checked for this one data record only.

Example

```
*$HR$ [P0001]
*$HR$ LAST_RECORD_ONLY = 'X'
```

6.2 PROVIDE

Description

At event [data compression], a decision is made as to whether to merge/compress split data records before processing continues. Two data records belonging to one infotype are only merged to form one new data record if they have neighboring or overlapping validity periods (BEGDA_1 <= BEGDA_2 AND ENDDA_1 >= BEGDA_2 - 1). Furthermore, the data records' field values that are *relevant* to the query must be identical. Infotype fields are regarded as *relevant* if they are used directly for selection or output, or if they indirectly influence selection or output because, for example, they are included in the calculation of an additional field. All of the text fields and HR additional fields that were calculated at the event [additional calculations] are also regarded as relevant. The PROVIDE_FIELD switch can be used to explicitly flag further fields as relevant. If two (or more) data records are merged, they are replaced by a new data record. The new data record has the earliest start date (BEGDA) and latest end date (ENDDA) of the merged data records.

The PROVIDE switch enables you to determine whether the compression option is checked at all for the data records of an infotype, or whether such data records are simply processed without being changed.

Standard system conduct

The compression option is only checked for infotypes that satisfy the following conditions:

- The infotype has time constraint 1 or 2
- The infotype does not have any subtypes
- The infotype is not a table infotype
- The infotype is not language-dependent

Furthermore, the compression option is not checked if the query uses just one split-dependent HR additional field of the infotype. See the documentation on the ADD_FIELDS_SPLIT_DEP switch.

Values

`'X'` - Check compression option, and compress data if necessary.

`' '` - Do not compress data.

Note

This switch is very similar to the PROVIDE ABAP statement.

Example

```
*$HR$ [P0001]
*$HR$ PROVIDE = 'X'
```

6.3 PROVIDE_FIELD

Description

If you set this switch, it only has an effect if the compression option is checked for the infotype (see the documentation on the PROVIDE switch).

Two data records are only merged if all of their relevant fields are identical. Relevant fields are determined automatically on the basis of the query definition. You can use the PROVIDE_FIELD switch to flag further infotype fields as relevant so that they are also checked.

Standard system conduct

Relevant fields are determined automatically on the basis of the query definition. For information on which fields are relevant, see the documentation on the PROVIDE switch.

Values

You must specify the technical name of the infotype field that is relevant to data record merging. This is a multiple-value switch, which means you can specify as many infotype fields as you want.

Note

The higher the number of fields specified by this switch, the lower the chances of two data records being merged. If you use this switch to specify all of the fields belonging to an infotype, it is tantamount to deactivating the compression option because two records must have at least one field that is different.

Example

```
*$HR$ [P0001]
*$HR$ PROVIDE_FIELD = 'PERSG'
*$HR$ PROVIDE_FIELD = 'PERSK'
```

6.4 PRIMARY_INFTY

Description

Personnel Administration includes the principle of infotype views. A new infotype (the secondary infotype) is created for an existing infotype (the primary infotype). The secondary infotype inherits the technical characteristics of the primary infotype and supplements it with additional fields. Infotype data records are maintained simultaneously (on one screen) for all of the fields belonging to both infotypes. From a technical point of view, they are still two separate infotypes with separate database tables. To facilitate assigning data records to each other, they are stored with identical keys. In the standard system, queries report on such infotypes as if they were not related to each other. Data records that belong together are not, therefore, output together. However, you can ensure that they are by setting the PRIMARY_INFTY switch. It must be set for the secondary infotype and include the name of the primary infotype.

Standard system conduct

No relationship is established between the primary and secondary infotype. All infotypes are processed as separate entities. The system does not report on data records that belong together.

Values

The switch is set for the secondary infotype and includes the name of the primary infotype.

Note

The InfoSet must include the primary and secondary infotypes. The system only supports the infotype view concept in Personnel Administration.

Example

```
*$HR$ [P0288]
*$HR$ PRIMARY_INFTY = 'P0021'
```

6.5 TIME_DEPENDENCE

A special feature of an infotype is its time constraint. It is defined each time an infotype is created, and determines the time dependence and validity of the data records belonging to this infotype. The query reads the time constraint set for an infotype, and then uses it to decide how to merge and then output the various data records. Data is only output if it is valid in the specified reporting period. If the reporting period is a key date, the situation is simple: the system outputs all of the data records that are valid on this key date. If each infotype has just one data record on this key date (which is the case for all infotypes with time constraint 1 and 2), output consists of just one line containing the data of these

infotypes. If an infotype has more than one data record on this key date (infotype with time constraint 3), more than one line is output for this infotype. The values of other infotypes are replicated. If two or more infotypes have more than one data record on this key date, all existing data records are multiplied, which means that the number of output lines is the product of the number of infotype data records.

The situation is more complicated if the reporting period is an interval rather than a key date. In this case, several data records can also exist for infotypes with time constraint 1 or 2. The question is how to output the data records. Should all of the data records be multiplied for the entire interval - as for a key date - or should the system only output infotype data records together if they have common validity? Is it sufficient for the data records to have common validity on one single day, or must they have common validity during the entire period?

The following algorithm has been implemented in the query: Infotypes are divided into three categories: period-dominant, period-sensitive, and period-independent. Each infotype belongs to just one of these three categories. The specified reporting period is divided into smaller intervals (so called splits). They cover the reporting period without leaving any gaps. These splits are calculated on the basis of the start and end dates of period-dominant infotypes. Splits are selected so that each start and end date of each and every data record (of period-dominant infotypes) coincides exactly with a split limit. As soon as the splits have been calculated, each split is processed separately as if it were a key date. The data records of period-dominant and period-sensitive infotypes that are valid on at least one day of the split are then output for each split. Furthermore, all of the data records of period-independent infotypes are output irrespective of whether they are valid in this split or not. The TIME_DEPENDENCE switch enables you to determine the category to which the infotype belongs. This explains its influence on the calculation of splits and type of output.

Standard system conduct

All infotypes with the following characteristics are classified as period-dominant (DOMINANT):

- The infotype has time constraint 1 or 2
- The infotype does not have any subtypes
- The infotype is not a table infotype
- The infotype is not language-dependent

All of the remaining infotypes (that is, the infotypes that fail to meet at least one of the criteria) are classified as period-sensitive (DEPENDENT). No infotype in the standard system is classified as period-independent (INDEPENDENT).

Values

`'DOMINANT'` - The infotype's data records influence the calculation of splits. Data records are output according to their validity for calculated splits.

`'DEPENDENT'` – The infotype's data records do not influence the calculation of splits. However, they are output according to their validity for calculated splits.

`'INDEPENDENT'` – The infotype's data records do not influence the calculation of splits. They are output for each calculated split, irrespective of whether they are valid in the split or not.

Note

If a key date is selected as the reporting period, this switch has no effect because splits are not calculated and all data records are output for the specified key date.

You are advised to use the 'period-independent' (INDEPENDENT) setting for infotypes if you are interested in ascertaining the existence of data records, rather than their validity.

Example

```
*$HR$ [P0006]
*$HR$ TIME_DEPENDENCE = 'INDEPENDENT'
```

6.6 DATA_REQUIRED

Description

Data does not always exist for all of the infotypes output in a query. If data records do not exist, the standard query response is to output initial values for the persons/objects concerned. Alternatively, you can suppress the output of such persons/objects completely. To do this, use the DATA_REQUIRED switch. If the switch is set, a person/object is only output if it has at least one data record in the reporting period.

Standard system conduct

The system outputs all persons/objects (that satisfy the selection conditions). If no data record exists for an infotype in the reporting period, initial values are output for the fields in question.

Values

`'X'` – Persons/objects are only output if they have at least one data record in the reporting period.

`' '` – Standard: the system outputs all persons/objects (that satisfy the selection conditions). If no data record exists for one of these persons/objects, initial values are output.

Note

The SPLIT_DATA_REQUIRED switch is similar and facilitates even finer system control.

Example

```
*$HR$ [P0004]
*$HR$ DATA_REQUIRED = 'X'
```

6.7 SPLIT_DATA_REQUIRED

Description

After splits have been calculated (see the documentation on the TIME_DEPENDENCE switch), valid data records are output for each calculated split. If no valid data record exists for an infotype of a split, initial values are output instead. If you set the SPLIT_DATA_REQUIRED switch, no initial values are output. Output is suppressed for the entire split instead.

Standard system conduct

If there are no valid data record for a split, initial values are output instead.

Values

`'X'` – Only those splits are processed the infotype has valid data records for.

`' '` – Standard: all splits are processed. If the infotype has no valid data record, initial values are output.

Note

The DATA_REQUIRED switch is similar. While DATA_REQUIRED enables you to determine that an entire person/object is skipped if data records are missing, SPLIT_DATA_REQUIRED enables you to determine that splits are skipped if data records are missing.

Example

```
*$HR$ [P0004]
*$HR$ SPLIT_DATA_REQUIRED = 'X'
```

6.8 NO_DUPLICATE_LANGU

Description

Some infotypes in Personnel Development contain language-dependent information. For example, infotype 1000 is language-dependent because it contains the object name in all available languages. In the standard system, the query outputs all of the data records that exist in the reporting period, which causes redundant information to be output in all available languages. To prevent this, you can use the NO_DUPLICATE_LANGU switch. If it is set, data is output in one language only. Redundant data records in translated languages are suppressed. The system first endeavors to output the data record in the logon language. If this data record does not exist, the data record is selected in accordance with language vector T778L.

Standard system conduct

The language dependencies of infotypes are ignored. All existing data records are processed and output (in all available languages).

Values

`'X'` – Redundant data records in other (translated) languages are ignored. The system only takes the data record in the logon language into account. If it does not exist, the data record with the highest priority (in accordance with language vector T778L) is taken into account.

`' '` - Standard: all data records are processed and output in all available languages.

Note

Example

```
*$HR$ [P1000]
*$HR$ NO_DUPLICATE_LANGU = 'X'
```

6.9 NO_INDIRECT_EVALUATION

Description

Some infotypes (such as 0008) contain wage types that are evaluated indirectly. The corresponding amounts are not stored on the database (which merely contains the value 0). Instead, they are calculated dynamically at runtime (in accordance with the Customizing settings).

Standard system conduct

At the [data determination] event, the system determines whether one of the wage types for infotypes 0008, 0014, 0015, and 0052 is evaluated indirectly. If this is the case, a calculation is triggered to determine the wage type amount.

Values

'X' – No check is performed to find indirectly evaluated wage types. An indirectly evaluated wage type is output with the amount 0.

' ' – Standard: the system determines whether wage types are indirectly evaluated. If so, the actual wage type amount is calculated.

Note

Indirectly evaluated wage types are calculated at the [data determination] event. Therefore, additional fields that access the wage type amount do not need to perform the indirect valuation themselves. Instead, they can use the amount that has already been calculated.

Example

```
*$HR$ [P0008]
*$HR$ NO_INDIRECT_EVALUATION = 'X'
```

6.10 IGNORE_WAGE_TYPE_OPERA

Description

Some infotypes (such as 0008, 0014, 0015, and 0052) contain deduction wage types that are included in calculations as negative amounts. This is controlled by the operation indicator. The absolute wage type amount is stored on the database (without a leading plus or minus sign). The operation indicator determines whether the wage type is a normal wage type, or a deduction wage type.

Standard system conduct

In the standard system, the operation indicator is reported on for every wage type of infotypes 0008, 0014, 0015, and 0052. If the wage type is a deduction wage type, the wage type amount is multiplied by -1 so that it is assigned a negative sign.

Values

'X' – The operation indicator is not evaluated. Just like normal wage types, deduction wage types are output without a leading plus or minus sign.

' ' – Standard: the operation indicator is evaluated, and the wage type amount is provided with a negative sign, if necessary.

Note

The operation indicator is evaluated, and the negative sign is set, just before the wage type amount is output. If additional fields that access the wage type amount are defined, they work with the absolute amount (without a leading plus or minus sign). If you want the calculation of an additional field to react to deduction wage types, the operation indicator must be evaluated by the additional field itself .

Example

```
*$HR$ [P0008]
*$HR$ IGNORE_WAGE_TYPE_OPERA = 'X'
```

6.11 CASE_SENSITIVE_SEL

Description

If you select on fields containing texts, there are two ways of proceeding: The selection is either case-sensitive (exact), or case-insensitive (tolerant). Case-sensitive means that all characters are taken into account with regard to whether they are written in uppercase or lowercase. If you select 'Miller', the system only finds this exact string and ignores, for example, 'miller' and 'MILLER'. Case-insensitive selections, which take no account of whether characters are written in uppercase or lowercase, work differently. In the standard system, the query uses the tolerant (case-insensitive) selection method. To make an exact (case-sensitive) selection, the CASE_SENSITIVE_SEL switch can be set. This also improves performance because the system does not have to devote processing time to converting uppercase and lowercase letters, and because data is selected directly from the database for all infotype fields. You are therefore advised to set the CASE_SENSITIVE_SEL switch whenever it is possible to do so.

Standard system conduct

In the standard system, selections are case-insensitive (tolerant). This facilitates flexible selection options, but is not the optimum solution as far as performance is concerned.

Values

`'X'` – All character-type fields are selected case-sensitively (exact uppercase/lowercase characters).

`' '` – Standard: all character-type fields are selected case-insensitively (tolerant uppercase/lowercase characters).

Note

If object selection has been activated, Ad Hoc Query uses a different selection method than SAP Query. To ensure optimum performance, the selection is always made case-sensitively (exactly). Setting the CASE_SENSITIVE_SEL switch therefore has no effect on Ad Hoc Query selections. It only applies to select-options defined within SAP Query.

Example

```
*$HR$ [P0002]
*$HR$ CASE_SENSITIVE_SEL = 'X'
```

6.12 ADD_FIELDS_SPLIT_DEP

Description

This switch determines how HR additional fields are calculated. For information on the features of customer-specific HR additional fields and on how to create such fields using the techniques used for HR additional fields in the standard system, access the Implementation Guide (IMG) and choose 'Personnel Management' -> 'Human Resources Information System' -> 'HR Settings for SAP Query' -> 'Additional Information on InfoSet Maintenance' -> 'Define Additional Fields'. An additional field is calculated by implementing a function module that is accessed by the query (or, to be more exact, accessed by the generated coding).

There are two points at which HR additional fields can be calculated for an infotype: at the [additional calculations] or [additional data calculation] event. In the standard system, the calculation takes place at the [additional calculations] event. The FM (for calculating the additional field) is accessed just once for each infotype data record (that exists at this time).

If the second method (the [additional data calculation] event) is used, the FM is accessed *at least* once for each infotype data record. This is because access occurs during split processing. For every calculated split (see the documentation on the TIME_DEPENDENCE switch), the system processes all of the infotype data records that are valid for this split. If

an infotype data record is valid for more than one split, it is processed more than once. The additional field is also calculated more than once.

The difference between the methods used to access the FMs lies in the way that the BEGDA_IT and ENDDA_IT parameters are supplied with data. If access occurs at the [additional calculations] event, the start and end dates of the current infotype data record are provided. However, if access occurs at the [additional data calculation] event, the start and end dates of the current split are provided. All other parameters are supplied with identical data, irrespective of which variant is used.

The event at which HR additional fields are calculated can be determined by the ADD_FIELDS_SPLIT_DEP switch. You can also use the SPLIT_DEPENDENT_AF and SPLIT_INDEPENDENT_AF switches to define different system conduct for individual HR additional fields.

The function and implementation of the additional field determines which of the two calculation methods is the right one to use. In the vast majority of cases, it will be the first (split-independent) method. This is especially so if the additional field is not time-dependent or only depends on the data of the current infotype data record. You only need to calculate the additional field for each split in exceptional circumstances.

Standard system conduct

In the standard system, HR additional fields are calculated at the [additional calculations] event, which means they are split-independent.

Values

`'X'` – All additional fields of the infotype are calculated split-dependently (at the [additional data calculation] event).

`' '` – Standard: all additional fields are calculated split-independently (at the [additional calculations] event).

Note

This switch is only really relevant to customer-specific HR additional fields. Correct system conduct has already been set up for the HR additional fields supplied by SAP (the calculation takes place split-independently at the [additional data calculation] event for most HR additional fields). Furthermore, this switch should only be used by experienced ABAP programmers who are quite sure about the effects of changed access on the calculation of additional fields.

From the point of view of performance, it is much better to calculate additional fields split-independently (that is, at the [additional calculations] event). This dispenses with superfluous (identical) calculations, all of which lead to the same result anyway. You should only flag additional fields as split-dependent if there is no other way of ensuring that the additional field is calculated correctly.

Example

```
*$HR$ [P9000]
*$HR$ ADD_FIELDS_SPLIT_DEP = 'X'
```

6.13 SPLIT_DEPENDENT_AF

Description

HR additional fields can be calculated split-dependently or split-independently (see the documentation on the ADD_FIELDS_SPLIT_DEP switch). The ADD_FIELDS_SPLIT_DEP switch changes the calculation method for all of the HR additional fields of the specified infotype, whereas the SPLIT_DEPENDENT_AF and SPLIT_INDEPENDENT_AF switches enable you to change system conduct for individual HR additional fields.

Standard system conduct

In the standard system, all HR additional fields are calculated split-independently.

Values

The technical name of the additional field to be calculated split-dependently must be specified. This is a multiple-value switch, which means you can specify as many additional fields as you want.

Note

This switch is only of interest to very experienced ABAP programmers, who should only use it if they are quite sure about the effects of changed access on the calculation of additional fields.

Example

```
*$HR$ [P9000]
*$HR$ SPLIT_DEPENDENT_AF = 'MY_ADD_FIELD_1'
*$HR$ SPLIT_DEPENDENT_AF = 'MY_ADD_FIELD_2'
```

6.14 SPLIT_INDEPENDENT_AF

Description

HR additional fields can be calculated split-dependently or split-independently (see the documentation on the ADD_FIELDS_SPLIT_DEP switch). The ADD_FIELDS_SPLIT_DEP switch can be used to flag all of the HR additional fields of an infotype as split-dependent. The SPLIT_INDEPENDENT_AF switch enables you to reset this setting for individual HR additional fields.

Standard system conduct

In the standard system, all HR additional fields are calculated split-independently. This means there is no point in using this switch unless you have already used the ADD_FIELDS_SPLIT_DEP switch to activate split-dependent calculations.

Values

The technical name of the additional field to be calculated split-independently must be specified. This is a multiple-value switch, which means you can specify as many additional fields as you want.

Note

This switch is only of interest to very experienced ABAP programmers, who should only use it if they are quite sure about the effects of changed access on the calculation of additional fields.

Example

```
*$HR$ [P9000]
*$HR$ ADD_FIELDS_SPLIT_DEP = 'X'
*$HR$ SPLIT_INDEPENDENT_AF = 'MY_ADD_FIELD_1'
*$HR$ SPLIT_INDEPENDENT_AF = 'MY_ADD_FIELD_2'
```