



---

# ***Cornerstone Project***

---

---

# ***Programming Standards - ABAP***

---

---

*Version 1.1*

---

---

*6 December 2011*

---

# Table of *contents*

<b>1</b>	<b>Document Version Control</b>	<b>5</b>
<b>2</b>	<b>Objective</b>	<b>6</b>
2.1	Audience	6
<b>3</b>	<b>Summary</b>	<b>7</b>
<b>4</b>	<b>Coding standards</b>	<b>8</b>
4.1	Package and transport request creation	8
4.2	Use of Object Oriented methodology in ABAP developments	8
4.3	Shared memory objects	10
4.4	Object Services	11
4.5	Generic Programming	11
4.6	Use of Dynamic BREAKPOINT and ASSERT statement	11
4.7	Creation of bespoke table	11
4.8	Report	12
4.9	Outputs	12
4.10	Enhancements	12
4.11	Dialog Programming	14
<b>5</b>	<b>Performance standards</b>	<b>15</b>
5.1	Summary	15
5.2	Databases Access	16
5.3	Application Layer	19
5.4	Communication (Dependency on N/W)	22
5.5	Performance Analysis Tools Usage Metrics	23
5.6	Other performance guidelines	23
<b>6</b>	<b>Naming conventions</b>	<b>31</b>
6.2	ABAP development	33
6.3	ABAP Objects Programming	35
6.4	Data Dictionary	37
6.5	ALE/IDOC Developments	39
6.6	Business Object and BAPI developments	40
6.7	Workflow Developments	40
6.8	Print Workbench and Layout Sets	44
6.9	ITS developments	45
6.10	Business Server Page (BSP) Applications	45

6.11	SAPSCRIPT Development	46
6.12	Authorisations	46
6.13	Legacy System Migration Workbench (LSMW)	47
6.14	ABAP Query	47
<b>7</b>	<b>Documentation standards</b>	<b>49</b>
7.1	Header Documentation Box	49
7.2	Comment Box	49
7.3	Comment box before FORM/METHOD declaration	50
7.4	Comment lines	50
<b>8</b>	<b>Special consideration</b>	<b>52</b>
8.1	Internationalisation and Localisation Aspects	52
8.2	Language	58
8.3	Country specific requirements,	58
<b>9</b>	<b>Quality Assurance</b>	<b>60</b>
9.1	Adherence to Programming Standards	60
9.2	ABAP Unit Test	60
<b>10</b>	<b>Security and Authorisations</b>	<b>61</b>
10.1	General	61
10.2	Reports and Programs	61
10.3	Viewing Tables	61
10.4	Maintaining Tables	61
10.5	Dialogue, Report and OO Transactions	61
10.6	ABAP Queries	61
<b>11</b>	<b>Appendix A</b>	<b>62</b>
11.1	Interfaces (non-PI) tools	62
11.2	Interface Management Solution	62
11.3	Interface directory structures	69
11.4	Interface file handling	72
11.5	Restarting of interfaces	78
<b>12</b>	<b>Appendix B – Inbound interface template</b>	<b>79</b>
<b>13</b>	<b>Appendix C – Outbound Interface Template</b>	<b>85</b>
<b>14</b>	<b>Appendix D – Axon Interface Monitor User Guide</b>	<b>92</b>
14.1	Selection screen	92
14.2	The control tree	92

14.3	Interacting with the tree	94
14.4	Interface configuration	100
<b>15</b>	<b>APPENDIX E - Technical Housekeeping Utilities</b>	<b>101</b>
15.1	Interface Run-time Monitor	101
15.2	BDC Session Monitor	101
15.3	Archive Purger	101
<b>16</b>	<b>APPENDIX F - Application Area</b>	<b>102</b>
<b>17</b>	<b>APPENDIX G - Useful Table and Structure</b>	<b>103</b>
<b>18</b>	<b>APPENDIX H – ABAP Data Type Details</b>	<b>104</b>
<b>19</b>	<b>APPENDIX I – Useful Reports</b>	<b>105</b>
<b>20</b>	<b>APPENDIX J – Useful Function Modules</b>	<b>106</b>
	Batch Input Sessions (BDCs)	106
	Text Processing	106
	Date Processing	106
	Popup Windows	106
	File Access	107
	Database Access	107
	Financial	107
	Installment Plan	108
	Dunning108	
	Miscellaneous	108
	<b>Screenshots For Popup Function Modules</b>	<b>110</b>
<b>21</b>	<b>APPENDIX K – Useful Transactions</b>	<b>113</b>
<b>22</b>	<b>APPENDIX L – Packages</b>	<b>115</b>
<b>23</b>	<b>Bibliography</b>	<b>116</b>
24	Performance Standards	116

# 1 Document Version Control

Version	Date	Comment	Completed By
1.0	11/17/2011	Initial Version	David Sentence
1.1	12/06/2011	Revised Version	Eunise Wong

## 2 Objective

The purpose of this document is to outline coding guidelines, standards and best practices in ABAP

The aim of this document is,

- ❖ To describe a consistent set of practices so that all coding will be done using uniform conventions and techniques. Therefore, once a programmer becomes familiar with these conventions, he will have a much easier time understanding all the other custom developed programs, which also follow the same set of practices.
- ❖ To recommend, explain and illustrate techniques that have been developed as efficient ways of handling certain situations in ABAP programming.
- ❖ To ensure the quality of custom developed programs, without limiting creative development.

Any development work undertaken, as part of the Connerstone Project will adhere to the guidelines and standards in this document.

### Note

- ❖ *Any new development tool, which is not part of this document should be first analysed for its usage, will have a proper entry maintained in this document of its usage. Thereby ensuring consistency in coding standards and better control.*
- ❖ *This document is not intended to replace SAP ABAP reference material and so it should not be used as means of finding out statement syntax*

### 2.1 Audience

This document is intended for use by developers before undertaking any custom developments in the area of ABAP.

### 3 Summary

Topics covered in this document are listed below,

- Coding standards
- Performance considerations
- Naming conventions
- Documentation
- Special considerations
- Quality Assurance
- Appendix
  - Axon tools
  - List of reusable programs

## 4 Coding standards

### 4.1 Package and transport request creation

Packages are designed to help developers modularise, encapsulate, and decouple units in SAP system. In the Cornerstone Project, one package would be created per work-stream. Any major bespoke development would have a sub-package created e.g. Deal construction. Every technical specification document should have the package information. Developments will not be started without appropriate package.

### 4.2 Use of Object Oriented methodology in ABAP developments

SAP introduced ABAP objects, the object oriented extension of ABAP as part of release 4.6C. This provides the entire required framework for developing programs using Object Oriented methodology.

They provide all the benefits of Object Orient programming and more,

- ❖ Data Encapsulation
- ❖ Reusability, through inheritance
- ❖ Event driven, Applications can be loosely coupled, providing greater flexibility

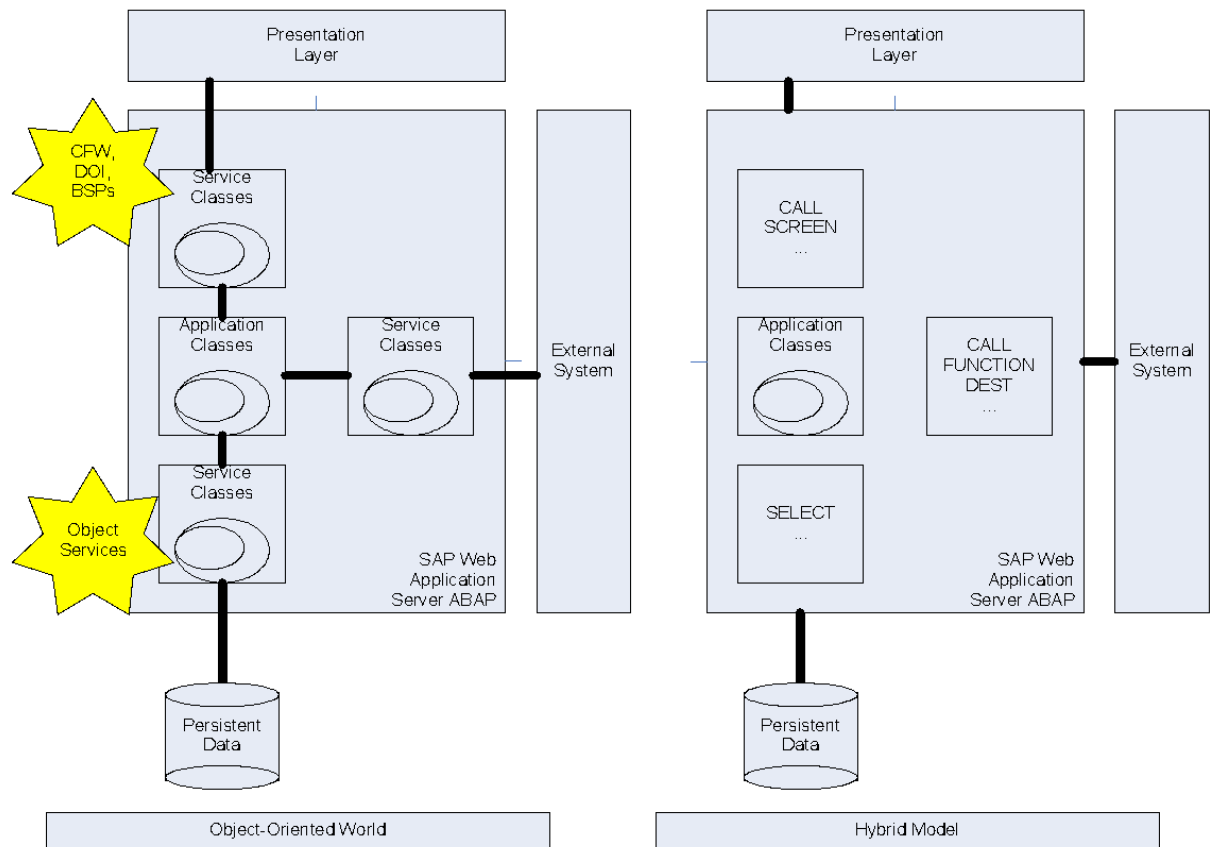
#### 4.2.1 Why ABAP Objects

Advantages of using ABAP objects and classes,

- ❖ ABAP objects are explicit, and therefore simpler to use. (Compared to Logical database, reports)
- ❖ ABAP objects offer cleaner syntax and semantic rules. E.g. obsolete and error-prone language concepts are strictly forbidden inside ABAP object classes.
- ❖ Only way to use new technology SAP Control Framework and BSPs are encapsulated in ABAP object classes.



### 4.2.2 Usage strategy



For all bespoke developments, first option will be to use Object Oriented programming, unless the OO programming is an overhead or is not feasible, e.g. like classical dialog programming (using ABAP Dynpro), customer-exits, form exits which involves simple coding requirement. Following guidelines will be followed,

- ❖ Use of function modules to minimum, e.g. encapsulating classical screens (Dynpros) or developing RFCs.
- ❖ Use methods instead of subroutines.
- ❖ Decouple classical screen programming from Application programming using ABAP Objects.

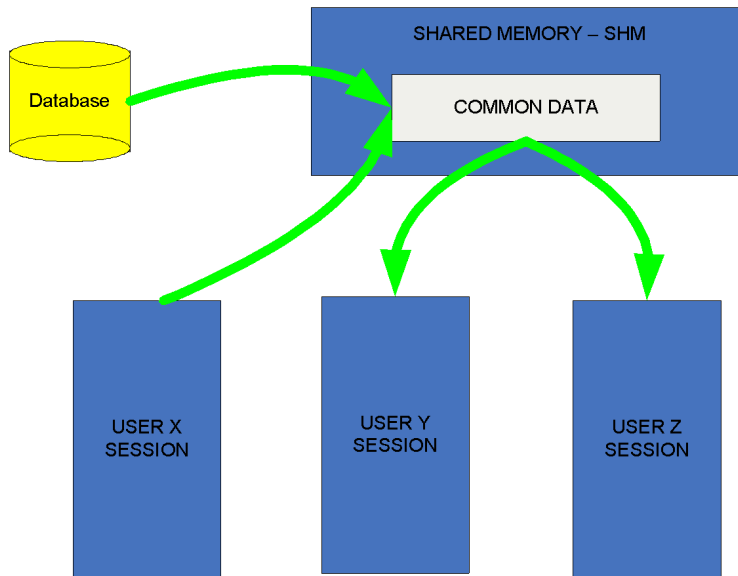
### 4.2.3 Obsolete Statements

Questionable constructs are still allowed outside ABAP Object classes to support the strict downward compatibility requirement of ABAP, thereby not requiring changing programs with every new release of ABAP. These statements must not be used in the procedural programs. **Use of these statements is highlighted as warning message in Code inspector and extended program check, and should not be ignored.**

- ❖ List of Obsolete statements in ABAP  
Go into display mode of any program using transaction SE38. Press HELP Icon or (CNTRL+F8) -> Select ABAP overview radio-button -> press enter -> Select the section Obsolete Statements and Concepts

## 4.3 Shared memory objects

From WAS6.40 (which would be used as a platform for ESAP applications), SAP has introduced shared objects for accessing data across user sessions. Thereby reducing the costly database reads for common data required across sessions. Transactions SHMA can be used to maintain the Shared Object Area, and SHMM can be used for monitoring. A root class, with Shared Memory-Enabled attribute checked needs to be created using Class builder.



### 4.3.1 Advantages

- ❖ Reduced memory consumption
  - Data is kept only once in the memory
- ❖ Reduces runtime costs
  - Data is aggregated only once
  - Fast access since access to the main memory not the database
  - In place aggregation for “arbitrary” data structures including strings, internal tables and references.
- ❖ Reduced maintenance costs due to
  - Simplified programming model in contrast to solutions based on import / export.

### 4.3.2 Usage Scenarios

- ❖ Shared buffers
  - Low update frequency (e.g. once a day to once an hour)
  - (Very) costly amount of data
  - Many parallel readers, single writer
- ❖ Shared buffers
  - Access by single writer or reader
  - Buffer concept is kept across transactions
- ❖ Coarse granularity locking concept
  - Only complete buffer can be locked
  - Rare updates
  - Readers and writers know each other

## 4.4 Object Services

Object services provide global services, which ABAP Object language elements cannot provide themselves, to applications written in ABAP Objects. These Object services include A Persistence Service and Transaction Service. Application objects are managed using Object Service objects known as class actors or agents. There is exactly one class actor for each application class. Please see the help document for further information.

[http://help.sap.com/saphelp\\_erp2004/helpdata/en/ab/9d0a3ad259cd58e10000000a11402f/frameset.htm](http://help.sap.com/saphelp_erp2004/helpdata/en/ab/9d0a3ad259cd58e10000000a11402f/frameset.htm)

## 4.5 Generic Programming

Special consideration will be given to the use of generic programming techniques to make the code reusable.

The generic programming concept is based on Object Patterns and uses Run Time Type Services (RTTS) to achieve this. Generic programming techniques will be used while developing relevant template programs.

## 4.6 Use of Dynamic BREAKPOINT and ASSERT statement

BREAKPOINT statements have been used as part of testing and error analysis exercises in previous versions of SAP. But these were static breakpoints and had to be commented before the program was transported to production. The soft breakpoint can also be set in the program or while debugging but it's a painstaking activity if one is not aware of program logic. In WAS6.40 new dynamic break-points have been introduced whereby an application programmer or programmer developing bespoke applications can code these breakpoints at strategic locations which will allow easy analysis of the program later. These breakpoints can be activated or deactivated from one central location transaction SAAB and also selective for certain Users. The dynamic breakpoints do not pose any execution overhead in the production system as long as they are not activated.

ASSERT statement can be used to detect deviation of the program from the normal behaviour. If a certain condition is not expected then ASSERT statement can highlight such a situation. A detailed log is generated which can point to areas in the program from which unexpected behaviour started. Similar to the dynamic breakpoints, ASSERT statement can also be maintained centrally through transaction SAAB. The behaviour of the ASSERT statement can be changed through this transaction, which can be aborting the program, or just generating a log without stopping/aborting the program.

### 4.6.1 Usage

Dynamic breakpoints and ASSERT statements will be coded in bespoke development to simplify debugging and analysis. A group name will always be the same as the program name in which these BREAKPOINT and ASSERT statements will be used.

## 4.7 Creation of bespoke table

The bespoke tables can be client dependent or independent, depending on whether the MANDT field is used as the first key field in the table. It is important to note in case of bespoke table creation, the question to be asked is how the data will be maintained, through enhancement /user-exit or manually. In case it is maintained manually, then there are two options, creating a dialog program for updating the values or using table maintenance generator. Please note that the table maintenance generated transaction has one drawback for single screen options, it does not support multi-user maintenance, i.e. only one user can maintain the values at any given moment of time. Also activation of buffers will be considered based on the guidelines specified in Performance Standards section.

Please note that there will be no generic authorisations for transactions SE16/SE16N and SM30/SM31, so use of authorisation group is mandatory for giving access to the right people to maintain a bespoke table. One authorisation group would be created per work-stream. All the bespoke tables maintained through table maintenance generator will have an authorisation group.

### 4.7.1 Usage.

The table maintenance dialog option should only be used to set control information, or similar table values that change infrequently, but will change. Examples are tax rates, overhead rates, program control tables, message tables, program or error status tables.

## 4.7.2 Modification

Also note that there are exits for any enhancement required for table maintenance dialog.

Transaction Se55 ->Environment -> Modification

## 4.8 Report

There are several options for developing report program; classical reports, ALV list, and ALV grid using function modules or OO (classes). From WAS6.40 (ESAP will be using this platform) SAP has introduced SAP ALV based on a new object oriented programming model supporting ABAP list viewer and SAP control framework technology. In all new reporting requirements SAP ALV will be used as a default, refer transaction se83 for report template program for The City. You can also find other examples and options in the same transaction. Only in the rare case where there is need to copy and change a standard SAP report would options other than ALV grid using OO will be used.

### 4.8.1 Demo programs and important transactions

Demo programs for SAP ALV - SALV\_LEARN\*, SALV\_DEMO\*, SALV\_FORM\_DEMO\*  
For graphical reporting try these program examples GFW\_PROG\* and GFW\_DEMO\_\*

Transactions SE83(Reuse Library) and DWDM(Demo Centre), have demo programs for ALV Grid and use of other controls like tree, text edit.

## 4.9 Outputs

Outputs can be developed using following tools SAPSCRIPTS, SMARTFORMS and INTERACTIVE FORMS (Adobe).

Refer [863893 Interactive forms: Release restrictions \(NetWeaver '04\)](#)

Preconfigured smartforms supplied by SAP would be used wherever possible. For systems based on Netweaver 04s interactive form (Adobe) should be used as the first option. For WAS6.40 system, Smartforms should be used for developing any new output requirements, SAPSCRIPT will be used only in case there is standard SAP output, which requires a very small change. Though SAP has released Interactive Forms (Adobe) from release WAS6.40, there recommendation is not to use it for productive purpose, instead Smartforms should be used, since it is easier to migrate to Interactive Forms during an upgrade.

### 4.9.1 Fixed texts

Any fixed text information required in output, will be maintained in standard texts (SO10) and this will be clearly documented in the technical specification document.

### 4.9.2 Important links

Interactive forms(Adobe)

<http://www.adobe.com/enterprise/partners/sap.html>

<http://service.sap.com/adobe>

Preconfigured Forms

[http://help.sap.com/bp\\_presmartformsv1500/index.htm](http://help.sap.com/bp_presmartformsv1500/index.htm)

## 4.10 Enhancements

Listed below are enhancement options available and their usage priority i.e. whenever there is an enhancement required, first check whether this can be done using Easy Enhancement Workbench, if not then check if BADIs can be used. If none of the above is available only then other enhancement options should be considered in the order specified in this document.

### 4.10.1 Easy Enhancement Workbench

The Easy Enhancement Workbench (Transaction -> EEWB) can be used to extend some Business Objects mainly in CRM environment. The type of extension is pre-defined and defined via wizards. The system then creates all the objects database tables, screens and application logic automatically. And this customer enhancement is included in the SAP standard.

This also allows users without ABAP knowledge the simple possibility of extending the SAP standard.

The extension can take place system-wide e.g. when extending a Business Object in CRM the data exchange to R3 is also extended and a table is created in the R3. For this to happen the System Landscape (SLD) needs to be setup.

An extension created through Easy Enhancement workbench does not differ technically from one created manually.

*Oss note: 494966      Composite SAP Note for Easy Enhancement Workbench*

### 4.10.2 Business Add-ins (BADI)

For each Business Add-In you have one interface and an adapter class that implements this. The user implements the interface. BADIs can be multi-instance. In which case more than one implementation can be active at any given moment. Also consider the filter entries, enter '\*\*' if CLIENT field is one of the filter field.

For any enhancement, BADIs will be preferred over Customer-exits were ever there is an overlap e.g. purchase order enhancement. Please note that with current versions of SAP BADIs also have enhancement for screen, GUI interfaces and tables.

Most of the BADIs can be located through the customising transaction – SPRO or through a search or the statement “CL\_EXITHANDLER” in the program.

#### 4.10.2.1 Transactions

Definition transaction	SE18
Implementation transaction	SE19

### 4.10.3 Business Transaction Events (BTE)

BTEs are predecessors of BADI, and available mostly in FI modules. Coding is done in function module which has fixed interface provide by the SAP. These can be of two types.

**Publish and Subscribe** in which case the additional activities could be carried out. The data cannot be sent back to SAP standard.

**Process** This allows sending back data to SAP standard.

BTE's can be located either by using the transaction FIBF or through a search for statement “OPEN\_FI” in the program.

#### 4.10.3.1 Transactions

BTE configuration	FIBF
-------------------	------

### 4.10.4 Customer Exits / Enhancements

These are one of older enhancements technique available in SAP, and can have function exits, customising include, screen exits and menu exits. BADIs should be used instead of customer-exits in case of overlap of functionality.

#### 4.10.4.1 Transactions

Definition transaction	SMOD
Implementation transaction	CMOD

### 4.10.5 Form Exits

These are enhancements, which use subroutines; these are specific to Sales and Distribution module. Again if there is an overlap of functionality between BADIs and Form exits then BADIs must be used.

A detail documentation of these exits is available in customising - transaction SPRO -> Sales and Distribution -> System Modifications -> IMG activity documentation or you can also find this documentation on SAP help.

#### 4.10.5.1 Notes on Usage

Each form exit or subroutine will have an include program, the coding will be done only in these includes and never in the subroutines.

### 4.10.6 Routines

These can formulae or requirement routines, which need to be coded, in case SAP standard ones do provide the required functionality. Most of these can be maintained through transaction "VOFM". Always a copy of SAP standards routine will be made and then changed as per requirement. There is always some amount of customising involved along with the coding in the routine for the solution to work. Also routines would be required in the substitution and validations area in FI module.

### 4.10.7 Append Structures

Append structures are used for enhancements that are not included in the standard. This includes special developments, country versions and adding customer fields to any tables or structures.

Please ensure the SAP recommended naming convention for naming the append structure fields. The following enhancements can be made to a table or structure with an append structure:

- ❖ Insert new fields
- ❖ Define foreign keys for fields of table/structure that already exist
- ❖ Attach search helps to fields of table/structure that already exist

#### 4.10.7.1 Notes:

**You must not use INCLUDE/APPEND structures in BAPI structures, because enhancements to INCLUDE structures generally lead to incompatible changes to the BAPI structure.**

### 4.10.8 Field Exits

Field Exits can be used, but are frozen at the maintenance level in 4.6C, this means the existing functionalities cannot change and is kept with all restrictions. These enhancements do not invalidate the guarantee. Program RSMODPRF is available for coding field exits

*Oss note 29377 FAQs: field exits (CMOD)*

## 4.11 Dialog Programming

This development involves creation of screens for capturing manual data entry. With the current version of SAP any dialog development should follow the MVC model for GUI development. The development can use any of the following techniques ABAP dynpro(SAP GUI), PCUI, BSP or Webdynpro. This depends on the kind of GUI access given to the user.

Special care should be taken while programming the database updates, please see the notes on SAP LUW, logical unit of work and also use of CALL FUNCTION ... IN UPDATE TASK in the Performance Standards section. Also using the number ranges and SAP locking could have significant impact on the performance and therefore guidelines mentioned in Performance Standards section should be followed strictly.

Dialog program is always required in case there is need to develop a bespoke application. Question related to data migration should be asked and considered during design and development.

Any ABAP dynpro screen design will follow the SAP guidelines from SAP Style guide.

[http://help.sap.com/saphelp\\_erp2004/helpdata/en/e1/8e51341a06084de10000009b38f83b/frameset.htm](http://help.sap.com/saphelp_erp2004/helpdata/en/e1/8e51341a06084de10000009b38f83b/frameset.htm)

## 5 Performance standards

### 5.1 Summary

The strength of multi-tier system is scalability, therefore a program can be written on a single server development system with very few users, but could be deployed on multi server production systems which processes huge volume of data, across continents, with thousands of people using it concurrently. So, scalability considerations need to be given when developing any program.

Also in multi-tier architecture, the load optimisation potential is the largest around 40-60% at the application layer. This can be achieved through proper process design, **applications** and customising.

#### 5.1.1 Programming guidelines to achieve scalable programs

##### Database Layer

- ❖ Use of appropriate indexes  
To optimally use the database, all frequently executed accesses to the database layer are supported by an appropriate index.
- ❖ Keeping the amount of data read to the minimum.
- ❖ Buffers and Caches  
No identical accesses to database layer

##### Application Layer

- ❖ Parallel Processing enabling  
Keep Enqueue locking time in programs to the minimum.
- ❖ Linear dependency  
Avoid any memory leaks  
There should not be a nonlinear increase in the processing time with increase in amount of data processed.
- ❖ Communication (Dependency on N/W)  
Link between presentations Layer  
Link between servers

##### Other performance guidelines

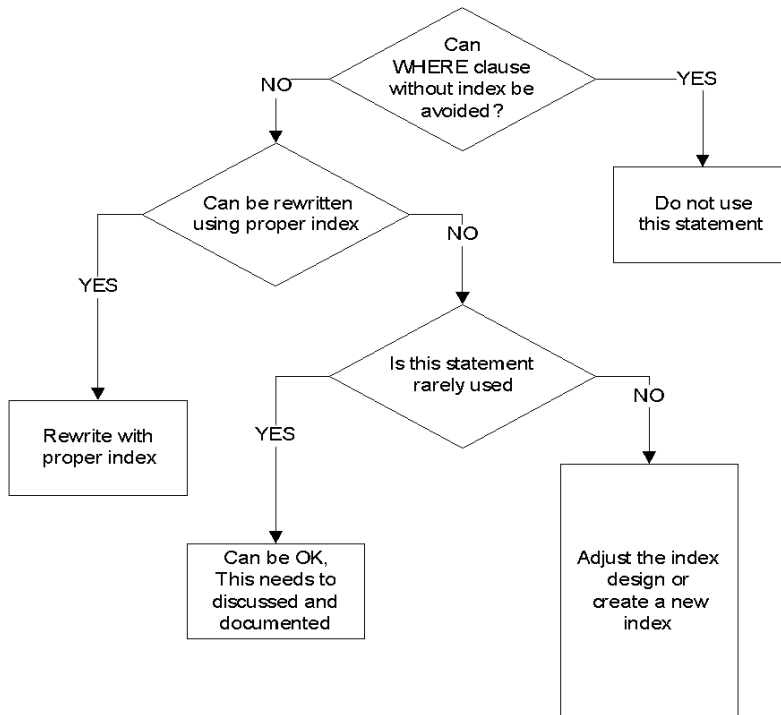
- ❖ Dead code
- ❖ Subroutine usage
- ❖ IF statements
- ❖ CASE v/s IF statements
- ❖ Loop statement
- ❖ Nested SELECT v/s JOIN
- ❖ Avoid unnecessary statements
- ❖ SINGLE SELECT is faster than SELECT UP TO 1 ROWS

## 5.2 Databases Access

### 5.2.1 Use of Appropriate Indexes

#### 5.2.1.1 Writing WHERE clauses.

Use the complete key or partial key in the WHERE clause. Ensure that primary key index or appropriate secondary index fields are used in the WHERE clause. If a situation arises where there is need to write a WHERE clause without index, then follow the flow chart below.



#### 5.2.1.2 Creating Secondary Indexes

A proper index design ensures that the search time is nearly independent of the number of table rows.

- ❖ Columns to be used with EQ (in the WHERE clause) should be at the beginning of index
- ❖ A highly modified table should not have too many indexes
- ❖ The more columns an index has, the higher the chance that an update will affect one of the indexed columns
- ❖ The cost for maintaining indexes depends heavily on the total length of all index fields

#### 5.2.1.3 Checking proper indexes are being used

##### 5.2.1.3.1 Code inspector - SCI

- ❖ A static check can be conducted on the program to ensure WHERE clause uses appropriate index on the table.

##### 5.2.1.3.2 Performance Analysis -ST05

- ❖ Sort column Min.time/r. values higher than 10,000µs, it is quite probable that these database accesses are not supported by proper index. Use Explain on this statement to check the access path.



## 5.2.2 Keeping the amount of data transferred to the minimum.

The application should not read data before it is guaranteed to use it. If in doubt, records should be read one by one, as they are needed.

### 5.2.2.1 WHERE clause

Use WHERE clause, instead of CHECK statement.

Use

```
SELECT * fom
WHERE name ...
ENDSELECT.
```

Instead of

```
SELECT * fom
CHECK .
ENDSELECT.
```

Ensure the data read is actually needed by the application i.e. Accesses to the database layer is with complete WHERE clause.

#### Note: Use of SELECT FOR ALL ENTRIES

Use DELETE ADJACENT DUPLICATES prior to executing the statement

Check if the internal table were not empty before using SELECT FOR ALL ENTRIES, else all records from the table would be fetched the WHERE clause would be ignored.

### 5.2.2.2 SELECT \* v/s SELECT field list

Try using projection view in the data dictionary, or use of field list in the select clause.

Use

```
SELECT field1 field2
INTO
FROM
WHERE
```

Instead of

```
SELECT *
FROM
WHERE
```

### 5.2.2.3 Use of SAP standard function module

Using the wrong function modules to fulfil a certain subtask can also cause unnecessary database accesses, such as when you call a function that does everything you want plus a few things you don't care about. Each function module used should fit your needs as closely as possible.

### 5.2.2.4 Make meaningful use of update modules.

Another statement that causes database access without an explicit SQL statement is CALL FUNCTION ... IN UPDATE TASK. The addition ... IN UPDATE TASK changes a simple function module call, which costs only a few microseconds, into an update call, for which at least two data sets are written to the database. The update task reads these data sets to regain interface information so that the function may execute with the correct parameters. After the update is completed, the data sets are deleted again automatically by the system. Although using the update task has advantages both for the dialog response time and the locking behaviour of most transactions, you should try to reduce the number of CALL FUNCTION ... IN UPDATE TASK whenever possible, which leads to the following guidelines:

- ❖ Check whether interface tables have at least one entry.
- ❖ Try to bind together several CALL FUNCTION ... IN UPDATE TASK statements.
- ❖ Avoid at all costs calling update function modules within loops.

### 5.2.2.5 Checking where more data is being fetched than required

#### 5.2.2.5.1 Code inspector – SCI

- ❖ Statements causing unnecessary data reads will be found like, CHECK statements within SELECT and ENDSELECT, also SELECT statements without WHERE clauses

#### 5.2.2.5.2 Performance Analysis - ST05

- ❖ Sort the list by column records, which show the total number of records, which have been selected and transferred by the statement. The higher number than expected is an indication that more entries than required are read.
- ❖ Sort the list by column tables. Count the total number of records per database table and compare it with the number of expected records.
- ❖ Identify database accesses to the same table entries from different programs. Ensure that the same information is read only once from the database using appropriate buffer mechanisms

### 5.2.3 Buffering and Caching

Buffering ensures that the data, which is repeatedly required by programs, can be buffered thus reducing costly data reads from the database.

#### 5.2.3.1 Avoiding statements that bypass buffers (on tables that are buffered)

Following statements bypass buffers; these statements need to be used with caution. Consider use of statements that invalidates/bypasses buffer

To take full advantage of buffering, you must be aware of the technical settings of a table as well as the techniques that bypass buffer processing. The following statements can be used to bypass buffer processing:

- ❖ `SELECT ... BYPASSING BUFFER`
- ❖ `SELECT ... FOR UPDATE`
- ❖ Any aggregate function (COUNT, MIN, MAX, SUM, AVG)
- ❖ `SELECT MIN(F1) FROM T1 WHERE ...`
- ❖ `SELECT DISTINCT ...`
- ❖ WHERE clause contains `... IS [NOT] NULL`
- ❖ `ORDER BY` (other than PRIMARY KEY)
- ❖ `GROUP BY` or `HAVING`
- ❖ JOINS and sub-queries
- ❖ Use of mandt field in join. Pointer to the Oss note:
- ❖ `SELECT ... FOR ALL ENTRIES` - if not all data is buffered
- ❖ For single-record buffering every statement where key is not fully specified: `no SELECT SINGLE`
- ❖ In generic case: no generic key
- ❖ Any native SQL statement (even INSERT, UPDATE, DELETE).
- ❖ Comparison between database columns: for example
  - `"SELECT * FROM DBTAB~FIELD1 > DBTAB~FIELD2"`

#### 5.2.3.2 Buffering tables

Table buffers are best suited for read-only data stored in customizing tables or small tables containing master data. When you choose a buffer type, make sure that it fits the table size and accesses: It takes much more effort to change a buffered table than a non-buffered one. Therefore, table buffers are best suited for read-only data stored in customizing tables or small tables containing master data.

##### Note: Buffering allowed but off

- ❖ In customer system buffering can be switched on (installation-dependent)
- ❖ Specification of buffering type is optional

#### 5.2.3.3 Use of READ modules and SHARED OBJECT Buffers.

Read modules are special function groups (e.g. Using read modules: `ME_EKKO_SINGLE_READ`) or classes that store information read from the database in global internal tables. The visibility of data is restricted to one SAP logical unit of work (LUW).

Read modules should be used:

- ❖ When buffering the information using technical setting is not feasible
- ❖ To prevent unnecessary database calls when the same data is used by different program units of the transaction
- ❖ To ensure consistent data within one transaction

To check identical selects statements in transaction ST05  
Second column "No. of identical selects in percent".

#### 5.2.3.4 Checking statements which need buffering

##### 5.2.3.4.1 Performance Analysis - ST05

- ❖ Second column "No. of identical selects in percent". This statements which are making calls to the database using same select clause. This is an area where buffering mechanism could be used.

## 5.3 Application Layer

### 5.3.1 Parallel processing

- ❖ Schedule multiple batch jobs  
Optimal for long running independent tasks
- ❖ Use tRFCs  
Use for short independent tasks that may run on other systems
- ❖ Use aRFCs  
Useful for short tasks that must be synchronised
- ❖ SAP Locking and Database Locks  
Locking prevents parallel processing. The locking should be for least possible time; unnecessary locking can cause performance issues. The importance of the lock is determined by how many users are likely to access the object simultaneously. Special attention should be given to the code, which is executed during which the lock is applied. Firstly, on reducing the runtime and also executing at least part of required program steps before lock is applied or after it is released.

#### 5.3.1.1 Checking Parallel processing bottlenecks

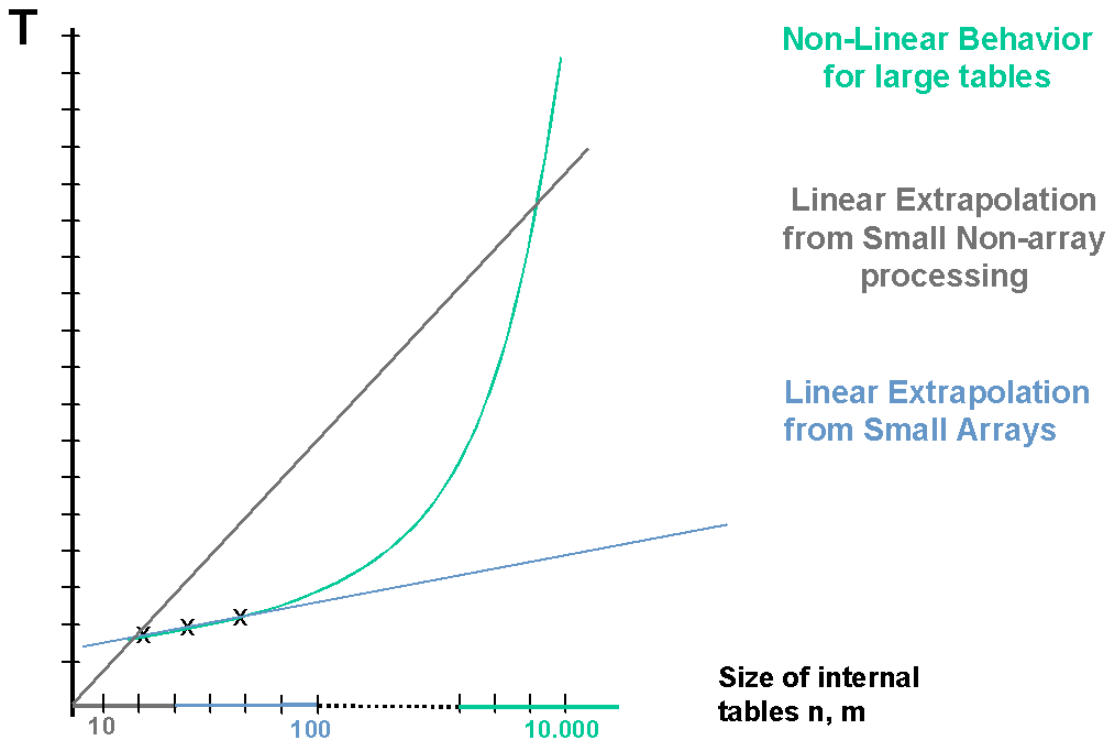
##### 5.3.1.1.1 SAP Enqueue

- ❖ Monitor the current system wide Enqueue situation with SM66 or SM50
- ❖ Run Performance Trace (ST05) with option Enqueue Trace on. Display extended trace list to see the timestamp for each statement. Look for exclusive locks and when they are released either explicitly or implicitly at the end of SAP LUW. Measure the locking time.

##### 5.3.1.1.2 Database Locks

- ❖ Run Performance Trace (ST05) with option SQL Trace. Display extended trace list to see the timestamp for each statement. Looks for locks set by INSERT, UPDATE, DELETE statements and for next COMMIT WORK, which releases the locks. Measure the locking time.
- ❖ Make sure that each exclusive lock is necessary and cannot be changed into shared lock.

### 5.3.2 Linear dependency.



*[Please note that the “Size” axis is logarithmic, the graph shows that the programs having nonlinear behaviour with increase in data].*

#### 5.3.2.1 Internal tables

Avoid too large internal tables, Try to keep internal tables small the size of the internal table can be checked in SE30(Runtime Analysis).

Use the right table type:

- ❖ Standard table for multiple access types, use BINARY SEARCH for mass accesses and try to SORT only once
- ❖ Sorted table, if generic key access is main access type
- ❖ Hashed table if single line access with fully specified key is only access type

##### 5.3.2.1.1 Hash tables V/S Sorted tables

**Hashed tables can only be used if main access uses full table key**

- ❖ Generic accesses are very slow
- ❖ Best example: Key contains a GUID

**Which single line access is faster?**

- ❖ The scaling of hashed-tables is better
- ❖ However, access costs are similar to sorted tables for sizes up to 100

- ❖ There is a slight advantage for hashed tables for very large tables

**Things to note:**

- ❖ That internal tables should generally not become too large
- ❖ Most large tables are large because operations work on generic key ranges and not only on single lines

**5.3.2.1.2 Sorted Standard tables V/S sorted tables**

**Sorted Standard Table**

- ❖ Access with binary search as good as access on sorted table
- ❖ Most flexible table
- ❖ Different access types can be supported
- ❖ Fastest filling method:
- ❖ If it is not necessary that the table is always sorted during filling
- ❖ Use APPEND , ... , APPEND and one SORT at the end
- ❖ A small number of additions should use INSERT instead of APPEND
- ❖ Sort is expensive as  $O(n \log n)$ .
- ❖ Every unnecessary extra sort should be avoided
- ❖ Note: The most expensive SORT is the one on an already sorted table

**Sorted tables**

- ❖ Are much more convenient if a specific order must be always guaranteed

**Table Type Usage Matrix**

	Standard	Sorted	Hashed
Main access range	Variable	For key ranges	For single lines
Main access type	By index	By key	Only by key
Key	Non unique	Unique or non unique	Must be unique
Remark	Most flexible type		

**5.3.2.1.3 Filling data into Internal table**

**From DB tables**

- ❖ Best with array fetch INTO TABLE or APPENDING TABLE
- ❖ If possible, try to use identical line types as the command INTO CORRESPONDING FIELDS is expensive, especially for buffered tables

**From other internal tables**

- ❖ Best with array fetch APPEND LINES OF or INSERT LINES OF
- ❖ Beware that MOVE-CORRESPONDING works only on the work area

With COLLECT lines with identical key are aggregated to one line, all numeric fields are summed up automatically

**APPEND**

- ❖ Only for standard table: independent of n

#### INSERT

- ❖ Standard table:  $\sim n$   
Location (Index) must be determined with READ. Use BINARY SEARCH as insert is only useful on sorted standard tables. Becomes faster for  $n > 4096$  when a B\* index is used.
- ❖ Sorted table:  $\sim \log n$   
Direct INSERT with similar cost as above. Rearrangement is less expensive as a B\* index is used
- ❖ Hashed table:  $\sim$  independent of  $n$   
Finding location is cheap. Rearrangement of the hashed function

#### 5.3.2.1.4 Reading data from Internal table

Work area versus header line

- ❖ Try to avoid header lines, they can cause errors

#### TRANSPORTING

- ❖ READ TABLE works with TRANSPORTING field list and 'NO FIELDS'
  - ❖ LOOP AT ITAB not with TRANSPORTING field list.
  - ❖ Use field list if amount of transported data is reduced considerably
  - ❖ Use TRANSPORTING NO FIELDS if only SY-INDEX or SY-SUBRC is needed
- ASSIGNING (from SAP R/3 4.5)
- ❖ Direct table access without copying to work area
  - ❖ Also no MODIFY necessary
  - ❖ Use ASSIGNING especially in nested tables

Slowest table access are sequential and grow linearly with  $n$

- ❖ READ standard table with (table) key  
(no BINARY SEARCH)
- ❖ LOOP at table where
- ❖ READ hashed table with key (not table key)

**Faster access use a sorted index and have a weak dependence on  $n$**

**Note:  $n =$  number of lines in the internal table**

- ❖ READ standard table with key  
with BINARY SEARCH (must be sorted)
  - Sort statement is expensive; use sort only once.
- ❖ READ sorted table with table key

**Fastest accesses are independent of  $n$**

- ❖ READ hashed table with table key
- ❖ READ standard table with index
- ❖ READ sorted table with index

#### 5.3.2.2 Checking for nonlinearity in processing times

##### 5.3.2.2.1 Code inspector – SCI

Poor-performing operations on internal tables select check Low Performance Operations on Internal tables. Nested loops and selects can be found using following checks,

- ❖ SELECTs in loops
- ❖ Changing database access in loops

- ❖ Nested loops

**5.3.2.2 Runtime Analysis Trace – SE30**

- ❖ Prepare two test cases. The only difference between two cases is increase of the data size by a factor of 10(x10).
- ❖ Run two tests cases and trace them with SE30
- ❖ Compare entries in the hit list of the two trace files. Runtime with an increase by a factor of more than 10 indicate highly nonlinear coding.

**5.3.2.3 Memory**

There can be dynamically growing memory consumption through Internal tables, ABAP / JAVA objects, Strings, Anonymous data objects. Controlling memory consumption can improve buffer use and let more users work on one instance.

**5.3.2.4 Checking for nonlinearity in memory usage.**

**5.3.2.4.1 Debugger**

- ❖ Use debugger to create memory snapshot after execution of subsequent tasks/packages/units.
- ❖ Use transaction S\_MEMORY\_INPSECTOR to compare the memory snapshots

**5.3.2.4.2 Tune Summary (ST02)**

- ❖ GOTO -> Current Local data -> SAP memory -> Mode list
- ❖ Display an overview of the memory size of each user session.
- ❖ Note the value of the size of the user session after subsequent tasks/packages/units.

**5.4 Communication (Dependency on N/W)**

**5.4.1 Minimising round trips**

**5.4.1.1 Communication with presentation server**

Network run times include uncertainty factors. Special care needs to be taken while code, which is influenced by Network speed. This can involve communicating with GUI, or with another server. Keep the data exchange to the minimum; follow the guidelines of 2 trips for communication with GUI. Avoid e.g. a program with a greater running time for the SAPGUI Progress Indicator than for the rest of the program, even without the satellite link. A remote function call (RFC) sends this SAPGUI Progress Indicator to the front end. Because a work-process is blocked on the application server during an RFC roundtrip, bad performance at the presentation-server level influences performance at the application-server level.

- ❖ Make reasonable use of the SAPGUI\_PROGRESS\_INDICATOR
- ❖ Do not synchronize the contents of controls to often (explizit flush)

**5.4.1.2 Server-to-Server communication,**

- ❖ Optimize the number of RFCs
- ❖ Use RFC connections as long as possible

**5.5 Performance Analysis Tools Usage Metrics**

Basic Tests	SCI	ST05	SE30	ST30	ST03	DB	OS
Premium Tests					N	monito	monito
Platform dependent						r	r
<b>General Checklist</b>							
Appropriate Indexes	X	X					

Complete WHERE clauses	X	X		X			
SAP buffer	X	X		X			
No identical selects		X		X			
Parallel processing enabled		X					
Linear dependency	X						
Two communication steps per dialog steps		X	X		X		
<b>Actual Measurements</b>							
Number of DB calls		X		X		X	
Net data volume transferred(KB)		X		X		X	
Peak memory consumption(MB)				X	X		X
CPU consumption of functions / methods (msec)				X	X		X
Number of round trips to presentation server and between servers		X		X	X		X
Amount of bytes transferred to presentation server				X	X		X

## 5.6 Other performance guidelines

### 5.6.1 Code examples in “Performance Tips and Tricks for ABAP Objects”

This contains very good programming tips and tricks for better program performance. You can reach this by getting into any program in display mode using transaction SE38 ->Environment ->Examples->Performance Examples. This has very good examples showing tips on performance improvements. Please use these with due consideration to the assumptions. Some of the examples are covered below.

### 5.6.2 “Dead” code

Avoid leaving "dead" code in the program. Comment out (or delete) variables that are not referenced and code that is not executed. Use program --> check --> extended program check to see a list of variables that are not referenced statically.

### 5.6.3 Subroutine usage

For good modularization, the decision of whether or not to execute a subroutine should be made before the subroutine is called. For example:

This is better

```
IF f1 NE 0.
  PERFORM sub1.
ENDIF.
FORM sub1.
...
ENDFORM.
```

Than this

```
PERFORM sub1.
```



```

FORM sub1.
  IF f1 NE 0.
    ...
  ENDIF.
ENDFORM.

```

### 5.6.4 IF statements

When coding **IF** tests, nest the testing conditions so that the outer conditions are those that are most likely to fail. For logical expressions with **AND**, place the mostly likely false first and for the **OR**, place the mostly likely true first.

#### Example - nested IF's:

```

IF (least likely to be true).
  IF (less likely to be true).
    IF (most likely to be true).
      ENDIF.
    ENDIF.
  ENDIF.
ENDIF.

```

#### Example - IF...ELSEIF...ENDIF:

```

IF (most likely to be true).
  ELSEIF (less likely to be true).
  ELSEIF (least likely to be true).
ENDIF.

```

#### Example - AND:

```

IF (least likely to be true) AND
  (most likely to be true).
ENDIF.

```

#### Example - OR:

```

IF (most likely to be true) OR
  (least likely to be true).

```

### 5.6.5 CASE vs. nested IFs

When testing fields "equal to" something, one can use either the **nested IF** or the **CASE** statement. The **CASE** is better for two reasons. It is easier to read and after about five nested **IFs** the performance of the **CASE** is more efficient.

### 5.6.6 Forcing next iteration in a LOOP

When the next iteration in a loop has to be executed, use

```

LOOP AT GT_ITAB.
  ...
  CONTINUE.
  ...
ENDLOOP.
instead of
LOOP AT GT_ITAB.
  ...
  CHECK 1 EQ 2
  ...
ENDLOOP.

```

### 5.6.7 MOVE-ing structures

When records a and b have the exact same structure, it is more efficient to **MOVE a TO b** than to **MOVE-CORRESPONDING a TO b**.

```
MOVE BSEG TO *BSEG.
```

is better than

```
MOVE-CORRESPONDING BSEG TO *BSEG.
```

## 5.6.8 SELECT and SELECT SINGLE

When using the **SELECT** statement, study the key and always provide as much of the left-most part of the key as possible. If the entire key can be qualified, code a **SELECT SINGLE** not just a **SELECT**. If you are only interested in the first row or there is only one row to be returned, using **SELECT SINGLE** can increase performance by up to three times.

## 5.6.9 WHERE clause

Although more important in earlier releases of SAP, it is still considered good practice to specify the fields in the WHERE clause in the same order as they appear in the primary key or index of the table.

## 5.6.10 Small internal tables vs. complete internal tables

In general it is better to minimize the number of fields declared in an internal table. While it may be convenient to declare an internal table using the LIKE command, in most cases, programs will not use all fields in the SAP standard table. For example:

```

Instead of this:
DATA: T_VBAK LIKE VBAK OCCURS 0 WITH HEADER LINE.
USE THIS:
DATA: BEGIN OF T_VBAK OCCURS 0,
      VBELN LIKE VBAK-VBELN,
      ...
END OF T_VBAK.

```

if only a small number of fields from VBAK are actually required in the program.

## 5.6.11 Row-level processing of a table

Selecting data into an internal table using an array fetch versus a SELECT-ENDSELECT loop will give at least a 2x performance improvement. After the data has been put into the internal table, then row-level processing can be done.

For example, use:

```

SELECT ... FROM TABLE <...>
INTO <ITAB> (CORRESPONDING FIELDS OF ITAB)
WHERE ...
LOOP AT <ITAB>
<DO THE ROW-LEVEL PROCESSING HERE>
ENDLOOP.

```

Instead of

```

SELECT ... FROM TABLE <...
WHERE ...
<ROW-LEVEL PROCESSING>
APPEND <ITAB>.
ENDSELECT.

```

## 5.6.12 Row-level processing and SELECT SINGLE

Similar to the processing of a **SELECT-ENDSELECT** loop, when calling multiple **SELECT-SINGLE** commands on a non-buffered table (check Data Dictionary -> Technical Info), you should do the following to improve performance:

- ❖ Use the **SELECT into <itab>** to buffer the necessary rows in an internal table, then
- ❖ Sort the rows by the key fields, then

- ❖ Use a **READ TABLE WITH KEY ... BINARY SEARCH** in place of the **SELECT SINGLE** command. Note that this only make sense when the table you are buffering is not too large (this decision must be made on a case by case Architecture and Infrastructure team).

### 5.6.13 Reading single records of internal tables

When reading a single record in an internal table, the **READ TABLE WITH KEY** is not a direct **READ**. This means that if the data is not sorted according to the key, the system must sequentially read the table. Therefore, you should:

- ❖ **SORT** the table
- ❖ Use **READ TABLE WITH KEY BINARY SEARCH** for better performance.

### 5.6.14 Sorting internal tables

When Sorting internal tables, specify the fields to Sorted.

```
SORT ITAB BY FLD1 FLD2.
```

is more efficient than

```
SORT ITAB.
```

### 5.6.15 Number of entries in an internal table

To find out how many entries are in an internal table use DESCRIBE.

```
DESCRIBE TABLE ITAB LINES CNTLNS.
```

is more efficient than

```
LOOP AT ITAB.
  CNTLNS = CNTLNS + 1.
ENDLOOP.
```

### 5.6.16 Nested SELECTs versus table views/joins

Since release 4.0, **OPEN SQL** allows both inner and outer table joins. A nested **SELECT** loop may be used to accomplish the same concept. However, the performance of nested **SELECT** loops is very poor in comparison to a join. Hence, to improve performance by a factor of 25x and reduce network load, you should either create a view in the data dictionary then use this view to select data, or code the select using a join.

Example:

```
SELECT * INTO GT_WA
  FROM SPFLI AS P INNER JOIN SFLIGHT AS F
    ON P~CARRID = F~CARRID AND
      P~CONNID = F~CONNID.
ENDSELECT.
```

is more efficient than

```
SELECT * FROM SPFLI INTO GT_SPFLI_WA.
SELECT * FROM SFLIGHT INTO GT_SFLIGHT_WA
  WHERE CARRID = SPFLI_WA-CARRID
    AND CONNID = SPFLI_WA-CONNID.
ENDSELECT.
ENDSELECT.
```

Due to the way that SAP has implemented JOINS it is important to follow the following general rules:

1. Do not join more than 3 tables in on a select statement
2. Ensure you join using all possible primary keys
3. **Use of MANDT in Join clause: Oss note: 621640**

Ensure that there is a one to one relationship between the tables being joined.

### 5.6.17 If nested SELECTs must be used

As mentioned previously, performance can be dramatically improved by using views instead of nested **SELECT**s, however, if this is not possible, then the following example of using an internal table in a nested **SELECT** can also improve performance by a factor of 5x:

Use this:

```
FORM SELECT_GOOD.
  DATA: T_VBAK LIKE VBAK OCCURS 0 WITH HEADER LINE.
  DATA: T_VBAP LIKE VBAP OCCURS 0 WITH HEADER LINE.
  SELECT * FROM VBAK INTO TABLE T_VBAK UP TO 200 ROWS.
  SELECT * FROM VBAP
    FOR ALL ENTRIES IN T_VBAK
    WHERE VBELN = T_VBAK-VBELN.
  ...
ENDSELECT.
ENDFORM.
```

Instead of this:

```
FORM SELECT_BAD.
  SELECT * FROM VBAK UP TO 200 ROWS.
  SELECT * FROM VBAP WHERE VBELN = VBAK-VBELN.
  ...
ENDSELECT.
ENDSELECT.
ENDFORM.
```

Although using "**SELECT...FOR ALL ENTRIES IN...**" is generally very fast, you should be aware of the four pitfalls of using it:

- Firstly, SAP automatically removes any duplicates from the rest of the retrieved records. Therefore, if you wish to ensure that no qualifying records are discarded, the field list of the inner **SELECT** must be designed to ensure the retrieved records will contain no duplicates (normally, this would mean including in the list of retrieved fields all of those fields that comprise that table's primary key).
- Secondly, if you were able to code "**SELECT ... FROM <database table> FOR ALL ENTRIES IN TABLE <itab>**" and the internal table <itab> is empty, then all rows from <database table> will be retrieved.
- Thirdly, if the internal table supplying the selection criteria (i.e. internal table <itab> in the example "**...FOR ALL ENTRIES IN TABLE <itab>**") contains a large number of entries, performance degradation may occur.
- "**SELECT...FOR ALL ENTRIES IN...**" can lead to extremely long select statements, which can cause ABAP short dumps, due to limitations of the underlying database on the length of the select statement

### 5.6.18 ORDER BY clause

Do not use Order By clause on non-key or non-indexed fields. Bring the data into an internal table with a single call and then sort the internal table. This relieves the database of the additional sort requirement and utilizes the processing and memory of the application server.

### 5.6.19 HAVING clause

In a **SELECT** statement, the **HAVING** clause allows you to specify a logical condition for the groups in a **GROUP-BY** clause. Effective use of the having clause can reduce the set of data transferred from the database to the application server. When the having clause is used, the aggregates and groups are constructed in the database instead of the application server, thereby reducing the resulting set.

### 5.6.20 SELECT \* versus SELECT individual fields

In general, use a **SELECT** statement specifying a list of fields instead of a **SELECT \*** to reduce network traffic and improve performance. For tables with only a few fields the improvements may be minor, but many SAP tables contain more than 50 fields when the program needs only a few. In the latter case, the performance gains can be substantial. For example:

Use:

```
SELECT VBELN AUART VBTP FROM TABLE VBAK
      INTO (VBAK-VBELN, VBAK-AUART, VBAK-VBTP)
      WHERE ...
```

Instead of using:

```
SELECT * FROM VBAK WHERE ...
```

### 5.6.21 WHERE clause

Specifying values for as many of the table's key fields in a WHERE clause will make the SELECT statement more efficient than checking values after the select.

For example:

```
PARAMETERS: P_LANGU LIKE SY-LANGU.
...
SELECT *
FROM T005T
WHERE SPRAS EQ P_LANGU.
...
ENDSELECT.
is more efficient than:
PARAMETERS: P_LANGU LIKE SY-LANGU.
...
SELECT *
FROM T005T.
      CHECK T005T-SPRAS = P_LANGU.
...
ENDSELECT.
```

Specifying the 'left-most'/least specific key fields in a WHERE clause improves efficiency. For example, the key fields (in sequence) of the table KNC3 (Customer special G/L transaction)

```
MANDT - Client
KUNNR - Customer number
BUKRS - Company code
GJAHR - Fiscal year
SHBKZ - Special G/L indicator
```

When selecting data from this table, it would be more efficient to specify a value in the WHERE clause for the field KNC3-BUKRS, rather than KNC3-GJAHR. That is:

```
SELECT *
FROM KNC3
WHERE KUNNR EQ '0000000001'
AND BUKRS EQ 'US01'.
...
ENDSELECT.
```

will be more efficient than:

```
SELECT *
FROM KNC3
WHERE KUNNR EQ '0000000001'
AND GJAHR EQ '1996'.
...
ENDSELECT.
```

- ❖ You can specify as many WHERE conditions as you like in all types of database tables - i.e. transparent tables, pool tables and cluster tables. However, you should be aware for performance reasons that complex WHERE conditions involving pool and cluster tables usually cannot be passed to the database system. They must be processed by the SAP database interface through post-selection.
- ❖ When accessing pool and cluster tables, these should be accessed using the full primary key.
- ❖ Avoid placing a 'SELECT' or a 'SELECT SINGLE' in a loop to minimize the number of database requests.
- ❖ Avoid using 'SELECT.... Into corresponding field' as the associated overhead with corresponding field could be significant.

- ❖ Selecting via non-key fields. When selecting records from a database table when only part of a field (on which selection is based) is known, use the LIKE option as part of the WHERE clause.

For example:

```
SELECT      *
FROM        T001G
WHERE       BUKRS EQ    'US01'
AND         TXTKO LIKE  '___PERS%'.
          .
          .
          .
          .
ENDSELECT.
```

is more efficient than:

```
SELECT      *
FROM        T001G
WHERE       BUKRS EQ    'US01'.
CHECK      T001G-TXTKO+2(4) = 'PERS'.
```

### 5.6.22 Avoid unnecessary statements

There are a few cases where one command is better than two. For example:

Use:

```
APPEND <TAB_WA> TO <TAB>.
```

Instead of:

```
<TAB> = <TAB_WA>.
APPEND <TAB> (MODIFY <TAB>).
```

And also, use:

```
IF NOT <TAB>[] IS INITIAL.
```

Instead of:

```
DESCRIBE TABLE <TAB> LINES <LINE_COUNTER>.
IF <LINE_COUNTER> > 0.
```

### 5.6.23 Copying or appending internal tables

Use this:

```
<TAB2>[] = <TAB1>[]. (IF <TAB2> IS EMPTY)
```

Instead of this:

```
LOOP AT <TAB1>.
  APPEND <TAB1> TO <TAB2>.
ENDLOOP.
```

However, if <tab2> is not empty and should not be overwritten, then use:

```
APPEND LINES OF <TAB1> [FROM INDEX1] [TO INDEX2] TO <TAB2>.
```

### 5.6.24 Declaring internal tables using R/3 Release 4.x syntax

See also 4.1.10 Using Internal Tables.

R/3 release 4.6 contains capabilities for hashing and sorting internal tables that improve system performance. The 4.6 syntax should be used whenever coding internal tables. The following examples illustrate the use of this syntax.

- ❖ First, declare the header line or work area (wa) as a type:

```
TYPES: BEGIN OF <LINE_TYPE>,
        FIELD1 LIKE ...,
        .
        .
        .
        FIELDN LIKE ..,
END OF <LINE_TYPE>.
```

- ❖ For example, here is a table line type with some vendor fields

```
TYPES: BEGIN OF TY_VENDOR_DATA,
        LIFNR TYPE LFA1-LIFNR,
```

```

NAME1 TYPE LFA1-NAME1,
END OF TY_VENDOR_DATA.

```

or

```
TYPES: <LINE_TYPE> LIKE <DD TABLE OR DDSTRUCTURE>.
```

- ❖ This version would include ALL of the vendor (LFA1) fields

```
TYPES: TY_VENDOR_DATA TYPE LFA1.
```

You may then declare the internal table (see Below) or, optionally, you may first declare a table type. Declaring a table type would be preferable in those cases where your program declares several internal tables of the same type (and therefore the table type can be reused by each internal table declaration) or where the internal table is passed to subroutines or methods (and therefore the table type can be included in the form/method interface definition).

```
TYPES: <T_TABLE_TYPE> TYPE <LINE_TYPE> OCCURS 0.
```

- ❖ This example would define a standard (i.e. unsorted) table type

```
TYPES: TY_T_VENDOR TYPE TY_VENDOR_DATA OCCURS 0.
```

or

```
TYPES: <T_TABLE_TYPE> TYPE
[ SORTED | HASHED | STANDARD | INDEX | ANY ]
TABLE OF <LINE_TYPE>
WITH [ UNIQUE | NON-UNIQUE ] KEY <KEY DEFINITION>].
```

- ❖ This example would define a vendor table type sorted by vendor number

```
TYPES: TY_T_VENDOR TYPE SORTED TABLE OF TY_VENDOR_DATA
WITH UNIQUE KEY LIFNR.
```

- ❖ Finally declare the table and the work area:

```
DATA:<T_TABLE> TYPE <T_TABLE_TYPE>,"ITAB W/ NO HEADER LINE
DATA:<WA_TABLE>TYPE <LINE_TYPE>. " WORK AREA FOR T_TABLE
```

- ❖ This example defines an itab using the table type we just defined above

Note: depending on which of the two above table types you choose this will define either a standard or sorted internal table.

```
DATA: T_VENDOR TYPE TY_T_VENDOR, " ITAB HAS NO HEADER LINE
WA_VENDOR TYPE TY_VENDOR_DATA." WORK AREA FOR T_VENDOR
```

- ❖ declaration of an internal table using a table line type

```
DATA: <T_TABLE> TYPE
[ SORTED | HASHED | STANDARD | INDEX | ANY ] TABLE
OF <LINE_TYPE>
WITH [ UNIQUE | NON-UNIQUE ] KEY <KEY DEFINITION>]
[WITH HEADER_LINE].
```

- ❖ This example declares a standard, unsorted internal table

```
DATA: T_VENDOR TYPE TY_VENDOR_DATA OCCURS 0 WITH HEADER LINE.
```

- ❖ This example declares an internal table sorted by vendor number

```
DATA: T_VENDOR TYPE SORTED TABLE OF TY_VENDOR_DATA
WITH UNIQUE KEY LIFNR.
WITH HEADER LINE.
```

## 6 Naming conventions

### 6.1.1 Transport requests

ARCHITECTURE AND INFRASTRUCTURE TEAM would publish a document specifying the details of transport request naming. Addition constraints for transport requests will be: ARCHITECTURE AND INFRASTRUCTURE TEAM NAMING + Development spec ID + free text

e.g. R1GTRSYSTCI – 0012 – “Free text”

### 6.1.2 Package

Package concept is an extension of development class that helps developers to modularise, encapsulate, and decouple units in SAP system. The following attributes distinguishes package from development class:

- 1) Package can define visibility of package elements through package interfaces
- 2) Package can restrict the usage of package elements through use access

Packages can be nested

Please refer to Appendix L – Packages for list of packages used in HydroOne.

For more information about package concept, please refer to the SAP help URL below:

[http://help.sap.com/saphelp\\_nw70/helpdata/EN/af/40bd38652c8c42e10000009b38f8cf/frameset.htm](http://help.sap.com/saphelp_nw70/helpdata/EN/af/40bd38652c8c42e10000009b38f8cf/frameset.htm)

### 6.1.3 Technical Specification Naming Convention

Program Type ID	Description
TBD	Data Conversion / Migration
E	Enhancements
I	Interfaces
R	Reports
W	Workflow
F	Forms

**TS\_<Program Type ID>\_<Application Area>\_<APSE ID>\_<Title> <Version>**

**Example:**

**For Function Spec FS\_E\_007\_Convert\_Legacy\_Acct\_To\_CA v1.0**

**The Technical Specification name would become**

TS\_E\_FI\_007\_Convert\_Legacy\_Acct\_To\_CA v1.0

Where –

- E = Technical Requirements ID (Reports). *See Table Above*
- FI = (FI for Contract accounts receivable and payable). *See list at the end of this document*
- 007 = APSE ID – A 3-digit numbers follow APSE development specification number.
- Title = The title of the document
- v1.0 = Document versioning; always started with 1.0 for initial documents version, increased by 1.0 for subsequent version, e.g v2.0.



### 6.1.4 Variable naming and declaration

ABAP/4 variable names can be up to 30 characters for DATA fields and subroutines and up to 8 characters for **SELECT-OPTIONS** and **PARAMETERS**, therefore, as a standard make the names descriptive. Since SAP segment/table-field names are hyphenated with a hyphen ('-'), use an underscore ('\_') to separate the words for program-specific variables. Whenever possible, the LIKE parameter should be used to define work fields.

Some variable types should be prefixed with a specified letter or letters (a distinction is made between global and local variables):

❖ Global variables:

Variable Type	Prefix
Selection screen parameter	p_
Select-options	s
Ranges	gr
Internal tables	gt_
Constants	gc_
Structure / Work Area	gs_
Work field / Variable	gw_
Field symbols	gfs_
Types	gty_

❖ Local variables:

Variable Type	Prefix
Form routine parameter	lp
Ranges	lr
Internal tables	let
Constants	lc
Work field	lw
Field symbols	lfs_
Types	lty_

❖ ABAP Objects

Variable Type	Prefix
ABAP object class	lcl_
Global ABAP object	g_
Local ABAP object	l_

When declaring internal variables, use the **LIKE** option only for program internal objects. Refer to ABAP Dictionary Objects using **TYPE** since these are data types. Use the reference objects in the ABAP Dictionary whenever possible to ensure that the variable has the same field attributes as the existing object, which has already been declared. If the type of field, to which you are referring, changes the ABAP runtime system updates all references automatically. It also stops the system from carrying out unnecessary (and maybe undesirable) type conversions.

Example: `DATA: LIFNR TYPE LFA1-LIFNR.`

## 6.2 ABAP development

### 6.2.1 ABAP Programs

The maximum length of an ABAP program name (SE38) is 30 characters.

<b>Format:</b>		<b>Z</b>	<b>AAAA</b>	<b>XXXXXXXXXXXXXXXXXXXXXXXXXXXX</b>
with	2 3 4	A	Application Area (see Appendix F)	
	5	T	Program type	
		D	Data migration	
		E	Enhancement	
		I	Interface	
		R	Report	
	6 30	X	Meaningful description	

### 6.2.2 Module Pools & Includes

Module pools should be named as follows:

<b>Format:</b>		<b>SAPM Z</b>	<b>AAAA</b>	<b>XXXXXXXXXXXXXXXXXXXXXXXXXXXX</b>
with	1 4	SAPM	required by SAP to denote online modules	
	5	Z	Z to denote customer named object	
	6 7 8	A	Application Area (see Appendix A)	
	9 30	X	Meaningful description	

Module pool includes should be named as follows:

<b>Format:</b>		<b>M Z</b>	<b>AAAA</b>	<b>XXXXXXXXXXXXXXXXXXXXXXXXXXXX</b>	<b>T NN</b>
With	1	M	required by SAP to denote online modules		
	2	Z	Z to denote customer named object		
	3 4	A	Application Area (see Appendix A)		
	5 27	X	Meaningful description corresponding to main program		
	28	T	I	PAI modules	
			O	PBO modules	
			F	subroutines	
	29 30	N	Sequential number (00 to 99)		

For the global data include, T and N are replaced by 'TOP'.

### 6.2.3 Transaction Codes

All SAP transactions have unique transaction codes, which are listed in tables TSTC and TSTCP. Transactions are maintained via transaction SE93. Transaction codes consist of up to twenty characters.

**Format:**            **Z AAAA XXXXXXXXXXXXXXXXXXXX**

with:                2 3 4    A        Application Area (see Appendix F)  
                       5 20    X        Meaningful description

### 6.2.4 Dynpros (Screens)

SAP screens are referred to as "Dynpros". Standard SAP components, such as transactions, menus and tables, contain Dynpros and the associated processing logic. The user can also generate customised Dynpros. (SE51 - Screen painter)

The identification of a Screen Painter Dynpro consists of an ABAP program name (30 characters) and a four-digit screen number.

**Format:**            **AA NNNN**

with:                1 30    A        Associated ABAP program name (See naming convention for ABAP programs above)  
                       31 34   N        Screen number (number in intervals of ten, starting at 9000)

### 6.2.5 Dialog Modules

The maximum length of a dialog module name (SE35) is 30 characters.

**Format**                **Z AAAA XXXXXXXXXXXXXXXXXXXXXXXXXXXX**

with:                2 3 4    A        Application Area (see Appendix F)  
                       5 30    X        Meaningful description

### 6.2.6 Messages

Messages (SE91) are referenced in ABAP programs by a twenty-character message class and a 3-character message ID (preceded by the message type identifier).

**Format:**            **Z AAAA XXXXXXXXXXXXXXXXXXXX NNN**

with:                2 3 4    A        Application Area (see Appendix F)  
                       5 20    X        Meaningful description  
                       21 23   N        Message number (sequentially assigned number)

**NB:**        The message number '000' should not be used.

### 6.2.7 Function Groups

Function group names are twenty-six characters in length. One function group should be created per table view maintenance screen, to ease potential transport problems.

**Format:**            **Z AAAA XXXXXXXXXXXXXXXXXXXXXXXX**

with:                2 3 4    A        Application Area (see Appendix F)  
                       5 26    X        Meaningful description

### 6.2.8 Function Modules

The maximum length of a function module name (SE37) is 30 characters.

**Format**                **Z\_ AAA\_ XXXXXXXXXXXXXXXXXXXXXXXXXXXX**

with:                3 4 5    A        Application Area (see Appendix F)  
                       6 30    X        Meaningful description

### 6.2.9 Logical Databases

A logical database creates a logical hierarchical view of several physical relational tables, giving the ability to link database tables, thus simplifying report programming through standardised read access, authorisation checks and selections.

Logical database names consist of twenty characters.

**Format:**            **Z AAA XXXXXXXXXXXXXXXXXXXX**  
with:                 2 3 4    A        Application Area (see Appendix F)  
                          5 20    XX        Meaningful description

### 6.2.10 Business Add-Ins

Business Add-Ins are a new SAP enhancement technique based on ABAP Objects. They can be inserted into the SAP System to accommodate user requirements too specific to be included in the standard delivery

BADI implementation names consist of twenty characters.

**Format:**            **Z AAA XXXXXXXXXXXXXXXXXXXX**  
with:                 2 3 4    A        Application Area (see Appendix F)  
                          5 20    XX        Meaningful description (ref. the BADI definition if possible)

## 6.3 ABAP Objects Programming

Note that we don't usually include a scope element in the name of the objects described in this section. This is for a number of reasons:

- ❖ The scope of a class component is explicitly defined as part of the declaration of the method – there is no need to specify it in the name,
- ❖ The traditional scope 'buckets' (global, local, etc.) don't map well to the OO paradigm,
- ❖ OO object and component names are supposed to be fixed and relevant to their responsibility – it would be absurd to have to rename a method because its scope changed if its functionality remained the same.

### 6.3.1 Classes (defined in DDIC or programmatically)

A class is a template for an encapsulated piece of data and functionality that represents a physical or logical object. It could be described as an object's type.

The maximum length of a class name is 30 characters. The name may consist of alphanumeric characters plus the special characters underscore ( \_ ) and forward slash ( / ). The forward slash ( / ) is used as a delimiter for the namespace prefix. The name may not begin with a digit.

**Format:**            **ZCL\_AAA\_XXXXXXXXXXXXXXXXXXXXXXXXX**  
with:                 5 6 7    A        Application area  
                          9 30    X        Meaningful description.

NB: Strictly, the descriptive part of a class name should be a noun. Note that SAP rarely follows this OO naming convention, since they have restricted the length of the class name in the DDIC.

### 6.3.2 Interfaces (defined in DDIC or programmatically)

The components of a class are divided into visibility sections, and this forms the external point of contact between the class and its users. For example, the components of the public section form the public interface of the class, since any user can access all attributes and method parameters. The protected components form an interface between the class and those classes that inherit from it (subclasses). Interfaces are independent structures that allow you to enhance the class-specific public points of contact by implementing them in classes. Different

classes that implement the same interface can all be addressed in the same way. Interfaces are the basis for polymorphism in classes, because they allow a single interface method to behave differently in different classes. Interface references allow users to address different classes in the same manner. Interfaces can also be nested.

The maximum length of an interface name is 30 characters. The name may consist of alphanumeric characters plus the special characters underscore ( \_ ) and forward slash ( / ). The forward slash ( / ) is used as a delimiter for the namespace prefix. The name may not begin with a digit.

**Format:**                    ZIF\_XXXXXXXXXXXXXXXXXXXXXXXXXXXX

with:    2 30    X            Meaningful description.

NB: Strictly, the descriptive part of an interface name should be an adjective that describes the ability that a class will gain if it properly implements the interface. E.g. handleable, deletable. Note that SAP rarely follows this OO naming convention, since they have restricted the length of the interface name in the DDIC.

### 6.3.3 Methods (defined in DDIC or programmatically)

Methods are the internal procedures of a class that determine the behaviour of an object. They can access all the attributes of their class and can thus change the object status. Methods have a parameter interface, through which the system passes values to them when they are called, and through which they can return values to the caller. The private attributes of a class can only be changed using methods. In terms of definition and passing parameters, methods are similar to the old function modules.

**Format:**                    XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

with:    1 30    X            Meaningful name.

NB: Method names should principally be verbs describing what will happen when the method is invoked. E.g. handle\_event, destroy, get\_page\_url. Separate words with underscores, since SAP is not case sensitive here.

### 6.3.4 Attributes (defined in DDIC or programmatically)

Attributes are internal data fields of any ABAP data type within a class. The content of the attributes specifies the status of the object. You can also define reference variables, which you can then use to create and address objects. This allows objects to be accessed within classes.

**Format:**                    XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

with:    1 30    X            Meaningful name.

### 6.3.5 Events (defined in DDIC or programmatically)

Events allow an object or class to trigger event handler methods in another class or object. Just as, in a normal method call, a method can be called by any number of users, any number of event handler methods can be called when an event is triggered. The trigger and handler are coupled at runtime. In a normal method call, the caller specifies which method it wants to call; the relevant method must be available. With events, the handler specifies the events to which it wants to react; a handler method must be registered for each event.

**Format:**                    XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

with:    1 30    X            Meaningful name.

NB: Event names should principally be past tense verbs describing what happened to cause the event to be raised. E.g. button\_clicked, window\_closed. Separate words with underscores, since SAP is not case sensitive here.

### 6.3.6 Internal Types (defined in DDIC or programmatically)

Types define your own ABAP data types within a class. Use the **TYPES** statement to declare constants Types are not instance- specific - they are available once and once only for all the objects in the class.

**Format:**                    TY\_XXXXXXXXXXXXXXXXXXXXXXXXXXXX

with:    4 30    X            Meaningful name.

### 6.3.7 Constants (defined in DDIC or programmatically)

Constants are special static attributes, whose values are specified when they are declared and which cannot be changed later. Use the **CONSTANTS** statement to declare constants. Constants are not instance-specific - they are available once and once only for all the objects in the class.

**Format:** C\_XXXXXXXXXXXXXXXXXXXXXXXXXXXXX  
 with: 3 30 X Meaningful name.

## 6.4 Data Dictionary

### 6.4.1 Domains

A domain is a central object for describing the attributes of a business object. A domain describes the physical value set for a field. This set of values is defined by specifying formal attributes, such as external format, length, etc., which are entered as individual values or in the form of a table.

The maximum length of a domain name is 30 characters.

**Format:** ZDOM\_XXXXXXXXXXXXXXXXXXXXXXXXXXXXX  
 with: 2 30 X Meaningful description

### 6.4.2 Data Elements

A data element is a semantic domain. It gives a precise description of the function of a domain in a specific business context for the benefit of the fields dependent on it.

The maximum length of a data element name is 30 characters.

**Format:** ZDE\_XXXXXXXXXXXXXXXXXXXXXXXXXXXXX  
 with: 2 30 X Meaningful description

### 6.4.3 Tables

The following section describes the requirements for the different types of tables and fields within them.

Data (transparent) Tables:

The maximum length of a standard SAP table name is 16 characters.

**Format:** ZTBL\_XXXXXXXXXXXXX  
 with: 5 16 X Meaningful description

Table Fields:

The maximum length of a table field is 16 characters.

**Format:** XXXXXXXXXXXXXXXX  
 with: 1 16 X Meaningful description

This applies to all table fields in **custom/bespoke** tables.

### 6.4.4 Append Structures

Append structures are used for adding customer fields to the standard SAP tables, and protect these additional fields from being overwritten by SAP upgrades. Whenever possible, an existing SAP append structure should be extended. If the table has no appends, the system proposes a standard name. This, too, should be used whenever possible. However, if a custom name is to be created, the following conventions apply.

The maximum length of the name of an append structure is thirty characters. It has to be created in the customer name range, even though it is an extension to an original SAP table.

**Format:** ZXXXXXXXXXXXXXXXXXXXXXXXXXXXXX  
 with: 2 30 X Name of the table to which the structure is appended

plus a numeric identifier

Fields in an Append Structure:

The maximum length of an append structure field is thirty characters.

**Format:**            **Z XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX**

with:    2 30    X            Meaningful description

### 6.4.5 Views

A view is a virtual table (not containing any data). It presents data from one or more related tables in a tabular form appropriate for application processing.

The maximum length of a view name is 16 characters.

**Format:**            **Z AAA XXXXXXXXXXXXXXX**

with:    2 3 4    A            Application Area (see Appendix F)

          5 16    X            Meaningful alphanumeric identifier

View Fields:

The maximum length of a view field is 10 characters.

**Format:**            **XXXXXXXXXX**

with:    1 10    X            Meaningful description

### 6.4.6 Structures

Structures are global definitions of the structure of data used during calculation operations within programs, or when passing data between programs.

The maximum length of a structure name is 30 characters.

**Format:**            **ZSTR\_XXXXXXXXXXXXXXXXXXXXXXXXXXXX**

with:    2 3 4    A            Application Area (see Appendix F)

          5 30    X            Meaningful description

Structure Fields:

The maximum length of a structure field is 30 characters.

**Format:**            **XXXXXXXXXXXXXXXXXXXXXXXXXXXX**

with:    1 30    X            Meaningful description

### 6.4.7 Search Helps (Matchcodes)

A search help (matchcode before Release 4.0) is a search criterion enabling efficient access to records without requiring knowledge of the key term.

Collective Search Help

Collective search helps (equivalent to release 3.x matchcode objects) have a maximum of thirty characters, with the first two characters being the Entity Code, and the remaining characters meaningfully assigned.

**Format:**            **ZSH\_XXXXXXXXXXXXXXXXXXXXXXXXXXXX**

with:    2 3 4    A            Application Area (see Appendix F)

          5 30    X            Meaningful description

Elementary Search Help

All data access is done via an elementary search help (equivalent to a release 3.x matchcode ID). A number of elementary search helps can be assigned to one collective search help.

The name of an elementary search help can be up to thirty characters in length.

**Format:**            **ZSH\_XXXXXXXXXXXXXXXXXXXXXXXXXXXX**

with: 2 3 4 A Application Area (see Appendix F)  
 5 30 X Meaningful description

### 6.4.8 Lock Objects

Lock objects are used to lock dictionary objects during a logical unit of work, e.g. until the next COMMIT is reached. The maximum length of a lock object name is 16 characters.

Format: E Z AAA XXXXXXXXXXXXXXXX

with: 1 E Constant 'E'  
 3 4 A Application Area (see Appendix F)  
 5 16 X Meaningful description

## 6.5 ALE/IDOC Developments

### 6.5.1 IDOC Segment Types (version-independent)

Format: Z AAA XXXXXXXXXXXXXXXXXXXXXXXXXXXX I

with: 2 3 4 A Application Area (see Appendix F)  
 5 26 X Meaningful description  
 27 I 0-9, sequential number to avoid conflict

### 6.5.2 Basic IDOC Types

Format: Z AAA XXXXXXXXXXXXXXXXXXXXXXXXXXXX II

with: 2 3 4 A Application Area (see Appendix F)  
 5 28 X Meaningful description  
 29 30 I 01-99, sequential number to avoid conflict

### 6.5.3 Extension Types

Format: Z AAA XXXXXXXXXXXXXXXXXXXXXXXXXXXX II

with: 2 3 4 A Application Area (see Appendix F)  
 5 28 X descriptions of the customized IDOC being extended  
 29 30 I 01-99, sequential number to avoid conflict

### 6.5.4 Message Types

Format: Z AAA XXXXXXXXXXXXXXXXXXXXXXXXXXXX

with: 2 3 4 A Application Area (see Appendix F)  
 5 30 X meaningful description, relating to the IDOC this message type is for

### 6.5.5 Process Codes

Format: Z AAA XXXXXXXXXXXXXXXXXXXXXXXXXXXX

with: 2 3 4 A Application Area (see Appendix F)  
 5 30 X meaningful description, relating to the IDOC this process code is for

### 6.5.6 Inbound ALE Function Modules

Format: ZIDOC\_INPUT\_XXXXXXXXXXXXXXXXXXXX

with: 13 30 X message type description



### 6.5.7 Outbound ALE Function Modules

Format: ZIDOC\_OUTPUT\_XXXXXXXXXXXXXXXXXXXX  
 with: 14 30 X message type description

## 6.6 Business Object and BAPI developments

### 6.6.1 Business Object Type (within the BOR)

Format: Z AAA XXXXXX I  
 with: 2 3 4 A Application Area (see Appendix F)  
 5 9 X meaningful description  
 10 I 0-9, sequential number to avoid conflict

### 6.6.2 Business Object Method

Format: xxxxxxxxxxxxxxxxxxxxxxxx  
 with: 1 20 x short description, case sensitive (e.g. Change Password)

### 6.6.3 BAPI based function module

Format: ZBAPI\_XXXXXXXX\_YYYYYYYYYYYY  
 with: 7 16 X Business Object Name  
 17 30 Y Method Name

## 6.7 Workflow Developments

The following is a list of general rules for Workflow object component names:

- Short Description should use uppercase letters on the first letter of each abbreviated word and lower case letters for the remainder of each abbreviated word. E.g. ChgMstr (with no spaces between abbreviated words)
- Underscores should be avoided if possible, and uppercase/lowercase used instead (within a work as well) to improve readability.
- Use of upper case/lower case does not affect identification; in other words, object component names are not case sensitive.
- Names should be nouns and begin with an uppercase letter.

SAP Object	Len	Naming Conventions
Object Type	10	<p><b>ZWO + ShortDesc</b></p> <p><b>O</b> Object</p> <p><b>ShortDesc</b> Short description of object name (7 chars)</p> <p>If an object type is defined as a <b>business object (BUSnnnn)</b>, the same name as that of the assigned data model should be chosen</p> <p>If <b>not a business object</b>, the 7 characters should use the following format:</p> <p>e.g. ZWOAAAAnnnnn</p> <p><b>AAA</b> Application area</p> <p><b>NNNNNN</b> Sequential number</p>

		e.g. Purchase Requisition - ZWOBUS2009
Object Type – Interfaces	10	<b>ZIF + ShortDesc</b> <b>IF</b> Interface <b>ShortDesc</b> Short description (6 chars)  e.g. ZIFSAP
Object name	32	<b>Z + ShortDesc + I</b> <b>ShortDesc</b> Short description (up to 30 chars) I 0 – 9 sequential number
Object - Key Fields	32	<b>K + ShortDesc</b> <b>K</b> Key Fields <b>ShortDesc</b> Short description (31 chars) e.g. KPurchReqNo
Object – Attributes	32	<b>AAAb + ShortDesc</b> <b>A</b> Attribute <b>a</b> D <input type="checkbox"/> Database field V <input type="checkbox"/> Virtual field <b>b</b> S <input type="checkbox"/> Single line M <input type="checkbox"/> Multi-line I <input type="checkbox"/> Instance independent O <input type="checkbox"/> Only modelled <b>ShortDesc</b> Short description (29 chars) e.g. ADSCostCentre
Object – Methods	32	<b>Mabc + ShortDesc</b> <b>M</b> Method <b>a</b> S <input type="checkbox"/> Dialog synchronous A <input type="checkbox"/> Dialog asynchronous B <input type="checkbox"/> Background <b>b</b> F <input type="checkbox"/> Function module R <input type="checkbox"/> ABAP report T <input type="checkbox"/> SAP transaction D <input type="checkbox"/> Dialog method B <input type="checkbox"/> API method O <input type="checkbox"/> Other <b>c</b> P <input type="checkbox"/> Parameter(s) N <input type="checkbox"/> No parameter(s) <b>ShortDesc</b> Short description (28 chars) e.g. MSTNCreatePurchReq

Object - Method - Function Modules	30	<b>ZWF_ + ShortDesc</b> <b>ShortDesc</b> Short description of method (19 chars)
Object – Method - BAPI Function	30	<b>ZBAPI_&lt;object type&gt;_&lt;method&gt;</b> <b>&lt;object type&gt;</b> Object type of this BAPI in BOR <b>&lt;method&gt;</b> Method name of this BAPI in BOR
Object - Method - ABAP/4 Reports	30	<b>ZWR + ShortDesc</b> <b>W</b> Workflow <b>R</b> ABAP Report <b>ShortDesc</b> Short description (20 chars)
Object - Method - Dialog Modules - ABAP/4	30	<b>ZWD + ShortDesc</b> <b>W</b> Workflow <b>D</b> Dialog Module - ABAP <b>ShortDesc</b> Short description (20 chars)
Object – Events	32	<b>WE + ShortDesc</b> <b>W</b> Workflow <b>E</b> Event <b>ShortDesc</b> Short description of event (30 chars) e.g. WEPurchReqApproved
Object - Implementation Program	30	<b>ZW + &lt;object type&gt;</b>
Standard Tasks	12	<b>ZWTSa+ ShortDesc</b> <b>TS</b> Standard task (single step) <b>a</b> D <input type="checkbox"/> Dialog B <input type="checkbox"/> Background <b>ShortDesc</b> Short description (7 chars) e.g. ZWTSDApprReq
Workflow Templates	12	<b>ZWWSa + ShortDesc</b> <b>WS</b> Workflow template (multi-step) <b>a</b> SAP Module <b>ShortDesc</b> Short description (7 chars) e.g. ZWWSMReqRel
Container Elements (field names)	32	<b>Cab + ShortDesc</b> <b>C</b> Container Element <b>a</b> W <input type="checkbox"/> Workflow tasks & templates T <input type="checkbox"/> Standard/Customer task R <input type="checkbox"/> Role resolution <b>b</b> O <input type="checkbox"/> Object type

		<p>D <input type="checkbox"/> DDIC field</p> <p><b>ShortDesc</b> Short description (29 chars)</p> <p>e.g. CTDPurchaseReqNo</p>
Standard Roles	12	<p><b>ZWRa + ShortDesc</b></p> <p><b>R</b> Standard Role</p> <p><b>a</b> <input type="checkbox"/> Function module</p> <p><input type="checkbox"/> Organizational object</p> <p><input type="checkbox"/> Responsibilities</p> <p><b>ShortDesc</b> Short description (8 chars)</p> <p>e.g.. ZWRF_REQUEST</p>
Standard Roles: Function module	30	<p><b>ZWF_ROLE + ShortDesc</b></p> <p><b>ShortDesc</b> Short description (15 chars)</p> <p>e.g. ZWF_ROLEGetRequisition</p>
Organisation Unit	12	<p><b>ZWUO + ShortDesc</b></p> <p><b>U</b> Organisational Structure</p> <p><b>O</b> Organization Unit</p> <p><b>ShortDesc</b> Short description (8 chars)</p> <p>e.g. ZWUOPurch</p> <p>Org. unit is needed if not using HR. If HR is being implemented, all PD org. objects and their names should be coordinated with the HR team</p>
Org Units – Jobs	12	<p><b>ZWUP + ShortDesc</b></p> <p><b>U</b> Organizational Structure</p> <p><b>P</b> Job</p> <p><b>ShortDesc</b> Short description (8 chars)</p> <p>e.g. ZWUPBuyers</p>
Org Units - Positions	12	<p><b>ZWUS + ShortDesc</b></p> <p><b>U</b> Organizational Structure</p> <p><b>S</b> Position</p> <p><b>ShortDesc</b> Short description (8 chars)</p> <p>e.g. ZWUSRawMatrl</p>

## 6.8 Print Workbench and Layout Sets

### 6.8.1 Application From

Format: ZPWB\_AA\_XXXXXXXXXXXXXXXXXXXXX  
 with: 6 7 A Application Area (see Appendix F)  
 9 30 X Meaningful description

## 6.8.2 Layout Sets

**Format:** ZSCR\_AA\_XXXXXXXXXXXXXXXXXXXXX  
 with: 6 7 A Application Area (see Appendix F)  
 9 30 X Meaningful description

## 6.8.3 Smartforms

**Format:** ZSMF\_AA\_XXXXXXXXXXXXXXXXXXXXX  
 with: 6 7 A Application Area (see Appendix F)  
 9 30 X Meaningful description

## 6.9 ITS developments

These naming conventions apply to developments on the Internet Transaction Server.

### 6.9.1 Internet Services

**Format:** Z AAA XXXXXXXXXXXXX  
 with: 2 3 4 A Application Area (see Appendix F)  
 5 14 X Meaningful description

### 6.9.2 MiniApps

**Format:** Z AAA XXXXXXXXXXXXX  
 with: 2 3 4 A Application Area (see Appendix F)  
 5 14 X Meaningful description, same text as the Internet service

### 6.9.3 Theme

**Format:** XX  
 with: 1 2 X 00-99, use 99 wherever possible

### 6.9.4 Template

Length is 40 characters and no naming conventions exist. Use a meaningful name.

## 6.10 Business Server Page (BSP) Applications

### 6.10.1 BSP Application Object

**Format:** Z\_AAA\_XXXXXXXXXXXXXXXXXXXXX  
 with: 3 4 5 A Application Area (see Appendix F)  
 6 30 X Meaningful description

### 6.10.2 Controller

**Format:** XXXXXXXXXXXXXXXXXXXX.do

with: 1 67 Xx Meaningful description (Case sensitive, so use keyword capitalisation - no underscores)

### 6.10.3 BSP page (View, Page with Flow Logic, or Page Fragment)

**Format:** XXXXXXXXXXXX.htm or XXXXXXXXXXXX.bsp

with: 1 67 Xx Meaningful description (Case sensitive, so use keyword capitalisation - no underscores)

### 6.10.4 Page attribute

**Format:** XXXXXXXXXXXX.htm or XXXXXXXXXXXX.bsp

with: 1 67 Xx Meaningful description (Case sensitive, so use keyword capitalisation - no underscores)

### 6.10.5 MIME object

MIME objects are usually imported, so the suffix will follow the convention of the imported object.

## 6.11 SAPSCRIPT Development

### 6.11.1 Layout Sets

Layout sets are SAP output forms that are defined using SAPSCRIPT. (SE71)

The layout set name can be a maximum of sixteen characters.

**Format:** Z AAA XXXXXXXXXXXXXXXX

with: 2 3 4 A Application Area (see Appendix F)  
5 16 X Meaningful description

### 6.11.2 Styles

Styles are character and paragraph definitions to be used in layout sets. (SE72)

**Format:** Z XXXXXXXX

with: 2 8 X Meaningful description

### 6.11.3 Standard Texts

Standard texts are SAP text modules to be used in various documents. (SO10)

They consist of a text name (up to thirty-two characters long) assigned to a text ID.

**Format (Text ID):** Z XXX

with: 2 4 X Meaningful alphanumeric identifier

**Format (Text name):** Z\_ AAA XX

with: 3 4 A Application Area (see Appendix F)  
4 32 X Meaningful description

## 6.12 Authorisations

### 6.12.1 Authorisation Objects

Authorisation objects may be created to apply additional security. An object has up to ten characters. The first should be a 'Z', the second an underscore and the remaining eight forms a meaningful description.

**Format:**            **Z\_XXXXXXXX**  
 with:     1 2     C        Constant 'Z\_'  
           3 10    X        Meaningful description

### 6.12.2 Authorisation Groups

Authorisation groups may be created to group together programs for authorisation checking. The authorisation group should be specified in the program attributes.

**Format:**            **Z AAA XXXXX**  
 with:     2 3 4    AAA     Application Area (see Appendix F)  
           5 8     X        Meaningful description

### 6.12.3 Authorisation Object Classes

Authorisation object classes group authorisation objects with common character. In standard SAP a functional module does this. A user-defined authorisation object class should be an extension of a delivered one - e.g. any additional FI authorisation objects would be placed in an additional FI authorisation object class. Therefore the 'extension' classes should be named in a similar way to the standard delivered classes.

**Format:**            **Z XXX**  
 with:     2 4     X        three letters of original SAP class (e.g. 'MMB')

**Remarks:**        SAP Security objects are hard-coded into the standard transactions. It is therefore very difficult to add security effectively. Careful analysis should be made before creating any new authorisation objects or classes.

## 6.13 Legacy System Migration Workbench (LSMW)

### 6.13.1 Projects

It is envisaged that one project will be created for each work stream.

**Format:**            **Z AAA XXXXXXXXXXXXX**  
 with:     2 3 4    A        Application Area (see Appendix F)  
           5 15    X        Meaningful description (e.g. MIGRATION)

### 6.13.2 Sub-projects

One sub-project will be created for each phase.

**Format:**            **Z AAA PHASE X**  
 with:     2 3 4    A        Application Area (see Appendix F)  
           9        X        Phase Number (i.e. 1, 2, 3 etc)

### 6.13.3 Object

The name of the data entity.

**Format:**            **Z XXXXXXXXXXXXXXX**  
 with:     2 15    X        Entity Name, followed by a sequence number (01, 02, etc)

The sequence number is there in the case where there are migration variations within one data entity.

## 6.14 ABAP Query

### 6.14.1 User Groups

The name of the user group.

**Format:**            **Z AAA XXXXXXXXXX**

with:    2 3 4    A        Application Area (see Appendix F)  
           5 12    X        Meaningful description

### 6.14.2 InfoSets

The name of the InfoSet

**Format:**            **Z AAA XXXXXXXXXXXXXXXXXXXXXXXX**

with:    2 3 4    A        Application Area (see Appendix F)  
           5 24    X        Meaningful description

### 6.14.3 Queries

The name of the Query

**Format:**            **Z AAA XXXXXXXXXX**

with:    2 3 4    A        Application Area (see Appendix F)  
           5 14    X        Meaningful description

### 6.14.4 Web Dynpro Application

This section outlines the naming convention for ABAP Web Dynpro developments

#### Web Dynpro Component

The maximum length of WD4A component name is 30 characters.

**Format:**            **Z AA XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX**

with        2 3        A        Business Area (see Appendix F)  
               4 30    X        Meaningful description

#### Web Dynpro Application

The maximum length of WD4A component name is 30 characters.

**Format:**            **Z AA XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX**

with        2 3        A        Business Area (see Appendix F)  
               4 30    X        Meaningful description



## 7 Documentation standards

ABAP/4 code is fairly self-documenting. However, it is wise to provide future programmers with documentation. Explain the purpose at the top of the program. Maintain a history of modification notes, dated, with the most recent change first. Comment work fields, and fields in work records especially those used for interfacing. Comment all subroutines with their purpose. Comments should explain what the code is doing, not how the code is doing it.

### 7.1 Header Documentation Box

The Header comment box similar to the one below should be coded after the REPORT, or after FUNCTION MODULE statement of customised ABAP/4 programs both for new and cloned programs. The documentation box must be used to list detailed information from functional and technical specifications as well as documenting modifications to the program.

A standard comment box can be created and maintained as a pattern via “Utilities□More Utilities□Mode□Create□Change pattern” in the ABAP/4 editor. Once set-up the comment box can be added to the code by entering `IC xxxx` in the command field of the ABAP/4 editor where xxxx is the comment box identifier.

```

*-----*
* Program Name   : ZMUKLOAD
* Title          : Upload Program for Material Data
* Create Date    : 28-Feb-2002
* Release        : 4.6C
* Author         : LROGERS
*-----*
* Description    : This program is used to upload the Material data from
*                  the UNIX file. It uses the BDC session method to upload
*                  the data.
*-----*
* CHANGE HISTORY
*-----*
*Date           | User ID      | Description                                     | Change Label *
*-----*-----*-----*-----*
* 8-Mar-2002   | PJOHNS      | Added a new fields to the                       | PJ01          *
*                  |              | report output.                                  |               *
*                  |              |                                                  |               *
*                  |              |                                                  |               *
*-----*-----*-----*-----*
    
```

After the program has been transported to production further changes to the code should be marked with a change-label. This can be of the form “XXXNN” where XXX are the initials and NN is a number. Coding lines to be deleted should be commented out and marked with the change-label and change label should be mentioned as reference in the change history in the header.

EXAMPLE.

```

..
..
WRITE: VBAK-VBELN,      "PJ01
       VBAK-VBELP.     "PJ01
..
    
```

After a period of time the commented out (deleted) lines should be removed from the coding to avoid the program having too many unnecessary lines and to avoid confusion.

### 7.2 Comment Box

A comment box is used to emphasise a statement or section in a program.

```

Example:
*-----*
* LOGIC SECTION
*-----*
    
```

A comment box should be inserted before all:

- declarative elements
- Events.

Example:

```
*-----*
* GLOBAL DATA *
*-----*
DATA: W_KUNNR,      "Customer nr
      W_MATNR.      "Material nr
*-----*
* TOP-OF-PAGE *
*-----*
TOP-OF-PAGE.

PERFORM WRITE_HEADER.
```

Other comment boxed which must be used are:

```
*-----*
* SELECT-OPTIONS / PARAMETERS *
*-----*
```

Before the declaration of parameters and select-options.

```
*-----*
* DATA DECLARATION *
*-----*
```

Before the data declaration part.

```
*-----*
* LOGIC SECTION *
*-----*
```

Before the start of the main program code. Mostly before START-OF-SELECTION.

### 7.3 Comment box before FORM/METHOD declaration

A declaration box before a FORM should be used. Comments and parameter explanation can be added here. Use the standard box as produced by SAP when double clicking on the PERFORM line to create the FORM.

Example:

```
*&-----*
*&      Form  WRITE_HEADER *
*&-----*
*      Write list header *
*-----*
* --> p_kunnr  Customer number *
* <-- rcode   return code *
*-----*
FORM WRITE_HEADER USING KUNNR
      CHANGING RC.

.....
ENDFORM.          " WRITE_HEADER
```

### 7.4 Comment lines

There are two ways of indicating comments in a report:

1. If the whole line is supposed to be a comment, enter an asterisk (\*) in the first column.

Example:

```
* Store Date in YYYYMMDD format in LONG DATE
MOVE: SY-DATUM+0(4) TO LONG_DATE-YEAR,
SY-DATUM+4(2) TO LONG_DATE-MONTH,
SY-DATUM+6(2) TO LONG_DATE-DAY.
```

Comment lines can be inserted anywhere in a report and so this should be used to describe the function of certain parts of the code.

2. If only the latter part of a line is supposed to be a comment, enter “ before the comment.

Example:

```
TABLES: TO01, "SAP Company Code Tables
         TO01C, "Cost Centre Accounting Control Table
         CSKS. "Cost Centre Master
```

The system interprets comments indicated by “ as blank characters. This option should be used to describe certain lines of code.

## 8 Special consideration

### 8.1 Internationalisation and Localisation Aspects

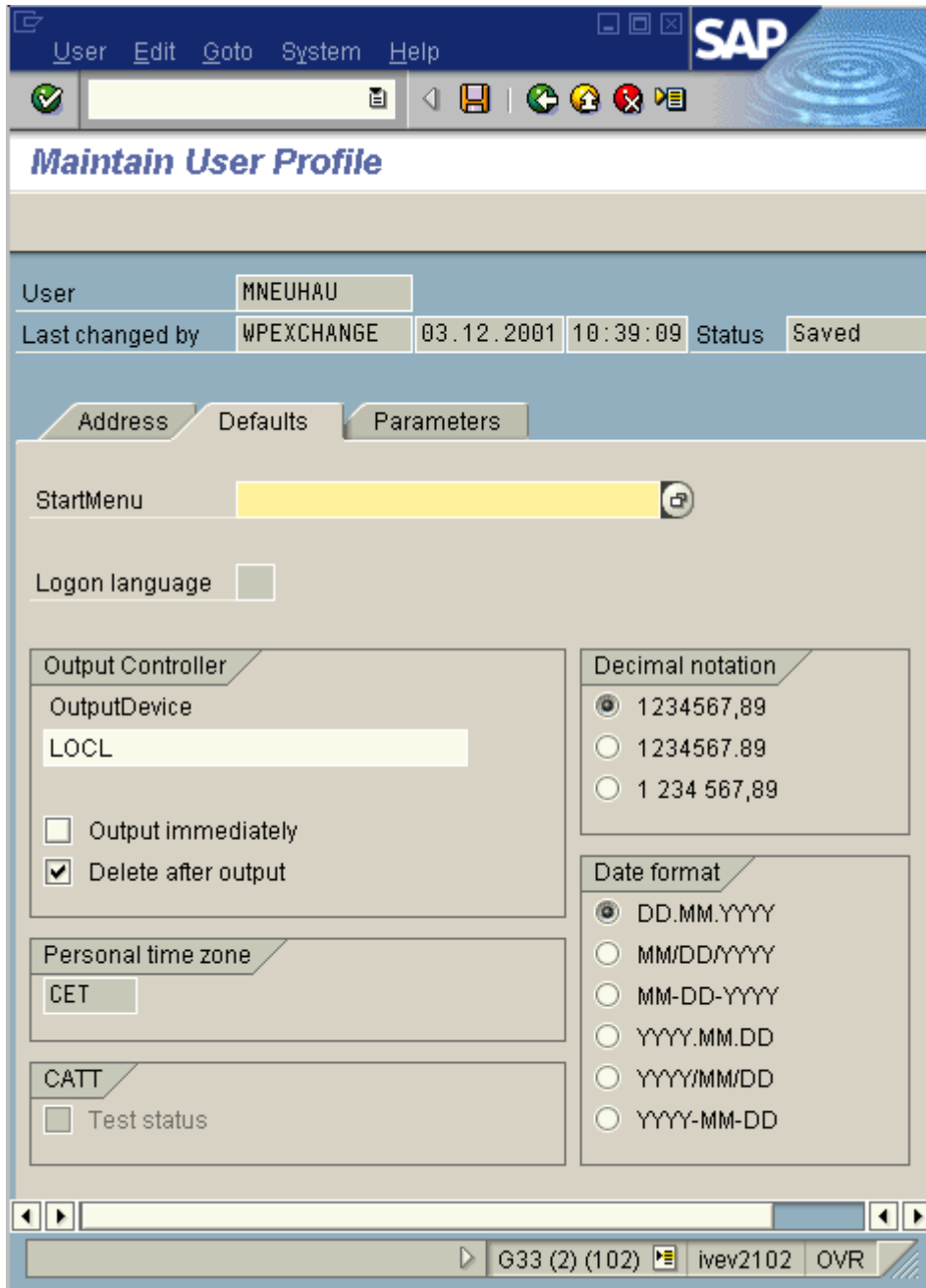
#### 8.1.1 General Remarks and Concepts in ABAP

Different countries use different regional settings e.g. for date, decimal point and paper size. However, the different regional settings in different countries still apply. For example, the decimal separator in Germany, France, and many Spanish speaking countries (including Chile) is a comma (,) and the thousand separator is a (.). So 1,000 would be interpreted in these countries as one rather than one thousand (represented as 1.000,00). If a SAP system started producing reports or screen displays of numbers in a single format worldwide, this could lead to major confusion to users and others (e.g. customers) reading printed reports or screen displays in many countries. Another obvious example is dates. In the US, the date format is MM/DD/YYYY, while the rest of the world is generally on DD/MM/YYYY. 01/03/2001 therefore needs to be interpreted as 1 March 2001 in most of the world, but as 3 January 2001 in the United States.

Regional settings may be different for different countries speaking the same language, so is not necessarily unique to a language. For example, Switzerland speaks French, German and Italian and has a dot (.) as the decimal separator. But in France (also speaking French) or Germany (also speaking German) the decimal separator is a comma (,).

A SAP system has several features for internationalization/localization purposes. Not all of them are explicitly listed in this document. Some examples related to programming are listed below.

Our approach to realize country specific standards is rather pragmatic to minimize programming efforts and realize a solution feasible with the standard system. For decimal notation and date format our recommendation is to make it mandatory for all users to adjust the user master record using the SAP menu 'System -> User profile -> Own data' in accordance with local custom.



Not every country specific format can be found here. . E.g. 1'234'567.89, using an apostrophe (single quote) as separator like it is common in Switzerland is not possible. If this is really needed, it is possible in principle to read the front-end settings from the Registry using function module 'GUI\_GET\_REGVALUE'.

However for reports running in background there is no front-end connection. Transferring front-end settings from the user scheduling the job to a standard batch user means a lot of programming effort. The pragmatic approach is to have a batch-user for every country specific setting in accordance with local custom.

### 8.1.2 ABAP Dictionary

Database tables should not have free-text-fields with language specific text, but should have a language key field and a language table where these texts can be translated.

Database tables containing amounts of money, measures of length, areas, weight, etc should have fields of type CURR (currency fields) and QUAN (quantity fields). For fields of type CURR a field of type CUKY (currency key, referenced by CURR fields) is mandatory to store the corresponding currency e.g. GBP for the amount of money. Similar to this there is a type UNIT (unit key) for QUAN fields specifying the unit for a quantity.

### 8.1.3 ABAP/4 Programming

For list creation in ABAP/4 there are similar features.

Text should never be hard-coded in the program source code but referred to as text-symbols, selection texts, header lines, messages, etc all of them can be translated to any of the imported languages. There is an extended syntax check on that.

- ❖ The formatting options for the **WRITE** statement allow specifying the currency using the **CURRENCY** addition. The number of decimal places for a currency key is stored in table **TCURX**. If there is no value for the specified key, the system assumes that the currency amount has two decimal places.
- ❖ Units can be set according to entries in table **T006** using the **UNITS** statement for fields of type P. If this table does not contain the specified unit key, this option does not have an effect.
- ❖ Setting date format and decimal character
- ❖ Every user can specify in his/her master record (User defaults) how the date is to be formatted on output and whether the decimal character should be a comma or a period.
- ❖ You can also set these output parameters for a particular program by using the statement **SET COUNTRY f**. This displays the decimal point and date in all subsequent output (**WRITE**) according to the settings specified in the table **T005X** for the country ID **f**. If the country does not exist in table **T005X**, the formats used are "." for the decimal point and "MM/DD/YYYY" for the date.
- ❖ The special form **SET COUNTRY SPACE** (or **f** contains **SPACE**) resets the decimal point and date display formats to the setting contained in the current user's master record. In this case, table **T005X** is not read and the return code value is always 0.
- ❖ The effect of **SET COUNTRY** is **not** restricted to the current program, but applies at once to all programs in the current roll area.
- ❖ When downloading lists to Excel the settings mentioned above will be effective. Depending on your Excel country version and the existing number, date and time format codes the download will only interpreted as a date, time or number figure if this format exists in your Excel version. If the format code does not exist the values will be treated as characters and cells will be general format cells with no specific number format.
- ❖ Paper size: please make sure that your list will fit the main paper sizes A4 (21 cm x 29.7 cm) and Letter (21.59 cm x 27.94 cm), i.e. does not exceed 21 cm x 27.94 cm.

### 8.1.4 MDMP

To be able to use several languages with several typefaces, systems are installed as 'MDMP systems'. The abbreviation 'MDMP' stands for Multiple Display code pages / Multiple Processing code pages.

In an MDMP system character data is encoded in different codepages. Depending on the view you choose you will handle only a part of the data correctly. In an MDMP system you have to take care of the proper use of language keys.

Although you may not develop on an MDMP system, your development may be transported or your system may get converted to such a system.

### 8.1.5 SET LOCAL LANGUAGE statement

The ABAP statement `SET LOCAL LANGUAGE lg` allows writing programs, which work on data of different languages in a controlled manner.

Note: The effect of **SET LOCALE** is not restricted to the current program, but affects all programs in the current roll area.

The **TRANSLATE** statement is dependent on the current language environment. The data to be processed may have been entered in another language environment. E.g. when German data is processed in a Russian environment

```
DATA          v_letter(1)  TYPE C value 'ö'.
TRANSLATE    v_letter      TO UPPER CASE.
```

will assign '!' to `v_letter`. In a German environment one would get 'Ö'.

#### 8.1.5.1 Solution 1

To avoid such an error the statement **SET LOCAL LANGUAGE lg** has to be used.

Example: An internal table with a language key has to be processed:

```
DATA: BEGIN OF rec_data OCCURS 0,
      langu LIKE sy-langu,
      txt(20) TYPE CHAR,
      END OF rec_data.

SET LOCALE LANGUAGE rec_data-langu.
TRANSLATE rec_data-txt TO UPPERCASE.
SET LOCALE LANGUAGE SPACE.
```

This is only a simple solution, since it does not require you to make extensive program changes. You call each **TRANSLATE TO UPPERCASE** in the appropriate environment. The command **SET LOCALE LANGUAGE SPACE** always reverts to the logon language

Unfortunately it has certain disadvantages:

- The **SET LOCALE** command is time-consuming.
- It is possible that the language contained in `data-txt` is not allowed. This would trigger a RABAX (exceptions can be tackled using the **CATCH** statement).
- The program might not run in the logon language.

**SET LOCALE LANGUAGE lg** eventually calls the C-function `setlocale`, which is expensive. The price differs with language and with platform but requires several microseconds. To process a huge amount of data the number of executions should be limited by rearranging the data.

#### 8.1.5.2 Solution 2

If sorting is no problem, use **SORT** to keep the number of switches to a minimum.

```
SORT I_DATA BY LANGU.
LOOP AT I_DATA INTO REC_DATA.
  IF REC_DATA-LANGU NE SY-LANGU.
    SET LOCALE LANGUAGE REC_DATA-LANGU.
  ENDIF.
  TRANSLATE REC_DATA-TXT TO UPPER CASE.
ENDLOOP.
SET LOCALE LANGUAGE SPACE.
SORT DATA BY KEYFIELD1 KEYFIELD2.
```

#### 8.1.5.3 Solution 3

If the data is not sorted and sorting is a problem, but the data can be quickly processed one could do several runs.

Create a small table with a language field. Do a first run in the logon language processing only the corresponding data and use **COLLECT** to store all languages in the language table. Looping over this internal table you switch to the next language and process the corresponding datasets by running again throw the whole data, etc.

#### 8.1.5.4 Solution 4

Process only the data corresponding to your logon language, if this does make any sense.

## 8.1.6 Unicode

In the long run SAP will convert to Unicode. Unicode is:

- ❖ A character encoding schema for (nearly) all characters used world wide
- ❖ Each character has a unique number ('Unicode code point')
  - Notation U+nnnn (where nnnn are hexadecimal digits)
- ❖ See <http://www.unicode.org> for complete code charts

In a Unicode system the character data is encoded in only one codepage. There is only one view and misinterpretation is not possible

### 8.1.6.1 Representation of Unicode Characters

The database may contain the same byte representation or any 1:1 transformation into any other Unicode representation (UTF-8, UTF-16 Big Endian, UTF-16 Little Endian).

UTF-16 – Universal Transformation Format, 16 bit encoding

- Fixed length, 1 character = 2 bytes
- Platform dependent byte order
- 2 byte alignment restriction

UTF-8 – Unicode Transformation Format, 8 bit encoding

- Variable length, 1 character = 1 – 3 bytes
- Platform independent
- No alignment restrictions

### 8.1.6.2 Unicode Restrictions

With release 6.10/6.20 SAP introduces new additions to existing key words to improve security, maintainability and readability of ABAP programs. To minimize the efforts and costs necessary to migrate from Non-Unicode to Unicode, programs should already now be designed in a way that takes these changes into account.

#### 8.1.6.2.1 String Processing

With Unicode you will have to distinguish between character processing and byte processing.

Character processing: string operations are only allowed for character-like operands

- ABAP types **C**, **N**, **D**, **T** and **STRING**
- Structures consisting only of characters (**C**, **N**, **D**, **T**)
- **X** and **XSTRING** are no longer considered character-like types

Byte processing: byte operations are only allowed for operands of type **X** or **XSTRING**

- Addition **'IN BYTE MODE'** for statements
- Prefix **'BYTE-'** for comparison operations

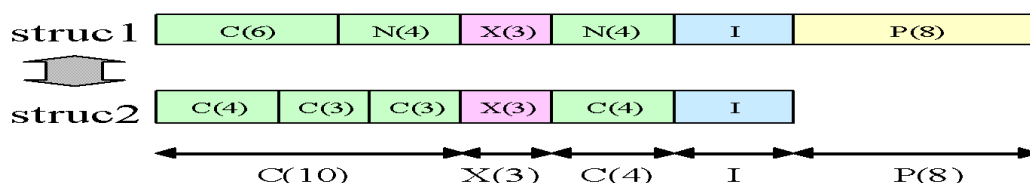
#### 8.1.6.2.2 Length and Distance

The above-mentioned must be taken into account when using the statement **DESCRIBE**. With Unicode you have to distinguish if you want to count the length and distance in bytes or in characters.

#### 8.1.6.2.3 MOVE –Statement

1. **MOVE** between incompatible Structures

To move between structures a matching data layout is required.





Example:

```
DATA: BEGIN OF CSTRUC,
      FIRST(10) TYPE C,
      TAB(1) TYPE C,
      LAST(10) TYPE C,
      END OF CSTRUC.
```

```
DATA: BEGIN OF XSTRUC,
      FIRST(10) TYPE C,
      TAB(1) TYPE X VALUE '09',
      LAST(10) TYPE C,
      END OF XSTRUC.
```

```
CSTRUC = XSTRUC. "UNICODE ERROR!
```

The example above will cause a Unicode error.

2. **MOVE** between Non-Character-like Structures and Elementary Fields

To move between non-character-like structures and elementary fields

- The elementary field must be of type character
- The elementary field must be no longer than the character prefix of the structure



Example:

```
DATA: BEGIN OF REC_PERSON_DATA,
      AGE TYPE I,
      NAME(20) TYPE C,
      END OF REC_PERSON_DATA,
      V_WA(1000) TYPE C.
```

```
V_WA = REC_PERSON_DATA. "UNICODE ERROR!
```

The example above will cause a Unicode error.

8.1.6.2.4 Similar Restrictions Like MOVE

1. Implicit Moves e.g. operations on internal tables

```
LOOP AT i_tab INTO rec_wa.
```

2. Structure Comparison

- ❖ between incompatible structures
- ❖ between structures and elementary fields

3. Moving/Comparing Internal Tables

Similar rules based on line types of tables apply.

4. Database Operations

Select into and update from work areas and internal tables

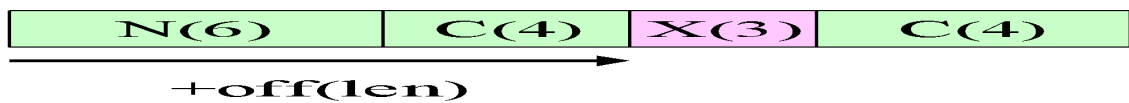
```
SELECT * FROM dbtab INTO rec_wa ...
SELECT * FROM dbtab INTO TABLE i_tab ...
UPDATE dbtab FROM rec_wa ...
```

...

### 8.1.6.2.5 Access with Offset or Length

1. The following restrictions apply when accessing structures with offset or length

- ❖ Structures must begin with character types
- ❖ Offset and length are counted in characters
- ❖ Access is only allowed within the character type prefix of the structures



2. The following restrictions apply when using **ASSIGN fld+off(len)**

- ❖ Access must not exceed field boundaries
- ❖ If **ASSIGN** fails, the field-symbol is set to 'unassigned'
- ❖ **NEW ... RANGE** addition allows the permissible boundaries to be expanded

## 8.2 Language

The ESAP implementation will span multiple countries and hence languages. Therefore any interaction with the user, either through screens or output, language translation should be possible. Therefore not use of literals, or hard coding for messages, labels. Text elements, like text symbols, selection texts should be used.

Note: Oss note 585116 Translation after installation of WebAS620.

### 8.2.1 Text handling

INCLUDE files can't define their own Text Elements - any Text Elements to which they refer must be defined in the main program which invokes the INCLUDE file. Therefore, if it is possible that an INCLUDE file may be invoked from more than one main program, constant text that is used within the INCLUDE file should be defined with the CONSTANTS statement.

In cases other than INCLUDE files, constant text that is printed in a report can be stored as Text Symbols. There are two ways that you can code references to these Text Symbols, either using TEXT-xxx or using '...(xxx)'. Here, xxx stands for a 3-digit number, and ... for the text of the Text Symbol.

The first form requires that you separately define a Text Symbol for number xxx. If xxx isn't a defined Text Symbol, the output is empty. The second form improves the readability of the program. The text between the single quotes should correspond to the text stored as the value of the Text Symbol. If it does not, the system uses the text stored for the Text Symbol. Exception: If there is no text saved under number xxx, the text between the single quotes is used.

Example: Text symbol number 001 has the text 'Please enter your name'. The following commands:

```
WRITE: / TEXT-001, □ preferred option
       / 'Please enter your name'(001),
       / 'What is your name?'(001).
```

all have the same output: "Please enter your name". In the ABAP Editor, you can compare the texts used in the program with the texts stored as Text Symbols by choosing "Goto -> Text elements -> Text symbols", then "Utilities -> Adjust -> Text symbols", then selecting the "Text symbols defined repeatedly/differently in program" radio button and clicking on the "Edit" soft button.

The advantages to the '...(xxx) form are readability and that the Text Symbol only needs to be maintained for multi-lingual clients (for those installations which use multiple languages). The advantage to the **TEXT-xxx** form is easier maintainability if **TEXT-xxx** is coded several times in the program and it needs to be changed.

### 8.3 Country specific requirements,

Again as this implementation programme spans across multiple countries, there could some country specific requirements, especially in the area of enhancements. No hard coding will be allowed in these cases;

```
i.e. If Country eq 'GB'  
      Do country specific logic  
    endif.
```

Is not allowed.

Instead a bespoke table would be used to for this purpose.

## 9 Quality Assurance

### 9.1 Adherence to Programming Standards

Any program object developed will be considered complete if it adheres to the programming standards, and also passes the Unit test. The details of the Unit test would be covered in the Test Strategy document. Completing a Checklist form as part of the development completion activity would check adherence to the Programming Standards. Most of the checks in the checklist would be automated using Code Inspector (transaction SCI); the idea is reduce manual work and thus manual errors as far as possible. A column in the checklist will specify which checks will be performed using Code inspector. In cases where performance of the development is critical to the functionality, like very high volume, or real-time response, premium checks using tools such as Performance analysis (transaction ST05) would be performed.

Checklist Summary:

- ❖ All the relevant documents (latest versions) maintained on APSE / SOLMAN
- ❖ Technical specification is up-to-date and all relevant sections are complete.
- ❖ Online documentation is maintained properly
- ❖ Naming convention is followed
- ❖ Code Indentation is complete
- ❖ Performance standards and guidelines are followed
- ❖ Error Handling
- ❖ Security



"Technical QA  
Checklist.doc"

### 9.2 ABAP Unit Test

ABAP Units is tool, which enables the developer to test units of code, by writing test methods within the code. Since these tests are created within the program, they can be easily synchronised if there are any changes in the program. The ABAP Unit are written using ABAP objects, so no new skills are required for creating these tests. Also these tests are not transported to production system therefore do not introduce any execution overheads.

By setting system parameter "abap/test\_generation"(in transaction RZ12), one can switch on or off ABAP UNIT class compilation in a system. It will be generally switched off in production.

## 10 Security and Authorisations

### 10.1 General

Transactions SE16, SM30, SM31 SA38 and SE38 are generally restricted in Production. All Reports and Programs to be run directly by users should have an associated Report Transaction that can be run from an appropriate Report Tree.

Authorisation Groups

For viewing and maintaining data in tables it is relevant to check the organisation level data of the user, for example which Sales Organisation they belong to. For Reports and Programs it is relevant to enter an appropriate Authority Group.

### 10.2 Reports and Programs

Reports and Programs must have an appropriate Authorisation Group set in the Program Attributes. To check Authorisation from ABAP, use the command Authority-Check.

### 10.3 Viewing Tables

To replace SE16 a new Parameter Transaction needs to be created per table to call SE16 so that the users who we can restrict which users can call the Transaction.

### 10.4 Maintaining Tables

A Parameter transaction should be created to call SM30/31. This will maintain a View of the actual table, that is restricted to one Country. The Authority Group will be in the Table Maintenance of the View.

### 10.5 Dialogue, Report and OO Transactions

With these Transaction types the appropriate Authorisation Object should be checked with the appropriate values. For example, you could use ZTAB\_VKORG, to ensure that the transaction is used only by certain Sales Organisations (i.e. Countries).

### 10.6 ABAP Queries

An Authority Check should be coded using the command 'Authority-Check' or Function Module 'Authority\_Check' to ensure only users with the correct profile and/or from the correct Country can run the Query.

## 11 Appendix A

### 11.1 Interfaces (non-PI) tools

In case no middleware is used to implement the interfaces, Axon has developed a number of tools to facilitate monitoring, file handling and error handling of interfaces. These should be used in all custom-developed interfaces, not using middleware (e.g. PI). These tools will include functionality for:

- ❖ Tracking and monitoring of interface runs
- ❖ Notification via SAP Office mail or e-mail in case of failures
- ❖ File handling (push/pull files using FTP)
- ❖ File archiving
- ❖ Re-sending of files after failed transmission
- ❖ Re-starting failed interfaces

In addition to these tools, interface development standards are in place, which ensure that all interfaces function in a similar way, reducing maintenance and support cost.

Standards regarding directory structures have also been defined.

Note that for all background job scheduling, the standard SAP job scheduling functionality will be used.

Two templates have also been set up to facilitate the writing of interfaces. For inbound interfaces use program template ZINBINTTEMPLATE (appendix C), and for outbound interfaces use ZOUTINTTEMPLATE (appendix D). These templates contain the relevant function modules, which are described in this section.

### 11.2 Interface Management Solution

#### 11.2.1 Overview

The Interface Management Solution consists of a number of bespoke SAP developments, which will be used by support personnel to monitor and manage all interfaces.

The Interface Monitor (program ZBCRIFMONITOR01, transaction ZBC\_IF\_MONITOR) is an interactive report, which lists all interface runs that occurred within a specified timeframe. For each interface run, it lists the start and end date/time, the interface priority, and the status (started/successful/failed).

The Interface Monitor also shows the interface run-time performance. Where an interface did not finish within the expected time frame, this will be highlighted using colour codes.

All interfaces have to be defined in a central interface definition table. This table holds details about the interface and the external system SAP interfaces with. Linked to this table is a notification table, which holds details of people who need to be notified in case of interface failure.

An interface can fail for various reasons, and each failure is assigned a status code. A number of generic status codes have been defined (e.g. interface run successful, inbound file not available, FTP transfer failed, etc). In addition to these generic statuses, the developer can define a number of interface-specific status codes. All status codes with their corresponding description are held in a central interface status code table.

The support documentation for each interface should document each interface specific status code and any relevant generic code, and define the necessary actions that should be taken to resolve the error.

The Interface Monitor queries data, which is generated during each interface run. Each time an interface starts; this is recorded in a history table. When the interface finishes (regardless whether it was successful or not), this record is updated with the relevant status code. Details about files generated/processed during the interface run are also recorded.

A number of function modules have been developed to ensure consistent and accurate update of these history tables. In addition, in case of failure, these function modules handle SAP-Office and/or e-mail notification to any number of recipients, with the details of the failure. It is important that when developing interfaces, the standard mechanisms are used, so that maximum benefits can be gained from the central management function.

In case of failure of an interface run, the Interface Monitor assists support personnel to identify the problem easily. In addition to that, it also helps to perform common tasks like:

- Check the job log
- Process a batch input session containing failed transactions
- Restarting interfaces after failure
- View the files used by a given interface run

A lot of time spent supporting interfaces is ordinarily taken up by the restarting of interfaces that failed in the middle of an interface run. If an inbound interface fails, the incoming file could be partially processed, and, if commits occur within the interface program (e.g. call transactions, BAPI calls with commit, etc) then a number of transactions have been recorded on SAP, and others haven't. Typically, the last successfully processed record in the file needs to be found. A new file is then created manually, only including the records that have not been processed. This can be a time-consuming process.

Although it is difficult to come up with a generic solution for every possible error scenario, the issue of restarting interfaces is being addressed through a standardised development approach, and a number of function modules which facilitate re-starting of interfaces. These function modules hook into the Interface Monitor, so that a failed interface can be restarted from there, without much need for manual intervention.

All 'configuration tables' for the interface management tool (Central interface definition table, Notification rules, Status codes) are maintainable from the Interface Monitor.

### 11.2.2 Central interface definition table

Every interface has to be registered in the Central Interface Definition Table (table ZBC\_IFHEADER). This table holds the following information:

- ❖ **Unique Interface Identifier:** this is a 4 digit alphanumeric unique identifier for the interface
- ❖ **Description:** Meaningful description for the interface
- ❖ **Direction:** inbound or outbound
- ❖ **Priority:** Defines the impact on the business in case the interface does not run or fails (critical/medium/low). Used for escalation purposes
- ❖ **Program Name:** the name of the ABAP interface program
- ❖ **External System:** the name of the external system we are interfacing with. This field is used to determine the physical file path
- ❖ **FTP account details:** IP address of remote system, user ID, password and remote directory for FTP
- ❖ **Email details:** in the case where interface files are delivered to a remote system/partner via email (attachment), the email address and recipient type (U = Internet address, B = SAPOffice user), has to be specified here.
- ❖ **Run Sequence Number Status:** each time the interface is run, the sequence number is incremented by one. This field holds the latest number used
- ❖ **Last Run Date/Time:** the date/time of the most recent successfully finished interface run
- ❖ **Retention period:** specifies a) How many days interface files should be kept on the system before deletion (only applies to files in the archive directory), b) how many days history records should be kept on the system before deletion. If no value is supplied, the interface will not keep history beyond the day in which they were created.

- ❖ **Restartable:** flag indicating whether a failed interface run can be restarted from the Interface Monitoring Tool or not
- ❖ **Expected Run-time:** runtime in minutes that this interface is expected to run for
- ❖ **Run-time threshold:** if the interface does not finish within the expected run-time plus the run-time threshold, an email notification will be sent (based on the defined notification rules).

### 11.2.3 Interface notifications

In the case an interface fails, the support organisation needs to be notified. The notification can happen either via SAP-Office mail or e-mail.

Depending on the day of the week, and the time of day, the support organisation might be different. To satisfy this requirement, a factory calendar has been set up on SAP, defining what the workdays are, i.e. excluding weekends, bank holidays etc. The factory calendar ID is Z1 (Interface Management Factory Calendar).

The factory calendar, in combination with the Interface notification table, defines the rules of who needs to be notified.

The Interface notification table (ZBC\_IFNOTIFY) has the following structure:

- ❖ **Unique Interface Identifier** as defined in the Central Interface Definition Table. This can be a wildcard (\*)
- ❖ **Working Day:** Yes/No flag, indicating whether the rule applies to a working day or not
- ❖ **Priority:** priority of the interface
- ❖ **Start Time:** the earliest time of day this rule applies to
- ❖ **End Time:** the latest time of day this rule applies to
- ❖ **Status type:** derived from the status code
- ❖ **Sequence Number:** incremental number, allowing multiple receivers for one rule
- ❖ **Receiver Type:** B for SAP user (i.e. SAP-Office mail), or U for Internet Address (i.e. email)
- ❖ **Receiver:** email address or SAP user ID

The notification logic will always try to find the most specific rule first (hours and priority are always checked), then applying wildcards, gradually making the rule more generic:

- 1) Read table with interface ID, Working Day Flag, Interface Status
- 2) Read table with interface ID, Working Day Flag, Interface Status = \*
- 3) Read table with interface ID, Working Day Flag = \*, Interface Status = \*
- 4) Read table with interface ID = \*, Working Day Flag, Interface Status
- 5) Read table with interface ID = \*, Working Day Flag, Interface Status = \*

This allows flexible rules, and minimises data maintenance.

Note that the interface status can have an influence on who gets notified. This allows for example that notifications (email report) are sent to business users in case the interface finished successfully, and a different notification to the support organisation in case of failure.

Example:

Intf. ID	WD	Prio	Start	End	Status	Seq	Rec.Typ	Address
----------	----	------	-------	-----	--------	-----	---------	---------



*	Y	1	08:00	17:59	F	1	U	Axonactive
*	Y	1	00:00	07:59	F	1	U	Call centre
*	Y	1	18:00	23:59	F	1	U	Call centre
*	N	1	00:00	23:59	F	1	U	Call centre
HR01	*	1	00:00	23:59	S	1	U	Bob
FI03	*	1	00:00	23:59	F	1	U	Axonactive
FI03	*	1	00:00	23:59	F	2	U	John

The above table implements the following rules:

- ❖ In case of failure of a priority 1 interface on a working day, between 8am and 6pm, Axonactive will be notified. Failure on any other day, or outside these hours will go to the call centre. However, this does not apply to interface FI03 (as there is a more specific rule that applies to this interface).
- ❖ When interface HR01 finishes successfully, Bob will be notified
- ❖ In the case where interface FI03 fails, both Axonactive and John are notified (regardless of date and time)

### 11.2.4 Interfaces not finishing within the expected time-frame

When an interface is still running after the expected run-time plus the threshold, an email notification should be sent. Program ZBCR\_IF\_RUNTIME\_MONITOR reads the history table for all interface runs that are in a status 'started' (since the last run of the program), and have been running longer than the expected run-time plus the threshold. For each interface run found, the notification table will be queried (interface status type = 'Started'), and where necessary, notifications will be sent (See Appendix F).

This program will run at regular intervals (e.g. every 15 minutes).

### 11.2.5 Interface history

A record is stored in the history table (ZBC\_IFHISTORY) each time an interface is run. The monitoring report queries this table to report on successful/failed interface runs, etc.

The following information is held in the history table:

- ❖ **Unique Interface Identifier:** as defined in the Central Interface Definition Table
- ❖ **Run Number:** unique number related to this run
- ❖ **Start Date/Time:** date/time this sequence started
- ❖ **End Date/Time:** date/time this sequence ended
- ❖ **User Name:** ID of the user running the interface
- ❖ **Batch Input Session Name:** for inbound interfaces, errors can be captured in a batch-input session, which can be processed manually by a user at a later stage. This field holds the name of that batch-input session.
- ❖ **Batch Input Session Queue ID:** ID uniquely identifying the batch input session.

- ❖ **Status code:** the status code of the interface run. Status codes are defined in the Interface Status Code table.
- ❖ **Run-time status:** can have the following values:

Run-time status	Description
❖ 0	❖ Finished within expected run-time
❖ 5	❖ Finished within expected run-time + threshold
❖ 8	❖ Did not finish within expected run-time + threshold

- ❖ **Job Name:** the name of the background job (a job is uniquely identified through its job name and job number)
- ❖ **Job Number:** Job ID of the background job (a job is uniquely identified through its job name and job number)

### 11.2.6 Interface files

Most interfaces either process an incoming file, or generate a file that needs to be transmitted to a remote host. All files processed/generated by an interface are recorded for each interface run. If one interface processes more than one file, there is one record for each file in this table.

Each file within a given interface run is assigned a unique file sequence number.

The following information is held in the file status table (table name ZBC\_IFFILES):

- ❖ **Unique Interface Identifier:** as defined in the Central Interface Definition Table
- ❖ **Run Number:** unique sequence number related to this run
- ❖ **File Sequence Number:** sequential number uniquely identifying a file within a given interface run
- ❖ **File name:** name of the file (excluding the file path)
- ❖ **File path:** the path of the file on the SAP box
- ❖ **File Status:** can have the following values:

Status	Description
I	File initialised. Outbound file creation started or inbound file awaiting processing (after OPEN DATASET statement)
C	File created (after CLOSE DATASET statement)
S	File sent successfully to remote host
F	File transfer to remote host failed
R	File received from remote host

P	File processing started (at least one record has been processed)
A	File archived

- ❖ Last record processed: the last record number in the file that was processed by the interface. This is used when restarting the interface.

### 11.2.7 Interface status codes

Each interface run is assigned a status code. This indicates whether the interface has started, finished successfully, or ended with errors. In case of errors, the status code indicates the reason for the failure.

A number of generic status codes have been defined, which can be used for each interface. These include statuses like (not a complete list):

- ❖ Interface Run Started
- ❖ Interface Run Successfully finished
- ❖ Error – Inbound File not available
- ❖ Error – FTP transfer failed

In addition to these generic status codes, the interface developer can define interface-specific status codes. Typically, these status codes are for error states that are specific to that interface, and would not apply to any other interface.

Status Codes are defined in table ZBC\_IFSTATCODES:

- ❖ **Unique Interface Identifier:** as defined in the Central Interface Definition Table. Wildcard (\*) can be used here for generic status codes.
- ❖ **Status Code:** 2 digit status code
- ❖ **Status Type:** high-level classification of status codes, can have the following values:

Status Type	Description
F	Failure
R	Started
S	Successful

- ❖ **Description:** description of the status code

### 11.2.8 Supporting function modules

A number of function modules have been developed, to aid the developer in correctly updating the above tables. These functions should be used in **all** custom interface programs.

- ❖ **Opening an interface run**

Function module: Z\_BC\_IF\_OPEN\_RUN

```
CALL FUNCTION 'Z_BC_IF_OPEN_RUN'
```

```

EXPORTING
    INTFID                =
    RUN_SIMULT            = 'Y'
    MAX_WAIT              = 10
IMPORTING
    RUNNO                =
    START_DATE           =
    START_TIME           =
EXCEPTIONS
    INTERFACE_ID_NOT_FOUND = 1
    ENQUEUE_FAILED        = 2
    INTERFACE_CURRENTLY_RUNNING = 3
    OTHERS                = 4
    
```

The first step in every interface program is opening the interface run. This creates an entry in the Interface history table.

It is possible to prevent opening a new interface run, if the same interface is already running, and this by setting the RUN\_SIMULT flag to 'N'. The default setting ('Y') allows simultaneous interface runs. This functionality only works correctly for programs running in background.

The function module returns the new interface run number. This should be stored in a temporary variable in the interface program, as it has to be passed to function module Z\_BC\_IF\_CLOSE\_RUN to close the interface run at the end of the program.

The function also returns the start date and time, as it is recorded in the history table for this interface run. This can be useful for interfaces that require date-specific SELECT statements (e.g. all employee cost centre changes since the last run of this interface). The start date/time should then be used in the SELECT statements (in order not to select anything after this date and time), to ensure that in the next run of the interface, no records are picked up, which have already been sent in a previous run. Function Z\_BC\_IF\_LAST\_RUN\_DETAILS can be used to determine the date/time of the most recently successfully finished interface run.

In order to increment the interface run number accurately, the ZBC\_IFHEADER table is locked before the update. In the case where there is already a lock on the table, the MAX\_WAIT parameter defines how many seconds the system should wait for the release of that lock.

❖ **Close an interface run**

Function module: Z\_BC\_IF\_CLOSE\_RUN

```

CALL FUNCTION 'Z_BC_IF_CLOSE_RUN'
EXPORTING
    INTFID                =
    RUNNO                =
    MAX_WAIT              = 10
    HISTORY_DETAILS       =
    NO_S_NOTIFICATION    =
TABLES
    EMAIL_CONTENT        =
    
```

EXCEPTIONS

INTERFACE_ID_NOT_FOUND	= 1
INTERFACE_RUN_NOT_OPENED	= 2
INVALID_STATUS_CODE	= 3
ENQUEUE_FAILED	= 4
OTHERS	= 5

The last step in every interface program is closing the current interface run. The interface run has to be opened first, using function module Z\_BC\_IF\_OPEN\_RUN. If this is not the case, exception INTERFACE\_RUN\_NOT\_OPENED is raised.

The HISTORY\_DETAILS structure contains the details to be updated in the history table (it is not necessary to pass the end date/time, as the system date/time is used):

- ❖ Interface Status
- ❖ Batch Input Session details (both group name and Queue ID<sup>1</sup> have to be provided)

The Interface notification table is checked, and if necessary, notifications are generated. The type of notification depends on the receiver type. Typically this will be a SAP-Office message or an Internet email.

It is possible to use parameter EMAIL\_CONTENT to pass the email message that has to be sent out. If this parameter is not passed, then a standard formatted message will be sent (which is different, depending on whether the interface finished successfully or failed).

In the case where the interface finished successfully, the last run date/time of the interface is updated in the Central interface definition table.

The parameter NO\_S\_NOTIFICATION is used to prevent the closure of a successful run from issuing a notification. This can be useful if you are writing a program such as the Runtime Monitor (ZBCR\_IF\_RUNTIME\_MONITOR) that runs successfully every few minutes. This avoids the need to create dummy notification records in the Notifications configuration table for each such program.

❖ **Date/Time of the last successful interface run**

Function module: Z\_BC\_IF\_LAST\_RUN\_DETAILS

```
CALL FUNCTION 'Z_BC_IF_LAST_RUN_DETAILS'
```

```
EXPORTING
```

```
INTFID =
```

```
IMPORTING
```

```
LAST_RUNDATE =
```

```
LAST_RUNTIME =
```

```
EXCEPTIONS
```

```
INTERFACE_ID_NOT_FOUND = 1
```

```
OTHERS = 2
```

---

<sup>1</sup> The Queue ID is returned by the BDC\_OPEN\_GROUP function, when opening the batch input session

## 11.3 Interface directory structures

### 11.3.1 Overview

A standard directory structure to store incoming and outgoing files for all interfaces has been put in place. There is a separate 'interface' directory, which is different on each SAP system (development/test/production). The system ID will be part of the interface path name.

Within this interface directory, there is a sub-directory for each external system SAP interfaces with. The name of this sub-directory corresponds with the 'External System' field in the Central interface definition table.

Each 'External System folder' contains the following sub-directories:

**Inbox:** used to store incoming files, which have not been successfully processed yet

**Outbox:** used to store outgoing files, awaiting delivery to the external system

**Archive:** used to store all files that have been successfully processed/delivered. This directory will be purged on a regular basis (depending on the retention period specified in the central interface definition table)

**Logs:** any log files should be stored in this directory

Example:

Files awaiting delivery to the BACS server would be stored in:

R31\interfaces\BACS\outbox\

where R31 the SAP system ID

File names should always start with the Interface ID, followed by an underscore (\_), and then followed by the interface run number, then and then a freeform prefix.

xxxx\_yyyy\_freeform

where xxxx the Interface ID

yyyyy the interface run number

Example:

BACS\_00025\_payments.txt

where BACS is the interface ID for the BACS interface, and 00025 is the run number.

Only files that have successfully been processed should be moved to archive. For outbound interfaces, in the case where a file transfer to an external system failed, the file should remain in outbox, and stay there until it can be successfully transferred. The same goes for inbound files. If a file is received, and for some reason it cannot be processed at the time the interface is run, it should stay in inbox until it can successfully be processed (or in the case where the file is corrupted, until it is manually removed from the inbox directory by a system administrator).

Interface programs should never use physical file names. Instead the developer should use the logical file names that have been set up. Logical file names allow more flexibility, and when using the provided function modules listed below, it reduces development time. Also, when transporting programs to other environments (test/production), programs do not require changes when the physical file paths change.

### 11.3.2 Supporting function modules

A number of function modules have been developed, to aid the developer with file handling. For a given Interface ID, these functions will generate the correct directory. Interface programs should never have hard-coded directories, and instead should always use the function modules below.

#### ◆ Determining outbound filenames

Function module: Z\_BC\_IF\_FNAME\_OUTBOUND

```
CALL FUNCTION 'Z_BC_IF_FNAME_OUTBOUND'
```

```
EXPORTING
```

```
INTFID =
```

```

RUNNO =
FILENAME_SUFFIX =
IMPORTING
FILE_NAME =
FILE_PATH =
FILE_NAME_WITH_PATH =
EXCEPTIONS
INTERFACE_ID_NOT_FOUND = 1
NO_PHYSICAL_PATH = 2
NO_PHYSICAL_FILENAME = 3
NO_ROOT_PATH = 4
OTHERS = 5
    
```

For a given Interface ID and file suffix, this function module determines the path, the filename (including interface prefix and run number) and the full filename (path + filename) of an outbound file:

```

FILE_NAME          filename without path
FILE_PATH          file path only, without filename
FILE_NAME_WITH_PATH filename with path
    
```

◆ **Determining interface file paths**

Function module: Z\_BC\_IF\_PATH

```

CALL FUNCTION 'Z_BC_IF_PATH'
EXPORTING
INTFID =
PATH_TYPE =
IMPORTING
FILE_PATH =
EXCEPTIONS
INTERFACE_ID_NOT_FOUND = 1
NO_PHYSICAL_PATH = 2
INVALID_PATH_TYPE = 3
NO_ROOT_PATH = 4
OTHERS = 5
    
```

For a given interface ID and path type, returns the corresponding physical path.

The PATH\_TYPE parameter can have the following values:

```

O          for the outbox directory
I          for the inbox directory
    
```

A for the archive directory  
 L for the logs directory

❖ **Get interface directory listing**

Function module: Z\_BC\_IF\_LOCAL\_DIR\_LIST

```
CALL FUNCTION 'Z_BC_IF_LOCAL_DIR_LIST'
  EXPORTING
    INTFID           =
    PATH_TYPE       =
  TABLES
    DIRLIST         =
  EXCEPTIONS
    INTERFACE_ID_NOT_FOUND = 1
    NO_PHYSICAL_PATH      = 2
    NO_ROOT_PATH         = 3
    DIRECTORY_LIST_FAILED = 4
    OTHERS              = 5
```

PATH\_TYPE parameter can have the following values:

O for the outbox directory  
 I for the inbox directory  
 A for the archive directory  
 L for the logs directory

**11.3.3 Logical paths and logical filename set-up**

This paragraph describes the logical path and logical filename set-up, and is mainly intended as a reference. The configuration is owned by the ABAP Team Lead and should by no means be changed by developers.

One logical path, ZBC\_IF\_ROOT has been defined, which specifies the interface root directory.

Four additional logical paths have been defined for the four sub-directories inbox, outbox, logs and archive:

- ZBC\_IF\_INBOX
- ZBC\_IF\_OUTBOX
- ZBC\_IF\_ARCHIVE
- ZBC\_IF\_LOGS

For each of the four 'interface sub-directory logical paths', there are four logical file names.

- ZBC\_IF\_INBOX\_FILE
- ZBC\_IF\_OUTBOX\_FILE
- ZBC\_IF\_ARCHIVE\_FILE
- ZBC\_IF\_LOGS\_FILE



The supporting function modules, listed in the previous paragraph, read these logical paths and filenames, and derive from it the physical paths and filenames.

## 11.4 Interface file handling

### 11.4.1 Introduction

Every file used by an interface is recorded in the interface file table (ZBC\_IFFILES). This allows support personnel to easily identify which files have been processed by a specific interface run, and where the files are currently held. It is therefore important to use the function modules listed below for all file handling.

For inbound files, the interface file table also holds the counter for the last successfully processed record. This will facilitate re-starting of interfaces in case of failure.

### 11.4.2 FTP transfers to and from remote hosts

Very often, files have to be transferred from/to remote hosts. The FTP account details to log on to the remote system are held in the Central interface definition table. A number of function modules have been developed to facilitate sending/receiving files via FTP:

#### ◆ Send a file to a remote host using FTP (Push)

Function module: Z\_BC\_IF\_FILE\_TRANSFER\_PUSH

```
CALL FUNCTION 'Z_BC_IF_FILE_TRANSFER_PUSH'
```

```
EXPORTING
```

```

    INTFID                =
    RUNNO                  =
    FILENO                  =
    FILE_NAME              =
    PATH_TYPE               = 'O'
    WRITE_FLAG_FILE        = ' '
    FLAG_FILE_NAME         = ' '
    DATE_TIME_STAMP        = 'X'
    FORCE_ARCHIVE_OVERWRITE = ' '
    BINARY_MODE             = ' '
    FTP_DIRECTORY          = ' '
    REMOTE_FILENAME        = ' '

```

```
EXCEPTIONS
```

```

    INTERFACE_ID_NOT_FOUND = 1
    NO_PHYSICAL_PATH       = 2
    NO_ROOT_PATH           = 3
    INVALID_FROM_PATH      = 4
    FILE_ALREADY_EXISTS     = 5

```

FTP_CONNECTION_FAILED	=	6
BINARY_MODE_FAILED	=	7
ASCII_MODE_FAILED	=	8
WRITING_FLAGFILE_FAILED	=	9
DELETING_LOCKFILE_FAILED	=	10
FTP_TRANSFER_FAILED	=	11
INVALID_FILE_SEQUENCE_NUMBER	=	12
INVALID_FILE_STATUS	=	13
OTHERS	=	14

The FILENO parameter, as it is returned by function module Z\_BC\_IF\_REGISTER\_FILE, is required. The status of the file should be 'C', prior to calling function Z\_BC\_IF\_FILE\_TRANSFER\_PUSH.

The FILE\_NAME parameter should be the filename without any path information. The path is derived from the PATH\_TYPE parameter, which can be O for the interface outbox directory, or A for the interface archive directory.

It is possible to write a flag file once the actual data file has been transferred. This is to avoid the scenario where the remote system starts processing the file before the FTP transfer has been completely finished. The remote host should check for the existence of the flag file, and only then start processing the actual data file (rather than checking for the presence of the data file itself). The flag filename can be specified through the FLAG\_FILE\_NAME parameter. If left blank, the filename of the flag file is identical to the filename of the actual data file, with an added extension of '.flg'.

If the FTP transfer was successful, then the file is automatically archived (i.e. moved to the archive directory), this to avoid it is sent again during a subsequent interface run. The default is to add a date/time stamp to the filename when it is moved to archive, however setting the parameter DATE\_TIME\_STAMP to blank disable this. By default, the system checks whether the file already exists in the archive, and if it does, will raise an exception. Again, setting the FORCE\_ARCHIVE\_OVERWRITE parameter to 'X' disables this.

The BINARY\_MODE parameter can be used to transfer files in binary mode.

By default, the FTP directory on the remote host is derived from the Central Interface Management table (ZBC\_IFHEADER). However, it is possible to overrule this setting by passing the remote directory in the FTP\_DIRECTORY parameter.

By default, the file will not be renamed during the FTP transfer. However, this can be achieved by passing the remote filename in the REMOTE\_FILENAME parameter.

This function will update the status of the file in the interface file table to S (sent but archiving failed), F (transfer failed) or A (archived).

**Important note:** This function module uses external commands to move and delete files on operating system level. Users using these function modules require authorisation object S\_LOG\_COM (class BC\_A), otherwise these external commands will not be executed.

◆ Retrieve a file from a remote host using FTP (Pull)

Function module: Z\_BC\_IF\_FILE\_TRANSFER\_PULL

```
CALL FUNCTION 'Z_BC_IF_FILE_TRANSFER_PULL'
```

```
EXPORTING
```

```
INTFID =
```

```
RUNNO =
```

```
FILE_NAME =
```

```

FORCE_OVERWRITE           = ' '
BINARY_MODE               = ' '
DELETE_FILE               = ' '
IMPORTING
  FILENO                   =
EXCEPTIONS
  INTERFACE_ID_NOT_FOUND  = 1
  NO_PHYSICAL_PATH        = 2
  NO_ROOT_PATH            = 3
  FILE_ALREADY_EXISTS     = 4
  FTP_CONNECTION_FAILED   = 5
  BINARY_MODE_FAILED      = 6
  ASCII_MODE_FAILED       = 7
  FTP_TRANSFER_FAILED     = 8
  DELETE_FILE_FAILED      = 9
  OTHERS                   = 10

```

Files are always received in the interface inbox directory. By default it is checked whether a file with the same name already exists in the inbox. If this is the case, then an exception is raised. Setting the FORCE\_OVERWRITE parameter to X disables this.

The BINARY\_MODE parameter can be used to transfer files in binary mode.

It is possible to delete the file on the remote host (only after successful transfer), by setting the DELETE\_FILE parameter to X.

The function automatically registers the new file in the Interface file table, and sets the status of the file to 'R'. It returns the file sequence number in the parameter FILENO.

◆ **Get remote directory listing using FTP**

Function module: Z\_BC\_IF\_REMOTE\_DIR\_LIST

```
CALL FUNCTION 'Z_BC_IF_REMOTE_DIR_LIST'
```

```

EXPORTING
  INTFID                   =
  PREFIX                   =
TABLES
  DIRLIST                   =
EXCEPTIONS
  INTERFACE_ID_NOT_FOUND  = 1
  FTP_CONNECTION_FAILED   = 2
  DIRECTORY_LIST_FAILED   = 3
  CHANGE_DIRECTORY_FAILED = 4

```

OTHERS = 5

By default, all files in the directory of the remote host (defined in the Central Interface Definition table, or if blank, the home directory of the FTP user) are listed. It is possible to only list files with a given prefix, by setting the PREFIX parameter.

### 11.4.3 Transferring files using e-mail

It is possible to deliver files to remote systems/partners via email. The file is included in the e-mail message as an attachment. This functionality works for both ASCII and binary files.

Function module: Z\_BC\_IF\_FILE\_TRANSFER\_EMAIL

```
CALL FUNCTION 'Z_BC_IF_FILE_TRANSFER_EMAIL'
```

EXPORTING

```

INTFID           =
RUNNO           =
FILENO          =
FILE_NAME       =
DOC_TYPE        = 'RAW'
PATH_TYPE       = 'O'
DATE_TIME_STAMP = 'X'
FORCE_ARCHIVE_OVERWRITE = ' '
    
```

EXCEPTIONS

```

INTERFACE_ID_NOT_FOUND = 1
NO_PHYSICAL_PATH       = 2
NO_ROOT_PATH           = 3
INVALID_FROM_PATH      = 4
FILE_ALREADY_EXISTS    = 5
INVALID_FILE_SEQUENCE_NUMBER = 6
INVALID_FILE_STATUS    = 7
EMAIL_TRANSFER_FAILED  = 8
NO_EMAIL_RECEIVER_DEFINED = 9
OTHERS                 = 10
    
```

This function works in very much the same way as function Z\_BC\_IF\_FILE\_TRANSFER\_PUSH, instead that the file is transferred using e-mail instead of FTP.

The DOC\_TYPE parameter is useful in the case of sending SAPOffice mails. If set correctly, when the user double clicks on the file attachment icon in SAPOffice, the associated application is launched, and the file is displayed correctly (e.g. BMP for bitmaps).

### 11.4.4 Creating interface files

In the case where an interface creates new files (typically for outbound interfaces), these should be registered in the Interface file table. This is done using the following function module:

Function module: Z\_BC\_IF\_REGISTER\_FILE

```
CALL FUNCTION 'Z_BC_IF_REGISTER_FILE'
  EXPORTING
    INTFID           =
    RUNNO           =
    FILE_NAME       =
    FILE_PATH       =
  IMPORTING
    FILENO          =
  EXCEPTIONS
    INTERFACE_ID_NOT_FOUND = 1
    INTERFACE_RUN_NOT_OPENED = 2
    OTHERS          = 3
```

Typically, this function should be called immediately after the OPEN DATASET statement.

The function creates an entry in table ZBC\_IFFILES, and set the file status to 'I' (creation started).

The function returns the file sequence number. This should be stored in a local variable within the interface program, as it will be required when transferring files via FTP, etc

Once the file has been created (all records have been transferred and the dataset has been closed), the Interface file table has to be updated with the new status 'C'. This is done using the following function module:

Function module: Z\_BC\_IF\_FILE\_CREATED

```
CALL FUNCTION 'Z_BC_IF_FILE_CREATED'
  EXPORTING
    INTFID           =
    RUNNO           =
    FILENO          =
  EXCEPTIONS
    INTERFACE_ID_NOT_FOUND = 1
    INTERFACE_RUN_NOT_OPENED = 2
    INVALID_FILE_SEQUENCE_NUMBER = 3
    INVALID_FILE_STATUS = 4
    OTHERS          = 5
```

Typically, this function should be called immediately after the CLOSE DATASET statement. The status of the file should be 'I', prior to calling this function.

### 11.4.5 Processing inbound files

Inbound files are typically received via FTP, by using function module Z\_BC\_IF\_FILE\_TRANSFER\_PULL. This function will automatically register the inbound file in the Interface file table. If the file is received in a different

way, then the file needs to be registered first using function module Z\_BC\_IF\_REGISTER\_FILE. This call is typically made immediately after the OPEN DATASET statement.

In order to make re-starting of failed interfaces as easy as possible, it is important to keep track of what records have been processed in the file. The Interface file table holds for each file used by a given interface run the record counter of the last successfully processed record. Therefore, each time the interface has processed a record (or a logical group of records), and these have been committed to the database, the record counter should be updated. This is done using the following function module:

```
Function module: Z_BC_IF_FILE_UPDATE_RECCOUNT
CALL FUNCTION 'Z_BC_IF_FILE_UPDATE_RECCOUNT'
  EXPORTING
    INTFID           =
    RUNNO           =
    FILENO          =
    RECORD_COUNTER  =
  EXCEPTIONS
    INTERFACE_ID_NOT_FOUND      = 1
    INTERFACE_RUN_NOT_OPENED   = 2
    INVALID_FILE_SEQUENCE_NUMBER = 3
    OTHERS                      = 4
```

### 11.4.6 Archiving files

Files that have been successfully processed should be moved into archive, so that a subsequent run of the interface doesn't pick up files that have already been processed. This is done using the following function module:

```
Function module: Z_BC_IF_ARCHIVE_FILE
CALL FUNCTION 'Z_BC_IF_ARCHIVE_FILE'
  EXPORTING
    INTFID           =
    FILE_NAME        =
    PATH_TYPE        =
    DATE_TIME_STAMP  = 'X'
    FORCE_ARCHIVE_OVERWRITE = ' '
  EXCEPTIONS
    INTERFACE_ID_NOT_FOUND      = 1
    NO_PHYSICAL_PATH           = 2
    NO_ROOT_PATH               = 3
    INVALID_FROM_PATH          = 4
    FILE_ALREADY_EXISTS        = 5
    OTHERS                      = 6
```

This function moves a file from either the inbox or the outbox directory to archive. The PATH\_TYPE should be either O (outbox) or I (inbox).

The FILE\_NAME parameter is the name of the file (without any path information).

When DATE\_TIME\_STAMP is set to X (default) a date and time stamp will be added to the filename when the file is moved to the archive directory.

When FORCE\_ARCHIVE\_OVERWRITE is set to X, when a file with the same name already exists in the archive directory, it will be overwritten. If not set to X, and the file already exists, exception FILE\_ALREADY\_EXISTS will be raised.

### 11.4.7 Deleting interface files

Interface files have to be kept in the archive directory for a certain time. The retention period can be specified for each interface individually (in the Central Interface Definition Table).

A program ZBCR\_IF\_PURGE has been developed, which should run on a regular basis and delete any interface files from the archive and logs directory, which are older than the retention period specified for their respective interface. This program will also delete entries from the history and file table (See Appendix F).

## 11.5 Restarting of interfaces

The technical design for any interface should take into account the possibility for restarting interfaces that failed in the middle of a run. This is particularly important for inbound interfaces, where an interface processes an inbound file. If the interface fails for whatever reason (e.g. system failure, ABAP short dump, etc.) then it should be possible to restart the interface with a minimum of manual intervention.

In addition to that, each interface should have the following hidden parameter on the selection screen:

- ❖ p\_rstrt: indicator to tell the interface that it is being restarted
- ❖ p\_runno: the interface run which will be restarted

The Interface Monitor will pass these parameters automatically when an interface is re-started. Specific logic should be coded into the interface program to handle these restarts (e.g. only start processing the inbound filename from the last processed record, etc.).

## 12 Appendix B – Inbound interface template

```

*&-----*
*& Report  ZINBINTTEMPLATE                               *
*&                                                    *
*&-----*
*&                                                    *
*&                                                    *
*&-----*

REPORT  zinbinttemplate                               .

*-----*
*  RESPONSIBILITY - ORIGINAL BUILD                    *
*-----*
* Author.....:                                       *
* Creation date.....:                               *
* Requested by.....:                               *
* Owner.....:                                       *
* Project.....:                                       *
* SAP Release.....: R/3 Enterprise 470              *
* Logical database..:                               *
*-----*
* Revised by.....:                                   *
* Change date.....:                                   *
* Change no.....:                                   *
* Initals.....:                                       *
* Description.....:                                   *
*-----*

*-----*
* Data Definition                                     *
*-----*

*Tables

*Infotypes

*Internal Tables

*Variables
DATA: gw_runno LIKE zbc_ifheader-runno,

```



```

gw_ifhistory LIKE zbc_ifhistory,
gw_filename_with_path(255),
gw_filename(255),
gw_filepath(255),
gw_fileno TYPE ziffileno,
gw_testline(5) TYPE n,
gw_line(255),
gw_reccount TYPE zifreccount.

```

\*Constants

```
DATA: gc_intid(4) VALUE '0002'.
```

```

*-----*
* Includes                                     *
*-----*

```

```

*-----*
* Selection Screen Definition                 *
*-----*

```

```

PARAMETERS: p_fname1(40) LOWER CASE.
PARAMETERS: p_fname2(40) LOWER CASE.

```

```

*-----*
* Initialization                             *
*-----*

```

```
INITIALIZATION.
```

```

*-----*
* S T A R T   O F   S E L E C T I O N       *
*-----*

```

```
START-OF-SELECTION.
```

```
PERFORM open_interface.
```

```
PERFORM open_file USING p_fname1.
```

```
PERFORM open_file USING p_fname2.
```

```
PERFORM process_data.
```

```

*-----*
*  E N D   O F   S E L E C T I O N                               *
*-----*
END-OF-SELECTION.

```

```

    PERFORM close_interface.

```

```

*-----*
*  F O R M S   . . . . . Start here                               *
*-----*
*&-----*
*&      Form  open_interface                                     *
*&-----*
*      text
*-----*
*  -->  p1      text
*  <--  p2      text
*-----*
FORM open_interface .

```

```

* Create a record in the run log
  CALL FUNCTION 'Z_BC_IF_OPEN_RUN'
    EXPORTING
      intfid           = gc_intid
*   RUN_SIMULT        = 'Y'
*   MAX_WAIT          = 10
    IMPORTING
      runno            = gw_runno
    .

```

```

ENDFORM.                " open_interface

```

```

*&-----*
*&      Form  close_interface                                     *
*&-----*
*      text
*-----*

```

```

* --> p1      text
* <-- p2      text
*-----*
FORM close_interface .

* Set the status which the interface run will be closed with.
* gw_ifhistory-ifstat =

* Close the run in the history table.
CALL FUNCTION 'Z_BC_IF_CLOSE_RUN'
  EXPORTING
    intfpid      = gc_intfid
    runno        = gw_runno
*   MAX_WAIT    = 10
    history_details = gw_ifhistory
  .

ENDFORM.                " close_interface
*&-----*
*&      Form  process_data
*&-----*
*      text
*-----*
* --> p1      text
* <-- p2      text
*-----*
form process_data .
*<<< Enter your processing here.
endform.                " process_data

*&-----*
*&      Form  open_file
*&-----*
*      text
*-----*
FORM open_file USING lp_fname.

```

```

* Pull the first file from FTP
CALL FUNCTION 'Z_BC_IF_FILE_TRANSFER_PULL'
  EXPORTING
    intfid                = '0002'
    runno                 = gw_runno
    file_name             = lp_fname
*   FORCE_OVERWRITE       = ' '
*   BINARY_MODE          = ' '
*   DELETE_FILE          = ' '
  IMPORTING
    fileno                = gw_fileno
  EXCEPTIONS
    interface_id_not_found = 1
    no_physical_path       = 2
    no_root_path           = 3
    file_already_exists    = 4
    ftp_connection_failed  = 5
    binary_mode_failed     = 6
    ascii_mode_failed      = 7
    ftp_transfer_failed    = 8
    delete_file_failed    = 9
    OTHERS                 = 10
.

CALL FUNCTION 'Z_BC_IF_PATH'
  EXPORTING
    intfid                = '0002'
    path_type             = 'I'
  IMPORTING
    file_path             = gw_filepath
*   EXCEPTIONS
*   INTERFACE_ID_NOT_FOUND = 1
*   NO_PHYSICAL_PATH       = 2
*   INVALID_PATH_TYPE     = 3
*   NO_ROOT_PATH          = 4
*   OTHERS                 = 5
.

```

```

* Open the file
CONCATENATE gw_filepath lp_fname INTO gw_filename_with_path.
OPEN DATASET gw_filename_with_path FOR INPUT
IN TEXT MODE ENCODING DEFAULT.

DO.
  gw_reccount = sy-index.
  READ DATASET gw_filename_with_path INTO gw_line.

  IF sy-subrc <> 0.
    CLOSE DATASET gw_filename_with_path.

    CALL FUNCTION 'Z_BC_IF_ARCHIVE_FILE'
      EXPORTING
        intfid                = '0002'
        file_name              = lp_fname
        path_type              = 'I'
        * DATE_TIME_STAMP      = 'X'
        * FORCE_ARCHIVE_OVERWRITE = ' '
      EXCEPTIONS
        interface_id_not_found = 1
        no_physical_path       = 2
        no_root_path           = 3
        invalid_from_path      = 4
        file_already_exists     = 5
        archive_failed         = 6
        OTHERS                 = 7
        .

    EXIT.
  ENDIF.

* Update record counter
CALL FUNCTION 'Z_BC_IF_FILE_UPDATE_RECCOUNT'
  EXPORTING
    intfid      = '0002'
    runno       = gw_runno
    fileno      = gw_fileno
    record_counter = gw_reccount.

```

```
ENDDO.  
  
CLOSE DATASET gw_filename_with_path.  
  
ENDFORM.          " open_file
```

## 13 Appendix C – Outbound Interface Template

```

*&-----*
*& Report  ZOUTINTTEMPLATE                               *
*&                                                *
*&-----*
*&                                                *
*&                                                *
*&-----*

REPORT  zoutinttemplate                               .

*-----*
*  RESPONSIBILITY - ORIGINAL BUILD                    *
*-----*
* Author.....:                                       *
* Creation date.....:                               *
* Requested by.....:                               *
* Owner.....:                                       *
* Project.....:                                       *
* SAP Release.....: R/3 Enterprise 470              *
* Logical database..:                               *
*-----*
* Revised by.....:                                   *
* Change date.....:                                   *
* Change no.....:                                   *
* Initals.....:                                       *
* Description.....:                                   *
*-----*

*-----*
* Data Definition                                     *
*-----*

*Tables

*Infotypes

*Internal Tables

*Variables
DATA: gw_runno LIKE zbc_ifheader-runno,

```

```

    gw_ifhistory LIKE zbc_ifhistory,
    gw_filename_with_path(255),
    gw_filename(255),
    gw_filepath(255),
    gw_fileno TYPE ziffileno,
    gw_testline(5) TYPE n.

*Constants
DATA: gc_intid(4) VALUE '0001'.

*-----*
* Includes                                     *
*-----*

*-----*
* Selection Screen Definition                 *
*-----*
*PARAMETERS:

*-----*
* Initialization                             *
*-----*
INITIALIZATION.

*-----*
* S T A R T   O F   S E L E C T I O N         *
*-----*
START-OF-SELECTION.

    PERFORM open_interface.

    PERFORM generate_fname.

    PERFORM register_file.

    PERFORM process_data.

```



```

*-----*
*  E N D   O F   S E L E C T I O N                               *
*-----*
END-OF-SELECTION.

    PERFORM create_file.

    PERFORM send_file.

    PERFORM close_interface.

*-----*
*  F O R M S   . . . . . Start here                               *
*-----*
*&-----*
*&      Form  open_interface                                     *
*&-----*
*      text
*-----*
*  --> p1      text
*  <-- p2      text
*-----*
FORM open_interface .

* Create a record in the run log
CALL FUNCTION 'Z_BC_IF_OPEN_RUN'
  EXPORTING
    intfId      = gc_intid
*  RUN_SIMULT  = 'Y'
*  MAX_WAIT    = 10
  IMPORTING
    runno       = gw_runno
  .

ENDFORM.          " open_interface
*&-----*
*&      Form  generate_fname                                     *
*&-----*
*      text
*-----*

```

```

* --> p1      text
* <-- p2      text
*-----*
FORM generate_fname .

* Generate the outbound filename from the interface ID, the run number
* and the supplied suffix.
CALL FUNCTION 'Z_BC_IF_FNAME_OUTBOUND'
  EXPORTING
    intfid      = gc_intid
    runno       = gw_runno
    filename_suffix = 'File_1'
  IMPORTING
    file_name      = gw_filename
    file_path      = gw_filepath
    file_name_with_path = gw_filename_with_path.

ENDFORM.                " generate_fname
*&-----*
*&      Form  register_file
*&-----*
*      text
*-----*
* --> p1      text
* <-- p2      text
*-----*
FORM register_file .

* Register the file to be created
CALL FUNCTION 'Z_BC_IF_REGISTER_FILE'
  EXPORTING
    intfid      = gc_intid
    runno       = gw_runno
    file_name    = gw_filename
    file_path    = gw_filepath
  IMPORTING
    fileno      = gw_fileno.

ENDFORM.                " register_file
*&-----*
*&      Form  create_file

```

```

*&-----*
*      text
*-----*
* --> p1      text
* <-- p2      text
*-----*
FORM create_file .

* Create the file
OPEN DATASET gw_filename_with_path FOR OUTPUT
IN TEXT MODE ENCODING DEFAULT.

TRANSFER gw_testline TO gw_filename_with_path.

CLOSE DATASET gw_filename_with_path.

* Register that the file creation is complete
CALL FUNCTION 'Z_BC_IF_FILE_CREATED'
EXPORTING
  intfid = gc_intid
  runno  = gw_runno
  fileno = gw_fileno.

ENDFORM.              " create_file
*&-----*
*&      Form  send_file
*&-----*
*      text
*-----*
* --> p1      text
* <-- p2      text
*-----*
FORM send_file .

* Push out the file to a remote host
CALL FUNCTION 'Z_BC_IF_FILE_TRANSFER_PUSH'
EXPORTING
  intfid          = gc_intid
  runno           = gw_runno

```

```

fileno                = gw_fileno
file_name             = gw_filename
path_type             = 'O'
*   WRITE_FLAG_FILE   = ' '
*   FLAG_FILE_NAME    = ' '
*   DATE_TIME_STAMP   = 'X'
*   FORCE_ARCHIVE_OVERWRITE = ' '
*   BINARY_MODE       = ' '
*   FTP_DIRECTORY     = ' '
*   REMOTE_FILENAME   = ' '
EXCEPTIONS
  interface_id_not_found = 1
  no_physical_path       = 2
  no_root_path           = 3
  invalid_from_path      = 4
  file_already_exists    = 5
  ftp_connection_failed  = 6
  binary_mode_failed     = 7
  ascii_mode_failed      = 8
  writing_flagfile_failed = 9
  deleting_lockfile_failed = 10
  ftp_transfer_failed    = 11
  invalid_file_sequence_number = 12
  invalid_file_status    = 13
  OTHERS                 = 14
.
IF sy-subrc <> 0.
*   MESSAGE ID SY-MSGID TYPE SY-MSGTY NUMBER SY-MSGNO
*           WITH SY-MSGV1 SY-MSGV2 SY-MSGV3 SY-MSGV4.
ENDIF.

ENDFORM.                " send_file
*&-----*
*&   Form   close_interface
*&-----*
*   text
*-----*
* --> p1      text
* <-- p2      text
*-----*

```

```

FORM close_interface .

* Set the status which the interface run will be closed with.
*   gw_ifhistory-ifstat =

* Close the run in the history table.
CALL FUNCTION 'Z_BC_IF_CLOSE_RUN'
  EXPORTING
    intfid           = gc_intid
    runno            = gw_runno
*   MAX_WAIT        = 10
    history_details = gw_ifhistory
  .

ENDFORM.                " close_interface

*&-----*
*&   Form   process_data
*&-----*
*   text
*-----*
* --> p1      text
* <-- p2      text
*-----*

form process_data .
*<<< Enter your processing here.
endform.                " process_data

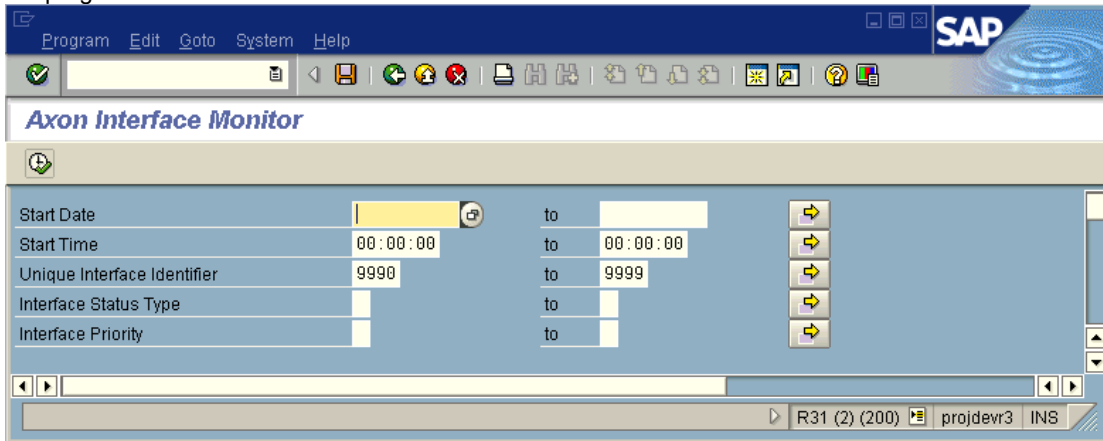
```

## 14 Appendix D – Axon Interface Monitor User Guide

This section introduces and describes the features of the Axon Interface Monitor. The monitor is launched by executing program ZBCRIFMONITOR01 or by choosing transaction ZBC\_IF\_MONITOR.

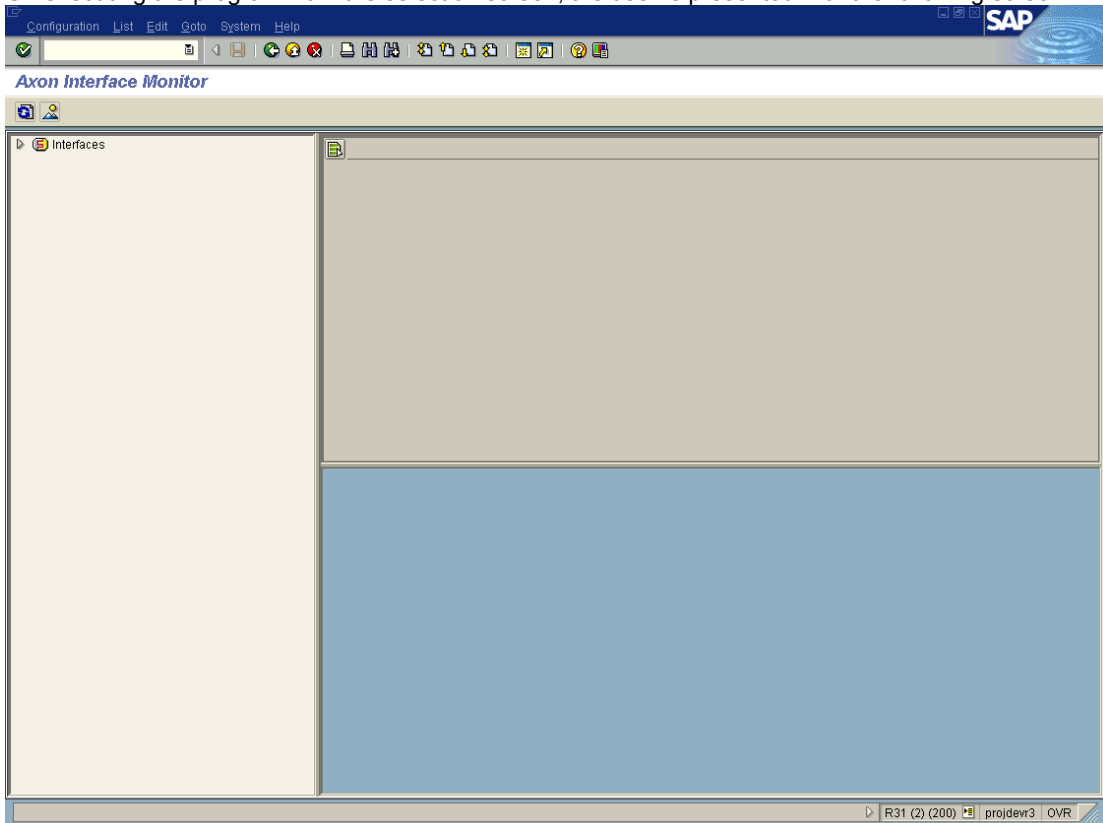
### 14.1 Selection screen

The selection screen of the Interface Monitor is very simple, allowing the user to restrict the histories selected by the program to a subset of Start Dates and Start Times.



### 14.2 The control tree

On executing the program from the selection screen, the user is presented with the following screen.



button  
ushes the  
in the  
itor from  
database.  
tree will  
y  
expanding  
clicking  
button.

The user must expand the nodes on the tree in the left-hand pane in order to see different levels of information.



Above we can see examples of all levels on the tree. The Interfaces (root) node contains two sub-nodes – Inbound and Outbound. Information in these top two levels of the tree is at the level of individual interfaces – the meaning of this will become clearer in the next section when the reports that can be launched from these top two nodes are explained.

Beneath both the Inbound and Outbound nodes are three Interface Status Type sub-nodes – Failed, Started and Successful. Information held in these Status Type nodes and below is at the level of individual interface runs. The bracketed number next to the node is a count of the number of runs in the history database. For example, the '(3)' appearing next to the Failed node under the Inbound node tells us that there are three failed inbound interface runs in the database.

Expanding one of the three Status Type nodes reveals a sub node for each interface that has runs in that Status Type node. Against the Interface node can be seen the priority of the interface and its name.

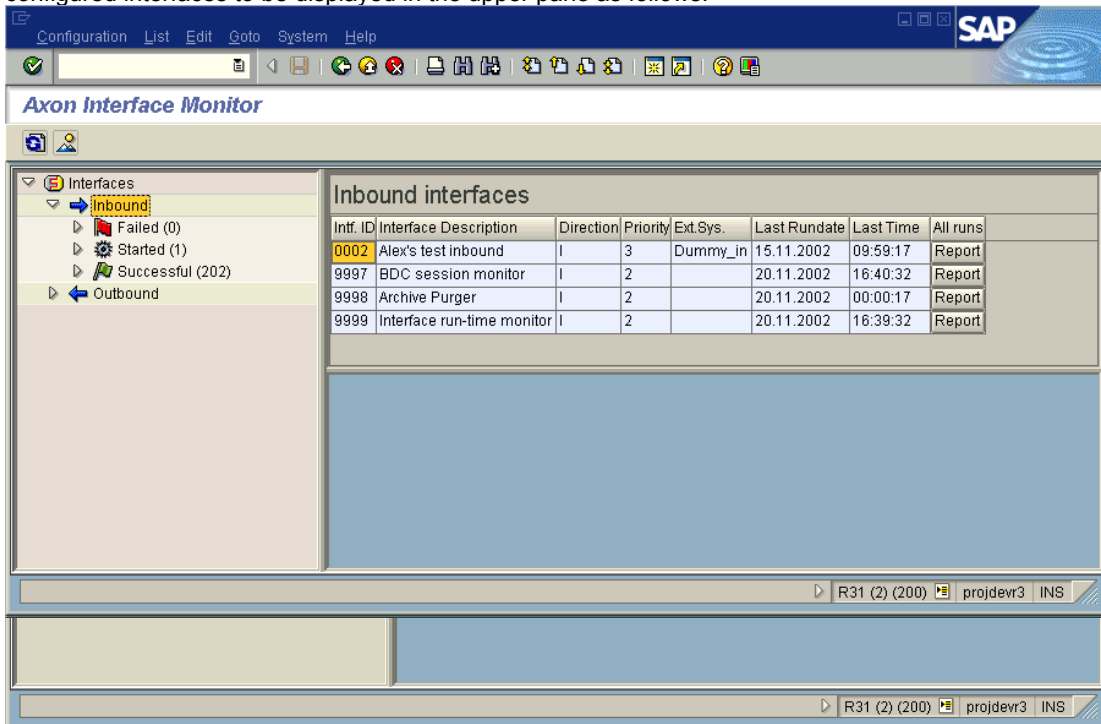
Expanding an Interface node reveals a node for each run held against that Interface node. Against the Run node can be seen its run number, its start date and its start time.

## 14.3 Interacting with the tree

Double clicking on a node of the tree causes different levels of information to be reported in the upper pane. Three types of report can be produced: **Interface level, Run level and File level.**

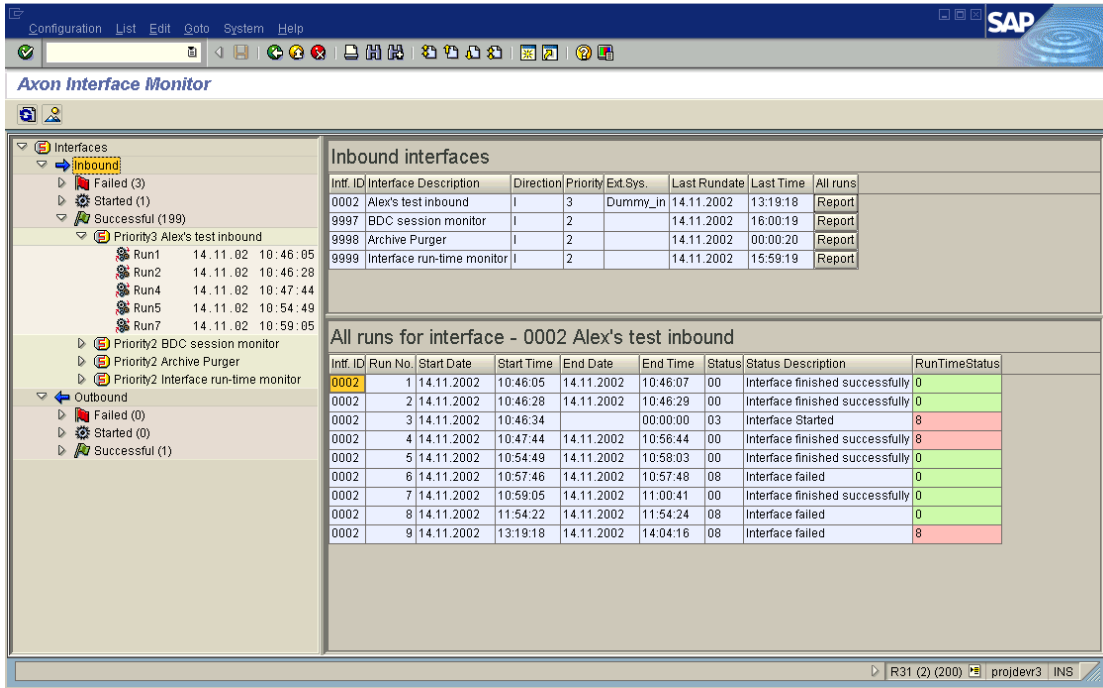
### 14.3.1 Interface level

Double clicking on the Interfaces (root) node or on either the Inbound or Outbound Node causes lists of configured interfaces to be displayed in the upper pane as follows.



Clicking the 'Report' button against any of the interface rows will cause a further report of all runs for that interface to be displayed in the lower pane. This provides a place to see run level information for each configured interface independently of status types.





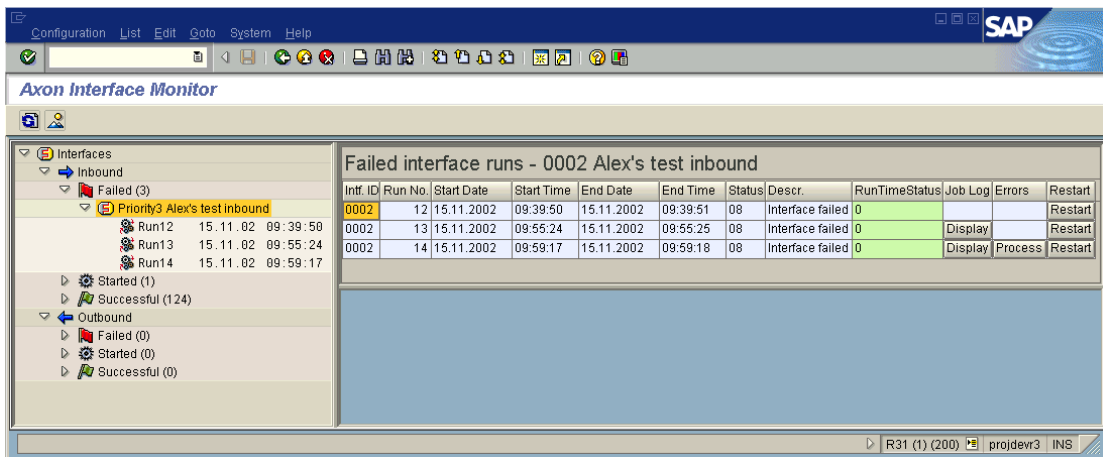
Run level

Double clicking on a Status Type node or an Interface node causes lists of interface runs to be displayed in the upper pane as follows.

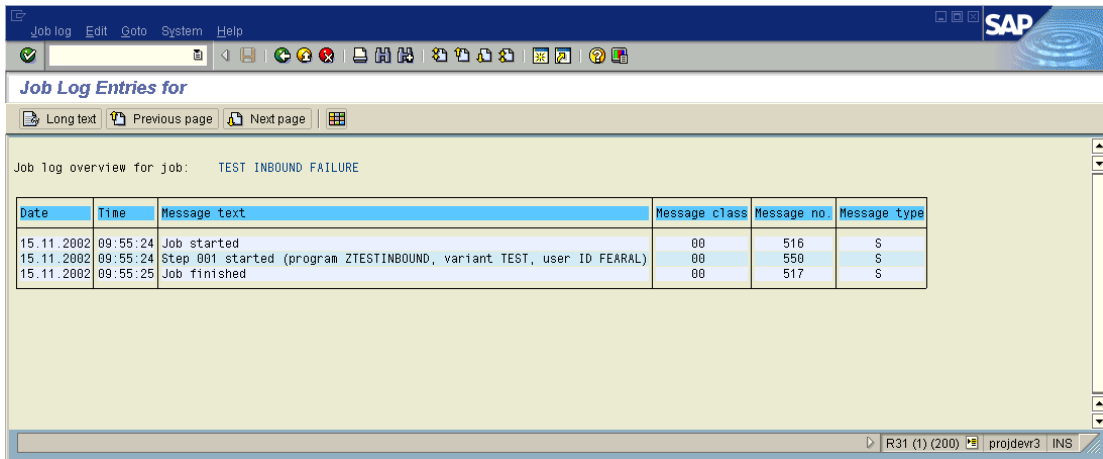
Notice the colour-coded run time status field. This field shows whether the run completed within the configured expected time (green), overran within the configured threshold value (yellow), or overran the threshold value (red). This allows support staff to see the performance of an interface over a number of runs, and to investigate the progress of a run that is ongoing.

Notice also the three fields at the end of each row – Job Log, Errors, and Restart. These fields can contain pushbuttons depending on the state of the run.

If a run occurred via a scheduled job, there will be a 'Display' button in the Job Log column. Clicking this button causes the SAP job log for that run to be displayed on the screen.



Press the Display button against a run



The screenshot shows the SAP Job Log Entries interface. The title bar includes 'Job log Edit Goto System Help' and the SAP logo. Below the title bar, there are navigation buttons: 'Long text', 'Previous page', and 'Next page'. The main content area displays 'Job log overview for job: TEST INBOUND FAILURE'. A table with the following data is shown:

Date	Time	Message text	Message class	Message no.	Message type
15.11.2002	09:55:24	Job started	00	516	S
15.11.2002	09:55:24	Step 001 started (program ZTESTINBOUND, variant TEST, user ID FEARAL)	00	550	S
15.11.2002	09:55:25	Job finished	00	517	S

At the bottom right of the window, the status bar shows 'R31 (1) (200) projdevr3 INS'.

Press the Process button against a run

If the interface run failed and the interface program generated a BDC session containing the errors, then there will be a 'Process' button in the Errors field. Clicking this button will pass control to the Batch Input: session overview screen (transaction SM35) for the session generated by the interface program. The user can then process the error session.

**Axon Interface Monitor**

Failed interface runs - 0002 Alex's test inbound

Intf. ID	Run No.	Start Date	Start Time	End Date	End Time	Status	Descr.	RunTimeStatus	Job Log	Errors	Restart
0002	12	15.11.2002	09:39:50	15.11.2002	09:39:51	08	Interface failed	0			Restart
0002	13	15.11.2002	09:55:24	15.11.2002	09:55:25	08	Interface failed	0	Display		Restart
0002	14	15.11.2002	09:59:17	15.11.2002	09:59:18	08	Interface failed	0	Display	Process	Restart

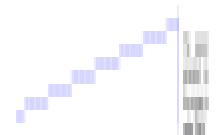
**Batch Input: Session Overview**

Selection criteria  
 Sess.: SM35 From: To: Created by: \*

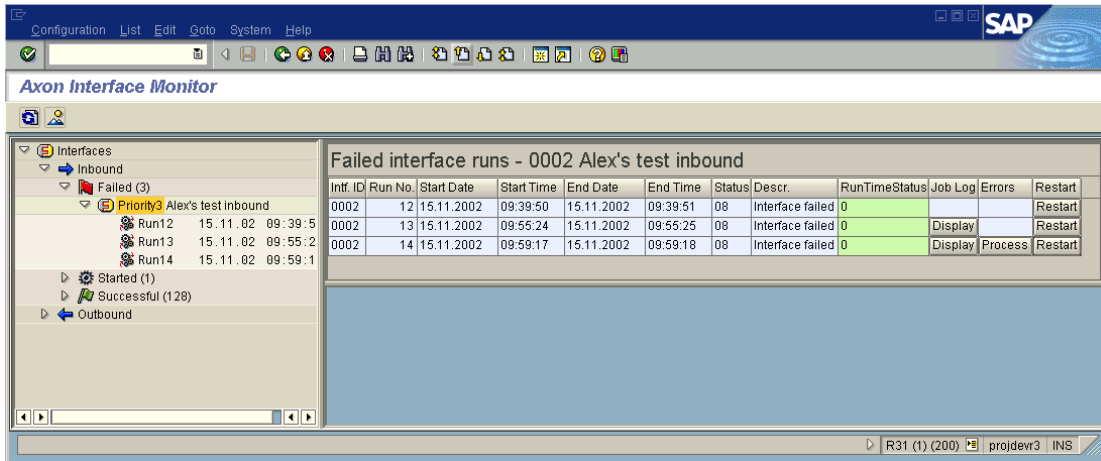
Buttons: New, Incorrect, Processed, In processing, In background, Being created, Locked

Session name	Sta.	Created by	Date	Time	Creation Pro...	Lock date	Authorizat.	Trans.	Trans.	Trans.	Screens	D.	Queue ID
SM35		FEARAL	13.11.2002	14:45:48	SAPMSBOT		FEARAL	1	0	0	2		0211131

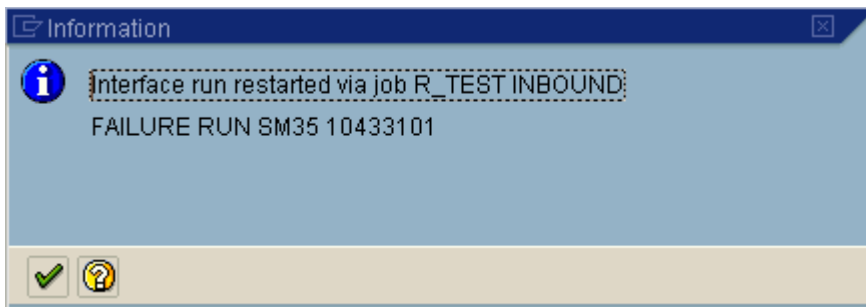
Sessions Found: 1



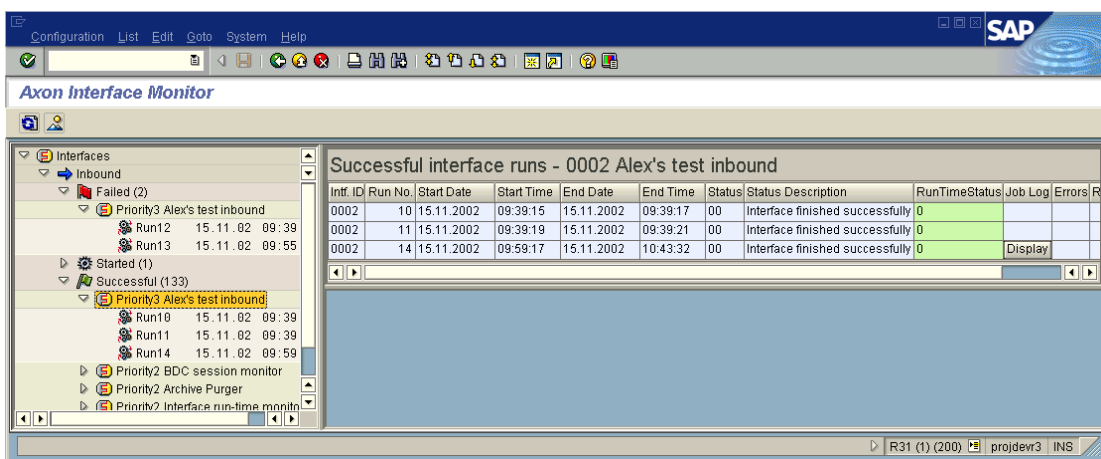
If the interface run failed and the interface is configured to be restartable, then a 'Restart' button will appear in the Restart field. Clicking this button will trigger the restart of the interface with the same variant as the job of the run that failed.



Press the Restart button against the run



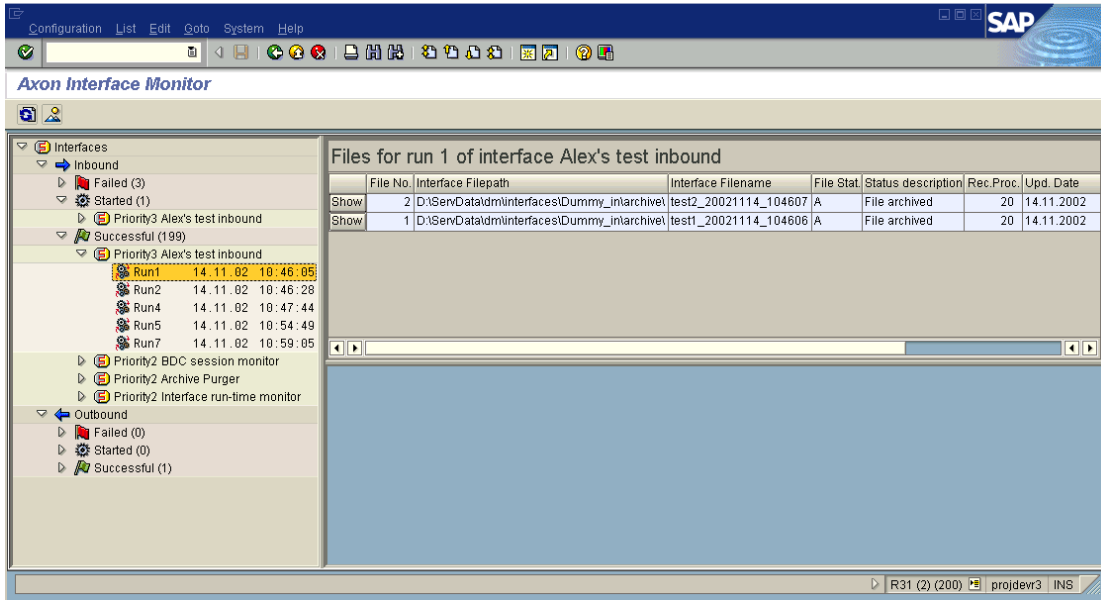
The display will not update itself automatically – the Monitor is refreshed, either by pressing the refresh button at the top of the screen or by restarting the Monitor program. When this has been done the Monitor will show the following.



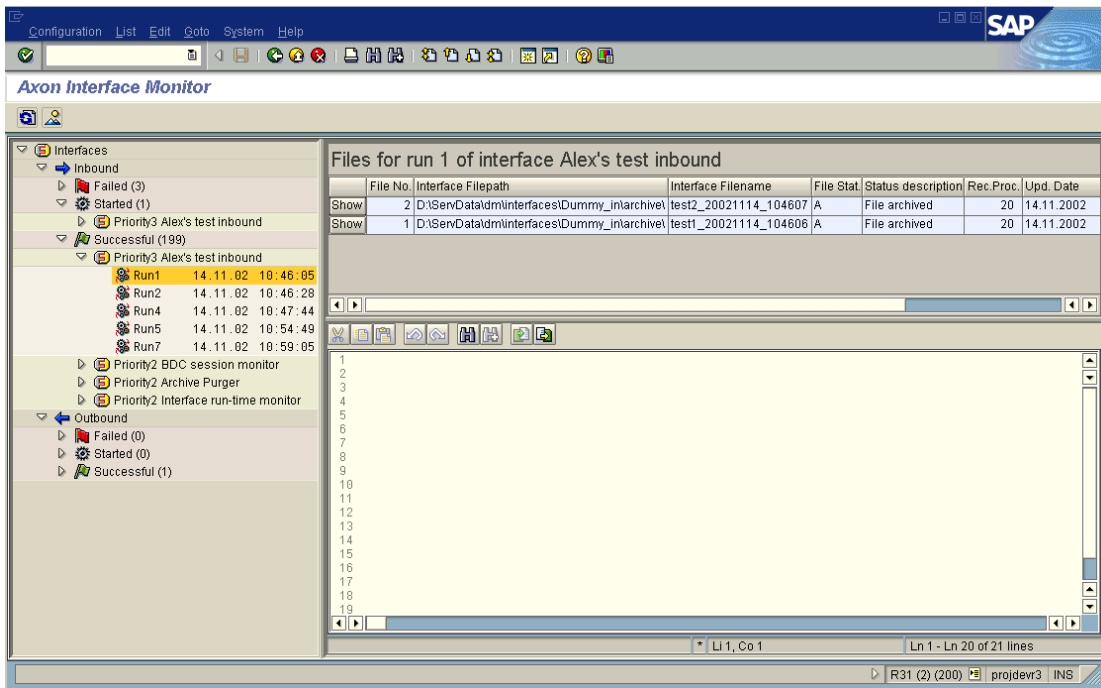
Refresh button

### 14.3.2 File level

Double-clicking on a Run node causes a report of files used by that interface run to be displayed in the upper pane as follows.

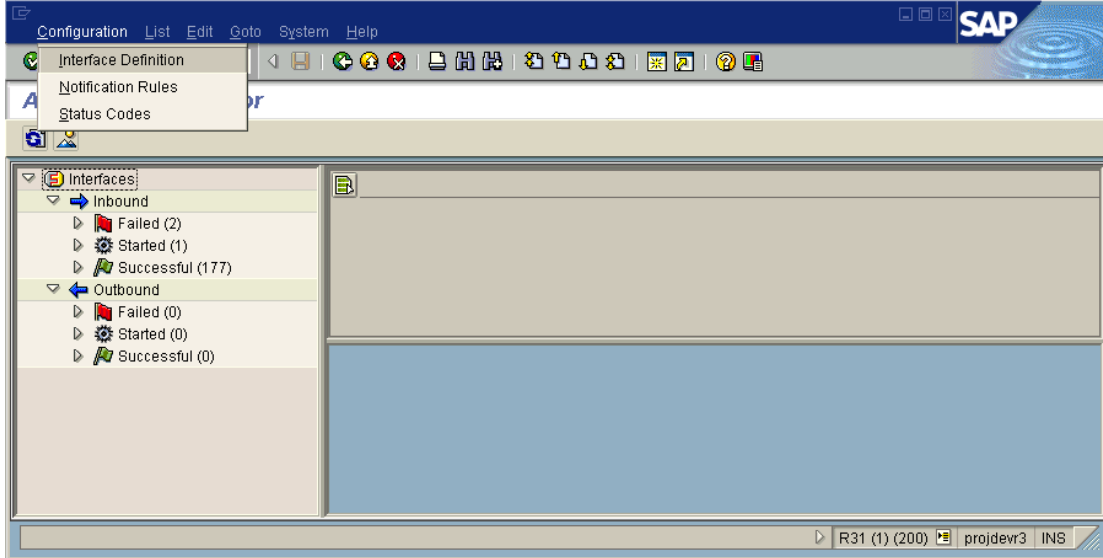


Clicking the 'Show' button against any file causes the contents of that file to be displayed in the lower pane. This can be downloaded to the application server (i.e. a local directory).



## 14.4 Interface configuration

The user can navigate to the configuration views by using the menu options as follows.



These three options allow the configuration of individual interfaces, notification rules, and interface status codes.

## 15 APPENDIX E - Technical Housekeeping Utilities

A number of housekeeping programs are provided with the Interface Management Solution. These are all set up as inbound interfaces, so that they can be seen via the Interface Monitor, although they take no files as input. Rather, they are designed to run periodically in the background in order to keep the information in the history tables up to date or to perform housekeeping functions. The scheduling of these programs must be maintained manually via SM36/37

### 15.1 Interface Run-time Monitor

This interface is a program which checks the run times of interface runs that have a status type of 'Started'. It compares the current run length against the configured expected run time and threshold value. It then sets the run-time status of the interface run accordingly. This interface should run frequently (say every ten minutes) in order that it keeps the run time statuses as up to date as possible.

Program name: ZBCR\_IF\_RUNTIME\_MONITOR

### 15.2 BDC Session Monitor

This program compares any error-holding BDC sessions that might be stored against interface runs against the status of those sessions in the system. It is assumed that the run has been processed successfully if such a session has been processed or deleted. The status of that run is changed to reflect that. This interface should run frequently (say every ten minutes) in order that it keeps the interface statuses as up to date as possible. When a change in status is detected, a notification is sent to the assigned responsible person.

Program name: ZBCR\_IF\_BDC\_MONITOR

### 15.3 Archive Purger

This program deletes history records and their archived files that are older than the retention period defined for a given interface. It should run daily, since the retention periods are defined in days. Note: runs that have no end date are NOT purged. This is to avoid the deletion of runs that are still in progress. Manual housekeeping of these records, should they become large in number, will be required.

Program name: ZBCR\_IF\_PURGE

## 16 APPENDIX F - Application Area

The following table lists the valid Application Area codes. The Application Area is a three character code.

Code	Description
FI	FI CA
BIL	Billing
DM	Device Management
CS	Customer Service
TBD	ICARE
TBD	MTCC
OM	On Market Deregulation
TBD	Basis, Security & FRICEW (Tools)



## 17 APPENDIX G - Useful Table and Structure

The following are tables that are useful in custom developed programs for reference and/or validation purposes.

Table Name	Table Type	Use
DD03L	Trans. Table	All tables in the data dictionary and their fields
NRIV	Trans. Table	Number Ranges
BDCDATA	Structure	Structure of table for batch input processing
BDCMSGCOLL	Structure	structure of table to hold all system messages during a CALL TRANSACTION command
T000	Trans. Table	Client Codes
T100	Trans. Table	System Messages
TADIR	Trans. Table	Directory of transportable developments
TDCT	Trans. Table	Directory of dialog modules
TDEV	Trans. Table	Development classes
TFDIR	Trans. Table	Directory of function modules
TLIBG	Trans. Table	Function groups
TFLIBT	Trans. Table	Function groups short text
TRCL	Trans. Table	ABAP report classes
TRCLT	Trans. Table	ABAP report classes texts
TRDIR	Trans. Table	Directory of reports
TRESE	Trans. Table	Reserved words
TSTC	Trans. Table	Transaction codes
TSTCT	Trans. Table	Transaction code texts
TVARV	Trans. Table	Select variables for ABAP variants
TVDIR	Trans. Table	Directory of table views
DD08L	Trans. Table	List of Check (Value) tables
TFTIT	Trans. Table	Contains all function module names + descr.
TSTCT	Trans. Table	All SAP transaction codes + descr.
SYST	Structure	Int. tbl filled by SAP during runtime. Developers can reference the runtime env. for their progr through the fields of this struc using the "SY-" prefix before the struc field name. There are several useful fields. Most commonly used is "subrc" in "sy-subrc" (give the return value of processes)
SCREEN	Structure	Int. tbl filled by SAP during runtime. Attribs of the current screen-fields may then be accessed. Screen-field contents, however, can only be accessed and changed within a "LOOP AT SCREEN.....ENDLOOP" statement.

## 18 APPENDIX H – ABAP Data Type Details

The following are tables that are useful in custom developed programs for reference and/or validation purposes.

Type	Description	Min. Length	Max. Length	Output Length	Default Justification	Output	Initial Value
C	Character (text)	1	65535	field length	left		<Blank>
D	Date	8	8	8	left		"00000000"
F	Floating point number	8		22		right	"0.0"
I	Integer	4		11		right	"0"
N	Numeric text	1	65535	field length	left		"000...0"
P	Packed number	8	16	2 * field length	right		"0"
T	Time	6	6	6	left		"000000"
X	Hexadecimal	1	65535	2 * field length	left		X"00"
CURSOR	Cursor for database operations						set by the system

When using ABAP's "DESCRIBE" keyword, additional data types are returned. These are as follows:

Type	Description
b	1 byte integer without sign
h	Internal table
s	2 byte integer with sign
u	structure without an internal table
v	structure with at least one internal table

## 19 APPENDIX I – Useful Reports

The following are tables that are useful in custom developed programs for reference and/or validation purposes:

Report Program Name	Description
RC1TCG3Y	Download files from and to file-server/front-end PC.
RC1TCG3Z	Upload files from and to file-server/front-end PC.
RSHOWTIM	Displays tips and tricks for ABAP Programming
RSANAL00	Program analysis
RSABAPSC	Statistical Program analysis to find all ABAP commands In a program
RSTXTRAN	Transfer text between clients with transport number
RSBDCSUB	Initiates Batch input sessions to run automatically when scheduled
RSDYNL10	List all DYNPROs which belong to a specific ABAP
RSTXSCRP	Transports SAPscript with transport number
RSTXCPFS	Transports SAPscript without transport number
RSTXR3TR	Transports SAPscript with transport number
RSTXFINF	List information about a SAPscript form
RSTXFCON	Conversion of paper format for SAPscript forms
RSTXFCOM	Compares two SAPscript forms
RSCLTCOP	Copy tables across clients
RSAVGL00	Table adjustment across clients
RSINCL00	Extended program list
RSORAREL	the Oracle Release
RSUSR006	List users last login
RSTXPFT4	Generates a PDF from any spool

## 20 APPENDIX J – Useful Function Modules

Below is a list of useful functions you may use in your own programs. SAP provides several which you should use as much as possible to avoid redundant code. The functions themselves can be accessed via the Object Repository or the function builder (transaction SE37).

### Batch Input Sessions (BDCs)

bdc_open_group	Creates a new batch input session
bdc_insert	Creates a record in a batch input session
bdc_close_group	Closes the batch input session
bdc_delete_session	Deletes the batch input session

### Text Processing

read_text	Reads text from the specified object into an internal table
read_stdtext	Reads standard text into an internal table

### Date Processing

date_compute_day	Determines the weekday as a numeric value (1-7) from a date specified. E.g. 01.01.2000 gives as result: 6 (i.e. Saturday)
date_convert_to_factorydate	Determines the factory calendar date from a date specified
date_get_week	Determines the week of the year given a specified date
easter_get_date	Determines the Eastern date given a specified date
factory_convert_to_date	Determines date given a factory calendar date
holiday_check_and_get_info	Determines if a given date is a holiday
fkf_dte_add_month	Adds a number of months to a date. Additional features for working days are available.
week_get_first_day	Determines the first day of the week for the week a specified date falls upon

### Popup Windows

popup_to_fill_command_line	Displays a pop-up window for user command input
popup_to_modify_text	Good to enter 1 line text. (Fig.1) answer will be J/A. Value1 contains the text entered.
popup_with_table_display	To display dialog with single column values to pick from (fig.2). 4 params are for dialog coordinates, 1 param for the text in titlebar and itab valuetab for the list of values to display. Picked choice is in choice. Will have value 0 if cancelled.
popup_to_confirm_data_loss	Will popup the standard message(see fig.3) Titletext is alterable. Answer contains J/N
popup_to_confirm_loss_of_data	Will popup a customisable message (see fig.4) Answer contains J/N
popup_for_interaction	See fig.5. The field ticon is the icon displayed. Can be Q(uestion), I(nformation), E(xclamation)

popup_to_confirm	See fig.6. Buttons + icons + question/title-text are customisable. Cancel button can be hidden.  popup_type can be like icon_message_information or the like. it is the message icon. Answer will contain 1,2 or A (i.e. cancelled).
popup_to_confirm_step	Same as above but is fixed (Yes, No, Cancel)
popup_to_decide	See fig.7. Textlines, button texts, titletext are customisable Cancelbutton can be hidden.
popup_to_get_value	Simple dialog popup to enter data (fixed format) See Fig.8.Entryfield format is related to a field in a table (need to be specified). Default value can be specified.
popup_to_get_one_value	Same as above but field length is alterable. Also this field is independent of a database field. See fig.9
popup_get_values	Dialog box for display and request of values from tables(s) in DB, without check. See fig.10,11 and 12.

**File Access**

ws_upload	File transfer from presentation server file to internal table. To read a file from workstation for processing.
upload	Same as ws_upload but with dialog to specify name of file to transfer.
ws_download	File transfer from internal table into presentation server file. To export/create a file (from program) to workstation.
download	Same as ws_download but with dialog to specify name for file.
f4_filename	F4 for filename / Filemanager support to locate file in a directory.

**Database Access**

isu\_db\_\*\*\*\_single Replace the stars with any db name and then you can use this fm to get data from transparent tables.

**Financial**

Fkk_show_payment_usage	Show for CA the debit/credit/open items in dialog box on the screen. Can be used for applications to show box.
Fkk_complete_account_read	Use this to get all items for a CA. If flag XACCU is set, it does not return cleared items.
Fkk_open_items_for_acc_read	Use this to get all open items for a CA.
Bapi_account_getbalances	Useful to get short overview of Acctn. Bal. (given CA & Business Partner).
Fkk_get_cleared_items	Gets all cleared items for CA.
Fkk_bp_line_items_sel_by_opbel	Gets all items for specific document.

**Installment Plan**

fkk_s_instplan_display	
fkk_s_instplan_provide	

fkf\_s\_instplan\_existence\_check  
 fkf\_s\_instplan\_getdetail  
 fkf\_s\_instplan\_getlist  
 bapi\_instmntpln\_createfromdata  
 bapi\_instmntpln\_getdetail  
 bapi\_instmntpln\_getlist  
 BAPI\_ISUFINDER\_FINDOBJOFDATA

**Dunning**

fkf\_read\_dunning\_history  
 fkf\_read\_dunning\_history\_level

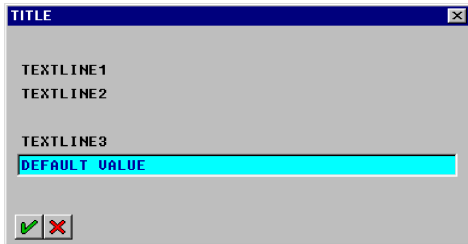
**Miscellaneous**

authority_check_dataset	Check file access authorisation. Are you allowed to Read, Change or Delete a file.
RZL_sleep	Gets application to sleep (wait) for a number of seconds.
aip9_status_read	Reads system/user status from any object in SAP db
sapgui_progress_indicator	Set 'Progress Indicator' (meter) in Current Window's statusbar. Incl. title to specify.
ws_execute	Execute an external program on the presentation server (your machine). Just fill in e.g. 'c:\windows\notepad.exe' in field program.
sxpq_command_execute	Executes external (UNIX) command.
Isu_address_provide	You pass in the address type with its associated key and the function module will return the address. For example, if you specify that the address type is 'C' for connection object and give the object number, the function module will return the connection object address.
Isu_structure_read	If you know one end of the master data chain (ie. Contract) and you need another end (ie. Premise), just specify the path to follow and give the starting key. For this example, the path is contract to installation, installation to premise (EVER to EANL, EANL to EVBS) and the starting point is the contract number 1000045. The function module will return the associated premise number.
ISU_S*	Is the naming convention for a series of function modules that will perform some type of service like displaying entries, updating a table etc.
CLSE_SELECT_AUSP	Used to get characteristic values from the classification system especially in the meter reader download form. Is using classification on the equipment record for such items as ERT frequency and tone.
RSPO_SX_OUTPUT_TEXTDATA	Spools data to a Logical Output Device (e.g. for transfer to (other) UNIX box or just for print from within an ABAP.
ISU_PRINT_EXPANDED	Does a reprint of a bill. Same as EA60 but then can be called from ABAP (e.g. when you need to do mass re-print).
GUI_GET_DESKTOP_INFO	ets info about the front-end

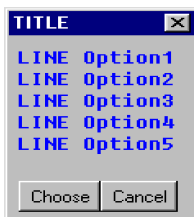
SAPSCRIPT_IMPORT_GRAPHIC_BDS	Upload of graphic file into SAP repository (same as transaction SE78).
GET_COMPONENT_LIST	Returns the field names of a given structure or internal table + data type.
GET_TEXT_PRINT_PARAMETERS	Pops up the print dialog
VIEW_MAINTENANCE_CALL	Call the maintenance transaction program which was created by the table maintenance generator. Pass Action Code (e.g. "U" for update) and the table name (as if you were in SM30).
RSPO_FRONTEND_PRINTERS_FOR_DEV	Retrieves a list of printers defined on the local machine, also tells which on is set as default printer.
FTP_CONNECT FTP_R3_TO_SERVER and FTP_SERVER_TO_R3	Connects to FTP server. Can use in combination with to transfer files from and to (SAP / UNIX)
CONVERT_ABAPSPoolJOB_2_PDF	Converts spool output to PDF
SO_NEW_DOCUMENT_ATT_SEND_API1	Use this to send anything to fax/email/pager etc. Provided that the SCOT settings are ok (done by BASIS)
RSEC_GENERATE_PASSWORD	Generates a random number of any length
EPS_GET_DIRECTORY_LISTING	List files from UNIX dir

Screenshots For Popup Function Modules

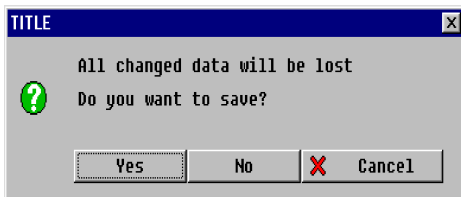
popup\_to\_modify\_text (fig.1)



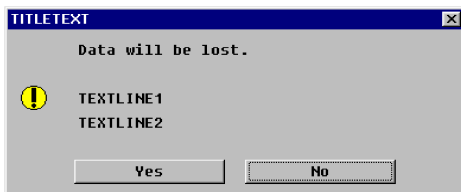
popup\_with\_table\_display (fig.2)



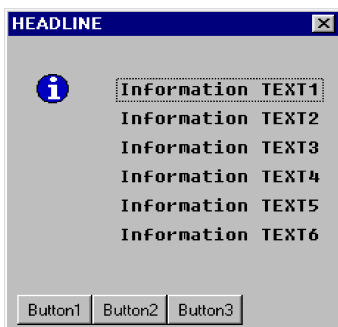
popup\_to\_confirm\_data\_loss (fig.3)



popup\_to\_confirm\_loss\_of\_data (fig.4)

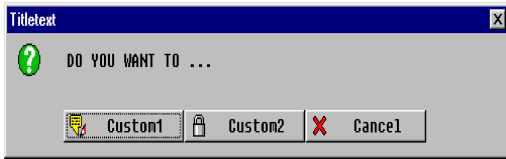


popup\_for\_interaction (fig.5)

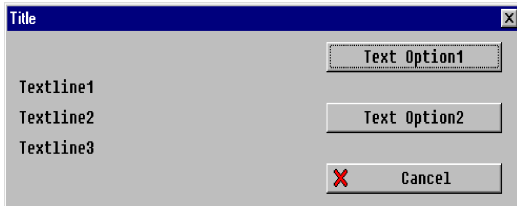




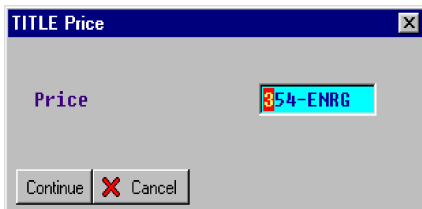
popup\_to\_confirm (fig.6)



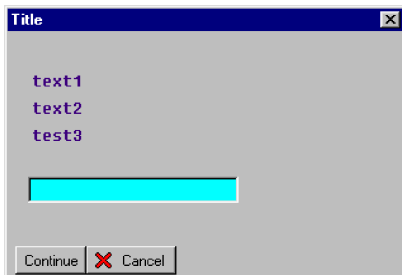
popup\_to\_decide (fig. 7)



popup\_to\_get\_value (Fig.8)



popup\_to\_get\_one\_value (Fig. 9)



popup\_to\_get\_values (Fig. 10)

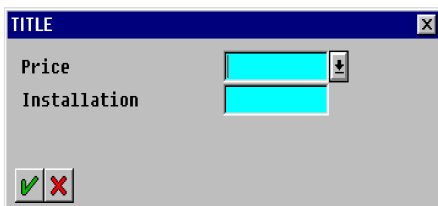


Fig.11

Fig.12

Price	PCat	Price lev.
101-CAP	2	
101-ENRG	1	
102-ENRG	1	
104-ENRG	1	
105-ENRG	1	
106-CAP	2	
106-ENRG	1	
117-CAP	2	
121-CAP	2	
121-ENRG	1	
123-CAP	2	
123-ENRG	1	
126-CAP	2	
130-CAP	2	
130-ENRG	1	
131-ENRG	1	
132-ADD-CH	2	
132-CAP	2	
132-ENRG	1	
133-ADD-CH	2	

Installation
Instal1.
1
2000000000
2000000001
3000000000
3000000001
3000000002
3000000003
3000000004
3000000005
3000000006
3000000007
3000000008
3000000009
3000000010
3000000011
3000000012
3000000013
3000000014
3000000015
3000000016

## 21 APPENDIX K – Useful Transactions

Transaction	Description
SE11	Data Dictionary
SE30	Runtime Analysis
SE32	Text Elements
SE36	Log DB
SE37	Function Modules
SE38	Editor
SE54	Create Main Prog for View
SE71	SAPScript Form
SE91	Message Class
SE93	SAP Transaction Codes
SA38	Running Program in background
SM02	Remove locks from USER objects
SM04	Running Transactions + Easy kill
SM30	Table Maint. Trans (Also SM31)
SM35	View batches / BDC Recording
SM50	Background job processing
SM51	Running Processes
SU01	User maintenance
SU53	Author. Screen (Print after Auth. Fail)
SO10	Text Objects
SO01	Mail Inbox
EA22	Display billing documents
EA30	Rates (also EA31, EA32)
EA35	Schema (Also EA36, EA37)
EA40	Display Print Documents
EA47	Discounts/Surcharge (Master Data)
EA50	Operands (Also EA51, EA52)
EA53	Rate Category (Also EA54, EA55)
EA56	Maintain Rate Types
EA60	Budget Billing Plan (Also EA61, EA62)
EG02	Device Category
EG31	Final Connection (Device -> Location)
EG33	Technical Installation of device
EG34	Billing installation of device
EG71	Display Rate Data
E41H	Create MRU
EL40	Street Route for MRU
ES30	Create Installation
ES55	Create Connection Object
ES60	Create Premise
ES65	Create Device location

EFCS	Form Class
DFRM	Application Form
CAA2	Contract Account Change
IQ02	Material Change
IW52	Notification
TJ30	User Status
SARA	Mass archiving of SAP objects
SLG1	Evaluate Application Log
SPAU	Modify-Adjust (ABP, Interf, Scr, Matchc, view, lock obj)
SPDD	Modify-Adjust (Domains, data elements, tables)
STUN	System Monitoring
SNRO	Number Ranges
SWLD	Workflow Maintenance Screen
SPRO	IMG Customising
DB20	Update Statistics

## 22 APPENDIX L – Packages

The following packages have been setup:

Sr.No	Work-stream	Package	Package name
1	BI	ZBI	Billing related developments (not migration or interfaces)
2	FI	ZFICA	FICA developments (not migration or interfaces)
3	DM	ZDM	DM developments (not migration or interfaces)
4	CS	ZCS	Customer service developments
5	OM	ZOM	On Market Deregulation developments
6	ICARE	TBD	ICARE developments
7	MTCC	ZMTCC	MTCC development
8	BW	ZBW	BW developments
9	CRM	ZCRM	CRM development

## 23 Bibliography

Give below are links are documents, which would be useful for further reading and information.

## 24 Performance Standards

Link to SAP performance URL <http://service.sap.com/performance>

Documents for further reading,



Efficient Database Programming.pdf



Efficient Database Programming\_exercis



"Efficient Database Programming with A



Memory Efficient ABAP Programming.p



Memory Efficient ABAP Programming\_e



Next GenerationABAP Perf



State-of-the-Art ABAP\_exercises\_solu



State-of-the-Art ABAP\_print.pdf



Test-Driven and Bulletproof ABAP.pdf



Test-Driven and Bulletproof ABAP\_exe



"TechEd 2004 Performance Analysis



"TechEd 2004 Quick Start on Performance



"Performance Analysis in a nutshell

Introduction to SAP ALV



5dc3e690-0201-0010-1ebf-b85b3bed962

Other important link [www.sdn.sap.com](http://www.sdn.sap.com)

*There are number of eLearning courses related to ABAP development in eLearning section.*