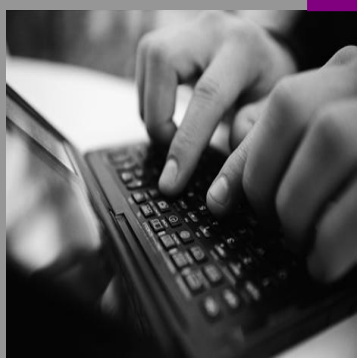
How-to Guide
SAP NetWeaver 2004



How To… Tips and Tricks for Developing High-Performance, Stable MI Applications

Version 1.00 – May 2006

Applicable Releases:
SAP NetWeaver 2004
(Mobile Infrastructure)

Contents

1 Introduction

This document offers tips and tricks to help develop mobile applications based on SAP Mobile Infrastructure.

Aim of this Document

The aim of this document is to provide developers with information about how to write mobile applications with particular focus on:

- Stability
- Performance
- Transactional security
- Cost of Ownership
- Maintainability
- Scalability

Target Group

This document has been written for developers of mobile applications based on SAP Mobile Infrastructure.

Prerequisites

You should be familiar with the following technologies:

- ABAP (Middleware/Back-end system)
- Java (Client)

You should also be familiar with the Mobile Infrastructure concepts.
The structure of this document is based upon the different phases of the project:

- The design/planning phase where you design your future implementation.
- The development phase where you implement your design and test the functional requirements.
- The packaging and deployment development phase where you package your application code in accordance with your deployment strategy and prepare for rollout.

2 Design Phase

2.1 Required Skills

The design phase requires the support of different groups and individuals:

- Business department
- Developers
- User Interface Designer

The business department are required to share their knowledge about the company's business processes, while the other stakeholders have to design the user interface and create the data model.

The developers need to have a strong understanding of the technologies and synchronization methods in Mobile Infrastructure.

2.2 Tools

2.2.1 Data Model

SAP does not deliver tools to support the design phase of an MI-based application development. However, you can use an application or tool you are already familiar with to do this. Create your application model/data model with a tool such as MS Visio Borland Together or by simply designing it on paper.. Once the application's model has been designed, an MI middleware and a back-end expert has to translate this model into SyncBOs.

2.2.2 View

The user interface should be designed for each and every screen of the application before you start to develop it. This rule applies to both JSP and AWT applications.
To create the layout for your screens, you can use a graphic program like Adobe Photoshop.
If you create a JSP application, you can create a first prototype of the UI for your developers using a tool such as InDesign or Dreamweaver.

2.3 Synchronization: SmartSync vs. Generic Sync

MI SmartSync is the future of MI synchronization. No more major development is planned for Generic Sync.
Nevertheless, it is still a good idea to consider Generic Sync as an option for your scenario, as it allows you to build a synchronization layer that fits your application perfectly, and that does not have the overheads of SmartSync (since SmartSync is required to suit all types of applications). However, choosing the Generic Sync option means you have to perform all the necessary tasks yourself, including creating a delta sync mechanism, reading and writing data, ensuring transactional security, supply server side monitors, and so on. SmartSync offers all of those features for free.
Although you can develop a fast and superior synchronization layer with Generic Sync, we recommend using SmartSync because SAP MI development will be focused on this area in the future.

2.4 SmartSync: 2-Way, Timed-2-Way or Server-Driven?

You can choose from several SyncBO types. In the following sections, we describe the advantages and disadvantages of the different types of SyncBOs and when you should use them.

2.4.1 2-Way

A typical 2-way SyncBO is used if you need to have the most up-to-date data on the client, and your data changes frequently. A typical example for this would be an order. When the user synchronizes after new orders were created in the back end, these orders need to be visible on the client.
The problem with this type of SyncBO is that every synchronization process has a huge impact on the back-end system because RFC connections have to be established between the middleware (MW) and the back-end system. As a result, the 2-Way SyncBOs should only be used when it is absolutely necessary.

Advantage	Disadvantage
Data is always up to date	High load on the back-end system and problems with reprocessing after errors

2.4.2 Server-Driven

With a server-driven SyncBO, all the changes required on one of the clients are sent directly to the middleware. This data is available on the client, therefore, after the next synchronization process.
The problem with T51 SyncBOs is that this replication can be time-intensive. In addition to this, the MEREPLIC_RESTART job must be planned in a short space of time, and, especially, if a large number of changes have been made to the SyncBOs, the load on the back-end system is extremely high. If you make considerable changes to many items for a particular SyncBO it is important to switch the replication off. This can be done as the server-driven SyncBO is based on the timed 2-way SyncBOs. If changes are only made infrequently, and the change does not have to appear on the client immediately (but instead after a few minutes) the server-driven technology can be used.

Advantage	Disadvantage
Less load on the back-end system in comparison with the 2-way SyncBOs. Server-driven SyncBOs also make sense if the number of items changed is too great for the timed 2-way SyncBO technology to run efficiently (the server-driven synchronization only affects the MW).	High load on the back-end system if a large number of changes are made.

2.4.3 Timed 2-Way

The timed 2-way SyncBO is the basic and standard SyncBO. A batch schedule is responsible for the data replication.

The problem with this type of SyncBO is that it always replicates the complete dataset when comparing each line item. This takes time, especially if you have a large number of objects and items.

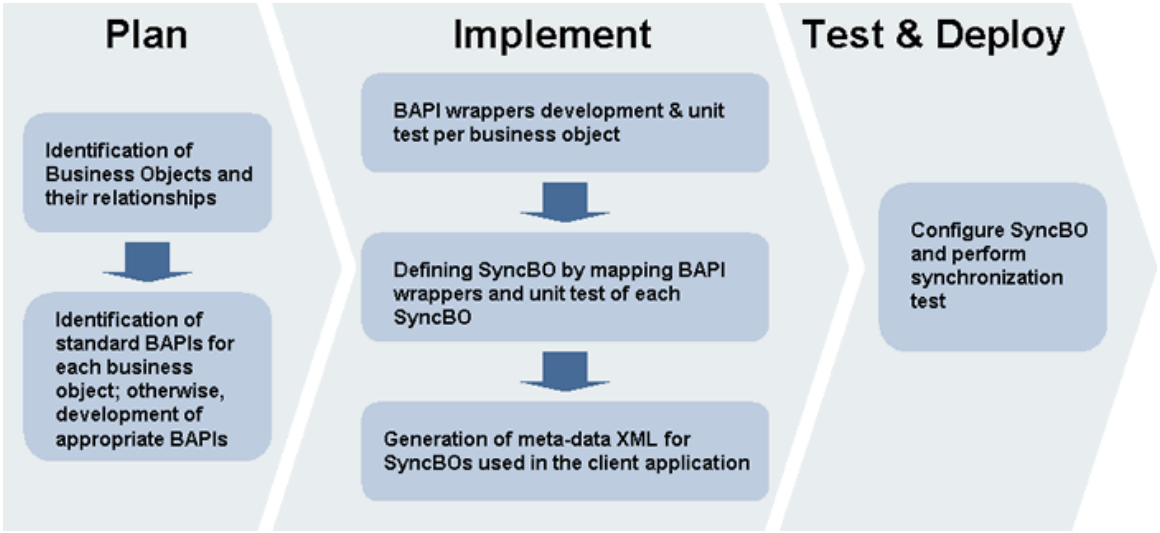
The benefit of this type is that you can schedule this replication during off-peak business hours and create different schedules based on the type of business object you need to replicate. For example, you can schedule master data replication on a once-per-month basis.

Advantage	Disadvantage
Saves system resources during peak business hours and synchronization process only affects the MW	MW-resource intensive. Back-end changes are only available on the client after the scheduled replication

2.5 SyncBO Model

The SyncBO design has a critical impact on performance and stability. A typical SyncBO consists of a single top structure and several line items. In a top structure, we store all the data that is necessary for the business process and that only changes infrequently..
Data is stored in line items if it is optional and it is possible to have multiple sets of the business object (for example, one order can have one or more partners associated with it).
It is important to note that cascading is only possible from **top** structure to **top** structure. It is not possible to check the cascading relations from a line item to a top structure of from line item to line item. The cascading check can only work properly if we define a link in the top structure to another top structure,.
Although the development of MI applications does not differ significantly from typical application development, the following points are important:

- When starting to design the SyncBOs, in addition to the fields required on the client, we must consider what fields are available in the business objects in the back-end system.
- Always think about the limited resources on the client; even a laptop is not as powerful as a back-end server with gigabytes of main memory and several CPUs. We need to limit the amount of data on the client to the absolute minimum.
- If you develop the client and back-end system at the same time, remember that objects in the back end and on the client influence each other. Even at this early stage in the development, try to avoid data conflict situations during design time.
- Plan your development in small iterative steps. Focus on performance and data load issues from the very start of the project. The earlier you find problems with performance or in the data model, the more time you have to fix them.
- All data from the client is directly written to the back end. If a conflict occurs because the client and back end have both changed the data since the last synchronization, the data on the client is directly replaced with the data from the back end. In a normal synchronization process, you get a reply from the back end as well if you request data. It may be that the request takes too long in the back-end system. In this case, you can switch to synchronous synchronization. This means you will wait longer for the timeout, and you are more likely to get the data in the same synchronization run.



2.6 The User Interface

You should always know the user for whom you are creating the application. Is it a sales representative who is comfortable with a PC? Or is it a service technician who usually works with a hammer and a drill?

Adapting to a new software product is not normally a challenge for IT people, but it can be particularly difficult for people who do not normally work with computers. The application, therefore, should be created in such a way that it can be used by technologically unskilled users as well.

With PDAs in particular, you should make sure that the text size and the button size of is not too small. At the very least, you should be able to use the application by means of the touch screen and your finger.

Invest enough time and thought into the design of the user interface and schedule prototyping and review sessions. This will save you time in the long term.

2.7 Framework: AWT vs. JSP

The Mobile Infrastructure offers two choices for the client side user interface: AWT and JSP. The technologies are not compatible, meaning you can not run MI JSP and MI AWT applications on the same device.

The following tables contain the advantages and disadvantages of both technologies:

AWT

Advantage	Disadvantage
• Native UI • Faster than JSP • "Richer" UIs possible • Can be combined with Swing or SWT on laptops	• Not compatible with SAP standard applications

JSP

Advantage	Disadvantage
• Browser-based UI • More familiar top most developers • Compatible with SAP standard applications	• Slower than AWT

Our recommendation is that you should always use AWT as long as you do not plan to run a standard SAP JSP application on the same client.

2.8 Hardware: Laptop vs. PDA

Whether a laptop is more suitable than a PDA or not depends on your business case. Laptops obviously offer better performance, and allow you to create a more complex UI because of the screen size.

However, laptops need to be booted or woken up from suspend mode, which can take time. PDAs are accessible immediately when required. In contrast to laptops, PDAs can be carried around easily in your pocket and can be taken to places where a laptop would be too big (for example, into a machine that needs repairing).

If you develop a laptop application and you plan to port it to the PDA MI runtime later, make sure that your code is compatible with Java 1.1.8. If this is the case, you might only need to change the UI and your controllers. If not, you will need to spend a lot of time rewriting the business logic.

2.9 Persistence: File I/O vs. DB2e

SAP Mobile Infrastructure offers two supported persistence layers: File I/O and DB2e. With file I/O, the data will be stored using the file system of the client. The DB2e persistence layer stores the data in a local installation of IBM's DB2e database.
With a small amount of data, file I/O can be faster, but the bigger the data volume gets, the more DB2e is recommended.
With regards to stability and transactional security, DB2e is the better choice. The only reason to choose File I/O would be the financial aspect: For DB2e you need to pay an additional license fee. However, DB2e may save you money in the long term.

3 Development Phase

3.1 Back End

3.1.1 Required Skills

When you create a mobile application, application development can decide how to create the data packages for synchronization. The following options exist:

3.1.1.1 Generic Synchronization

Generic Synchronization can be implemented using Java APIs for the parameterization of function module calls in the back-end system. Parameterization allows the developer to define the precise amount of data to be transferred during synchronization. Function modules can be called after a “wrapper function” is generated by the MI Wizard (to find out more about the MI Wizard, refer to the documents included in the Mobile Development Kit (MDK), available on the SAP Developer Network).

3.1.1.2 Smart Synchronization

Smart Synchronization is a synchronization technique specially adapted to meet the requirements of complex mobile applications. Once application development has defined the data models for synchronization, the system administrator can explicitly define what data is to be synchronized. Only the BAPI wrapper function modules can be called on the back-end system.

A BAPI wrapper is a function module similar to a BAPI that is defined by following a certain set of rules. Conceptually, a BAPI wrapper plays the role of a private method of a business object (SyncBO) and resides in a back-end system. A SyncBO in Smart Synchronization calls the BAPI wrapper on the basis of a request from a mobile device.

It is important, therefore, that each BAPI wrapper demonstrates the parameters and behavior expected by Smart Synchronization. Developing MI applications on a back-end system based on MI involves writing function modules following certain rules. The developer must have a good understanding of MI and enough experience to write ABAP function modules.

3.1.2 BAPI Wrapper Development

For each business object (SyncBO), up to five BAPI wrappers can be created and used:

BAPI Wrapper	Description
GetList	Returns a list of business object header data on the basis of the selection criteria specified in the Import parameters.
GetDetail	Returns one header data as an Export parameter out of the header data list on the basis of the object key(s) specified in the Import parameters. It also returns one or more list(s) of item entries associated with the header data.
Create	Creates a single business object, and returns object key(s).

Change	Modifies the header and/or items of the specified business object. The function needs to replace the entire item data with the entries of Table parameters.
Delete	Deletes the specified business object (including the items).

It is very important that the BAPI wrappers are implemented in accordance with the rules required by MI (as described below). Violating these rules may cause unexpected runtime errors which are difficult to identify and solve.

3.1.2.1 General Rules

Follow the BAPI development rules (http://service.sap.com/bapi).

3.1.2.2 BAPI Wrapper-Specific Development Rules

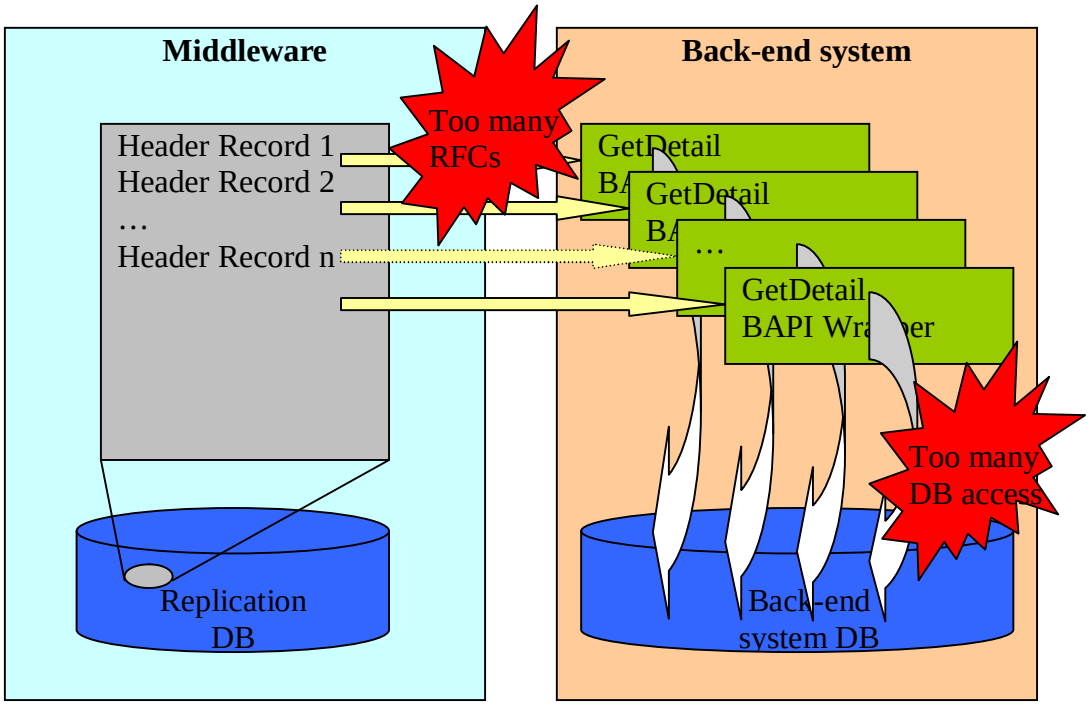
- Processing type of a function module must be remote-enabled module.
- Parameter called “RETURN” where the type is BAPIRET2 structure needs to be defined as either Export or Tables parameter.
- Parameters can only refer to either a structure or a field of a structure (<structure>-<field>).
- Changing parameter CANNOT be used.
- Exceptions CANNOT be used.
- Parameters CANNOT refer to structures that include other structures; the reference structure needs to be a flat structure, and fields in the structure CANNOT refer to types.
- Consistency within the business object must be ensured by appropriately defining Export/Import/Tables parameters with five types of BAPI wrappers.
- “Commit Work and Wait” needs to be executed in the update BAPI wrappers (Create/Change/Delete).
- GetList BAPI Wrapper: No header records should be returned with object key (r3key) as initial value.
- GetDetail BAPI Wrapper:
  - Should not return initial object key (r3key) for header
  - Should not return item rows with duplicated object key
  - Header structure (export) object key should always be the same as the object key of the import parameter
- Create BAPI Wrapper: Error handling should be considered.
- Modify BAPI Wrapper:
  - Parameter of tables (item rows) should contain all items
  - Error handling should be considered
- Delete BAPI Wrapper: Throw error message when the record does not exist in back-end system.

3.1.3 Performance Tips

3.1.3.1 Call GetAllDetails BAPI Wrapper Instead of Calling GetDetail BAPI Wrapper

What is the performance problem?

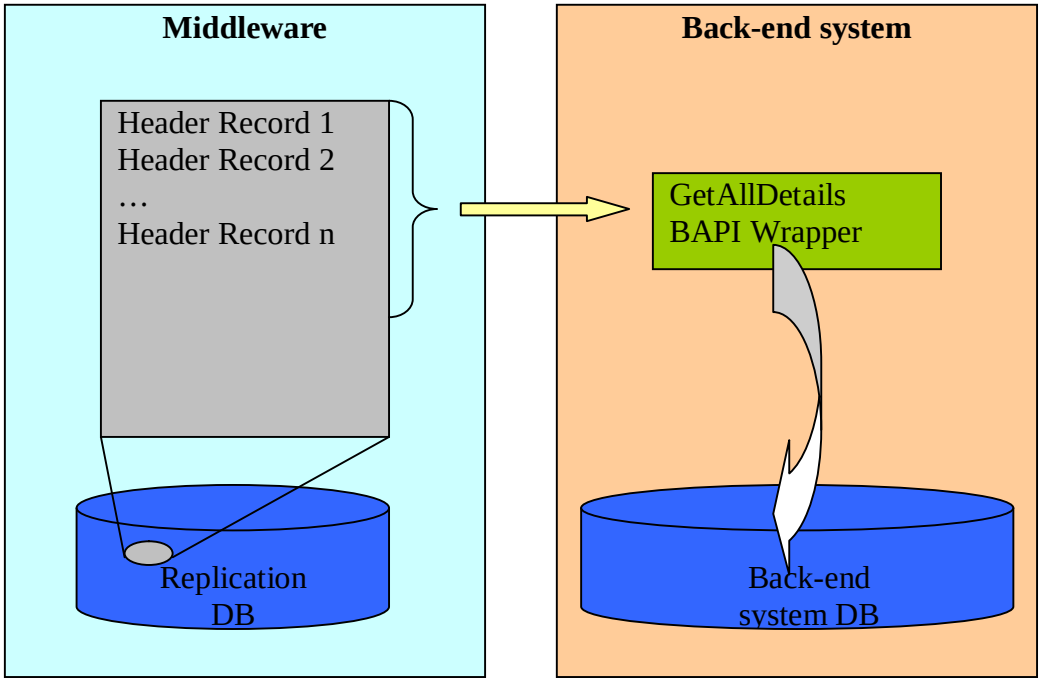
When replicating the data from the back-end system to the middleware, the standard behavior of middleware is to call GetDetail BAPI Wrapper for each header record that exists in the RDB. The following graphic depicts the calling process. Because the call is made for each header record, if the data volume of header record is large, too many RFC calls between middleware and back-end systems will be made. Similarly, too many DB access attempts to the back-end system DB will be performed.



Standard behavior of data replication from back-end system to middleware

What to do

The solution to the above mentioned performance problem is to call the GetAllDetails BAPI Wrapper for all header records instead of calling GetDetail for each header record. After calling GetList BAPI Wrapper, the middleware only needs to call the back-end system once to get all the item tables for all the header records that exist in the replication DB. The DB access to the back-end system table can be minimized because the item tables can be retrieved from tables filtered by all the header records. The following illustration describes the new calling process.



Call GetAllDetails BAPI Wrapper instead of calling GetDetail BAPI Wrapper

As the GetAllDetails BAPI Wrapper is not supported by the standard MI implementation, the above solution has to be handled by the application to implement user exits of SyncBOs.

Below you can find sample codes for 2-way SyncBOs..

SyncBO type: S01

Function module type: S01

Exit Key	Sample Codes

200	`*  Additional declaration for GETDETAILS Bapi Wrapper`
	`data:`
	`  ldt_delivery  TYPE STANDARD TABLE OF`
	`  LTS_I01E_E_DELIVERY_HEADER,`
	`  lds_delivery  TYPE LTS_I01E_E_DELIVERY_HEADER,`
	`  ldt_010_details  TYPE STANDARD TABLE OF lts_i01t_t_delivery_010,`
	`  ldt_e001_details  TYPE STANDARD TABLE OF lts_i01t_enhancement_001,`
	`  ldt_e002_details  TYPE STANDARD TABLE OF lts_i01t_enhancement_002.`

399	`IF NOT i_inbox_data[] IS INITIAL.`
	`  APPEND lds_syncwl  TO e_upload_syncwl.`
	`  READ TABLE ldt_h_from_mbl INTO lds_h_from_mbl INDEX 1.`
	`  PERFORM dsd_deliv_build_upload_data`
	`  USING  lds_syncwl`
	`  lds_h_from_mbl`
	`  ldt_p_from_mbl`
	`  CHANGING e_upload_data[] .`
	`  PERFORM dsd_deliv_end_of_downloader`
	`  USING  e_conflict_r3data[]`
	`  e_conflict_rtn[]`
	`  e_upload_data[]`
	`  ldc_badi_parms`
	`  CHANGING ldc_sync_generic`
	`  e_download_data[] .`
	`  EXIT.`
	`ENDIF.  "IF NOT i_inbox_data[] IS INITIAL.`

430	`  LOOP AT ldt_header`
	`  ASSIGNING <l_header> .`
	`*---------------------------------------------------------------`
	`* Fill ldt_delivery (lds_delivery-delv_no)`
	`*---------------------------------------------------------------`

```
       lds_delivery-delv_no = <l_header>-ktop-delv_no.
       APPEND lds_delivery TO ldt_delivery.
*---------------------------------------------------------------------
* End of ldt_delivery entry creation
*---------------------------------------------------------------------
      endloop.
*---------------------------------------------------------------------
* Make a call to GetDetails function using the keys returned
* in GetList (could be moved below after filtering check)
*---------------------------------------------------------------------
      IF NOT ldt_delivery[] IS INITIAL.
        lds_log_parameters-jobtype  = gcf_bapi.
        lds_log_parameters-jobname  =
        'GETALLDETAILS'.
        CALL FUNCTION 'MEREP_OBJECT_START_END'
          EXPORTING
            operator       = gcs_object-start
            log_parameters = lds_log_parameters
          IMPORTING
            return         = lds_return.
        IF lds_return-type <> gcs_rc-success.
          RAISE failure.
        ENDIF.

        CALL FUNCTION 'GETALLDETAILS'
          DESTINATION ldc_sync_generic-rfcdest
          IMPORTING
            return                = lds_ie__return
          TABLES
            it_delivery_header    = ldt_delivery
            enhancement_001       = ldt_i___enhancement_001
            enhancement_002       = ldt_i___enhancement_002
            et_delivery_010       = ldt_i___t_delivery_010
          EXCEPTIONS
            communication_failure = 1  MESSAGE ldf_msg_text
            system_failure        = 2  MESSAGE ldf_msg_text.
        ldf_smapi_subrc = sy-subrc.

        ldt_010_details[]  = ldt_i___t_delivery_010[].
        ldt_e001_details[] = ldt_i___enhancement_001[].
        ldt_e002_details[] = ldt_i___enhancement_002[].

        CALL FUNCTION 'MEREP_OBJECT_START_END'
          EXPORTING
            operator       = gcs_object-end
            log_parameters = lds_log_parameters
          IMPORTING
            return         = lds_return.
      ENDIF.
*---------------------------------------------------------------------
* End of GetDetails call
*---------------------------------------------------------------------
```

```
if 1 = 0.
```

```
endif.
ldt_i___t_delivery_010[]  = ldt_010_details[].
ldt_i___enhancement_001[] = ldt_e001_details[].
ldt_i___enhancement_002[] = ldt_e002_details[].
```

```
*  Process for corresponding TOP record
   check <L_I___T_DELIVERY_010>-delv_no = ldf_ii__i_delno.
```

Important

It is important to note that our recommendation to call GetAllDetails BAPI Wrapper does NOT mean the old approach of calling GetDetail BAPI Wrapper is obsolete. GetDetail BAPI Wrapper is essential other circumstances and it has to be implemented because it is still referenced by the middleware.

Furthermore, the solution may or may not require additional work to adjust the implementation in the user exits of SyncBOs when upgrading to a higher version of MI or applying a new Support Package of MI on the middleware. This can be established with the help of a MI consultant.

3.2 Middleware

3.2.1 Required Skills

As with the back-end system development, the development of the middleware differs depending on which synchronization method you use.

3.2.1.1 Generic Synchronization

No development is necessary on the middleware, but some tables will need to be configured to call the function modules on the back-end system.

3.2.1.2 Smart Synchronization

A SyncBO can be considered a business object that has some public and private attributes, as well as some private methods. From a design perspective, a SyncBO is metadata that

1. Defines the business object model of an application
2. Defines the way data is exchanged between mobile devices and the back-end system
3. Defines how the Synchronizer (the runtime format of the SyncBO) is generated

MI furnishes a multitude of tools to design, generate, run, and monitor Smart Synchronization. The developer, therefore, must have a good understanding of the MI toolkit, including the SyncBO Builder, Synchronization Emulator, Profile Dialog, Monitoring Tool, Log Tool, Purge Tool, and Migration Tool.

3.2.2 Tools

MI provides toolkits for system administrators and developers of mobile applications for Smart Synchronization. It is necessary to understand these tools when implementing mobile applications. There is an administration toolkit and a development toolkit.

3.2.2.1 Administration Toolkit

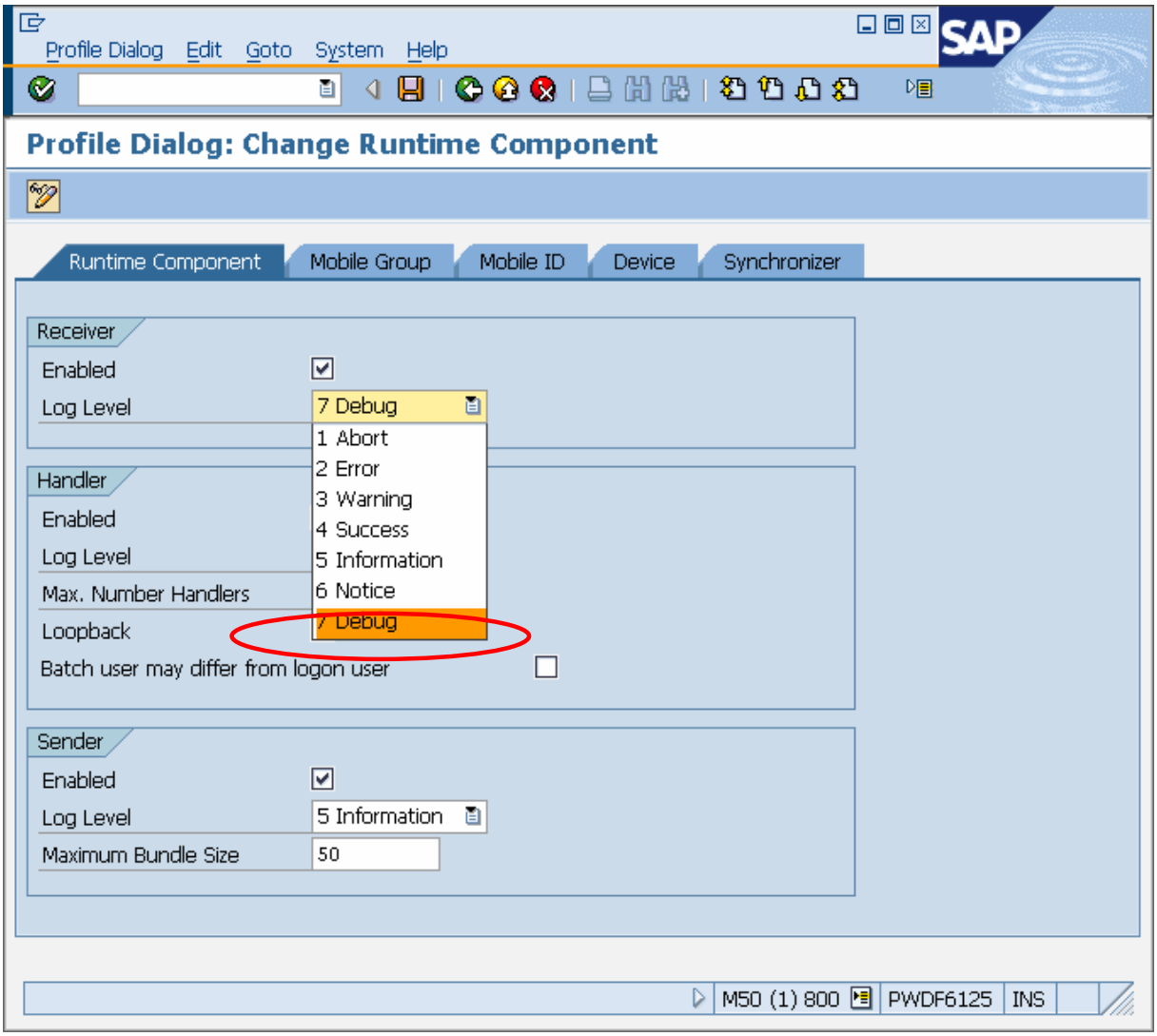
The administration toolkit allows a system administrator to configure what data should be synchronized. You can also control the runtime behavior of the replication processes, monitor sent messages, analyze any problems that occurred and clean up data. Descriptions and tips for using the tools follow below.

Profile Dialog (MEREP_PD)

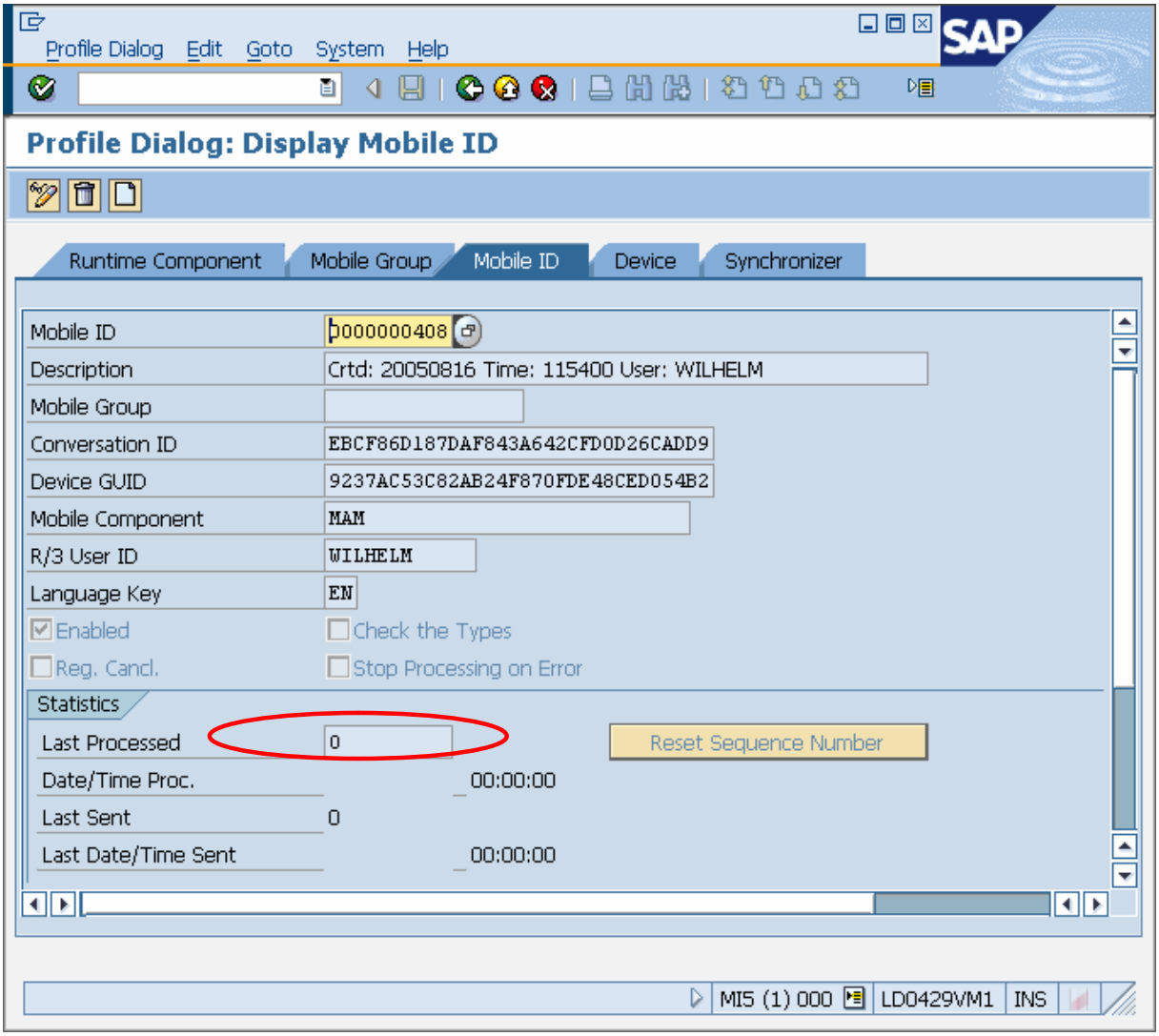
The profile dialog is used to maintain the following Smart Synchronization configuration data:

- Receiver control record
- Handler control record
- Sender control record
- Synchronizer control record
- Device control record (mobile ID)

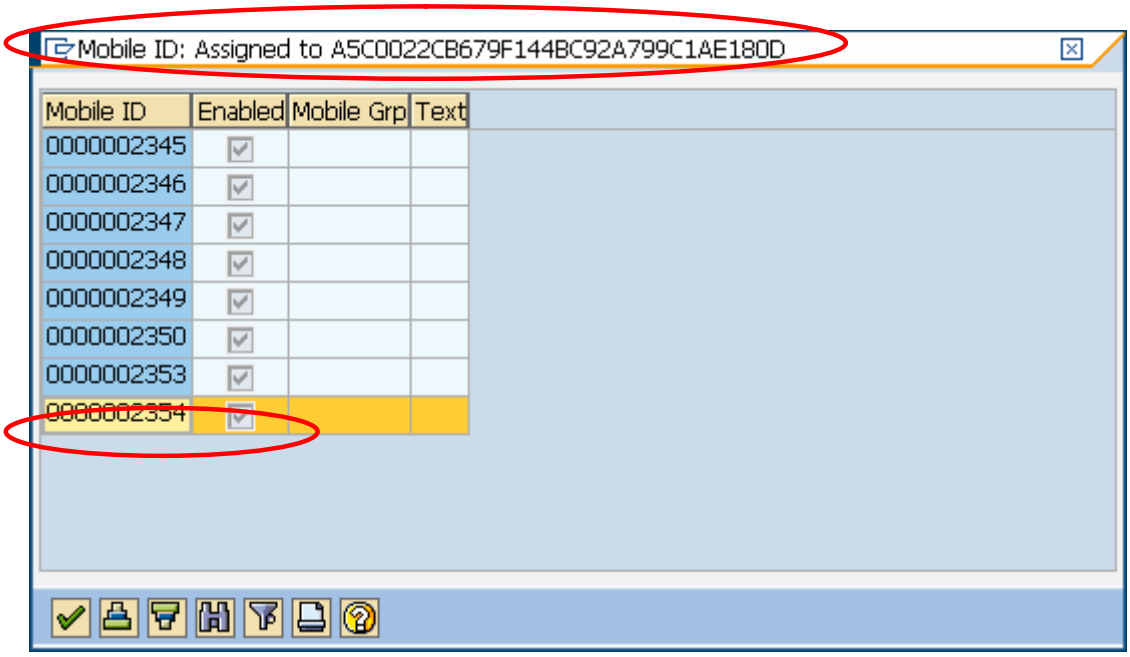
Hint 1: Switch the log level to “7 Debug” during development/debugging to record more information for error recovery. Lower the log level to reduce the size of log file when the system is stable to save disk space.



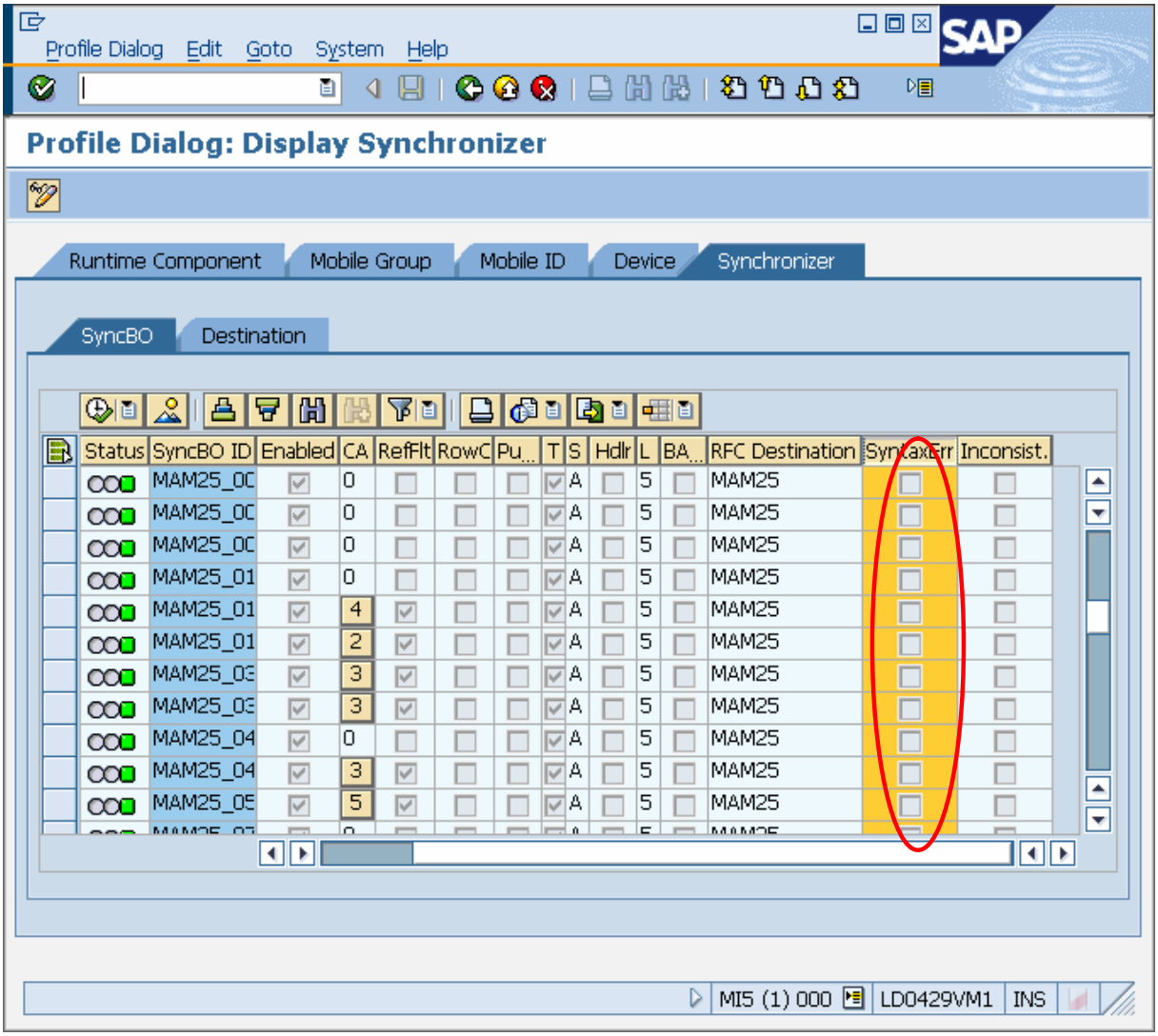
Hint 2: Adjust the “Last Processed” number if synchronization stopped due to a missing sequence number for inbound messages. When you see the messages are in I-Waiting or I-Partial status in the monitoring tool, if the message stopped due to a missing sequence number or you change the status of the message to “Ignored”, the “Last Processed” number should be adjusted to continue the synchronization process.



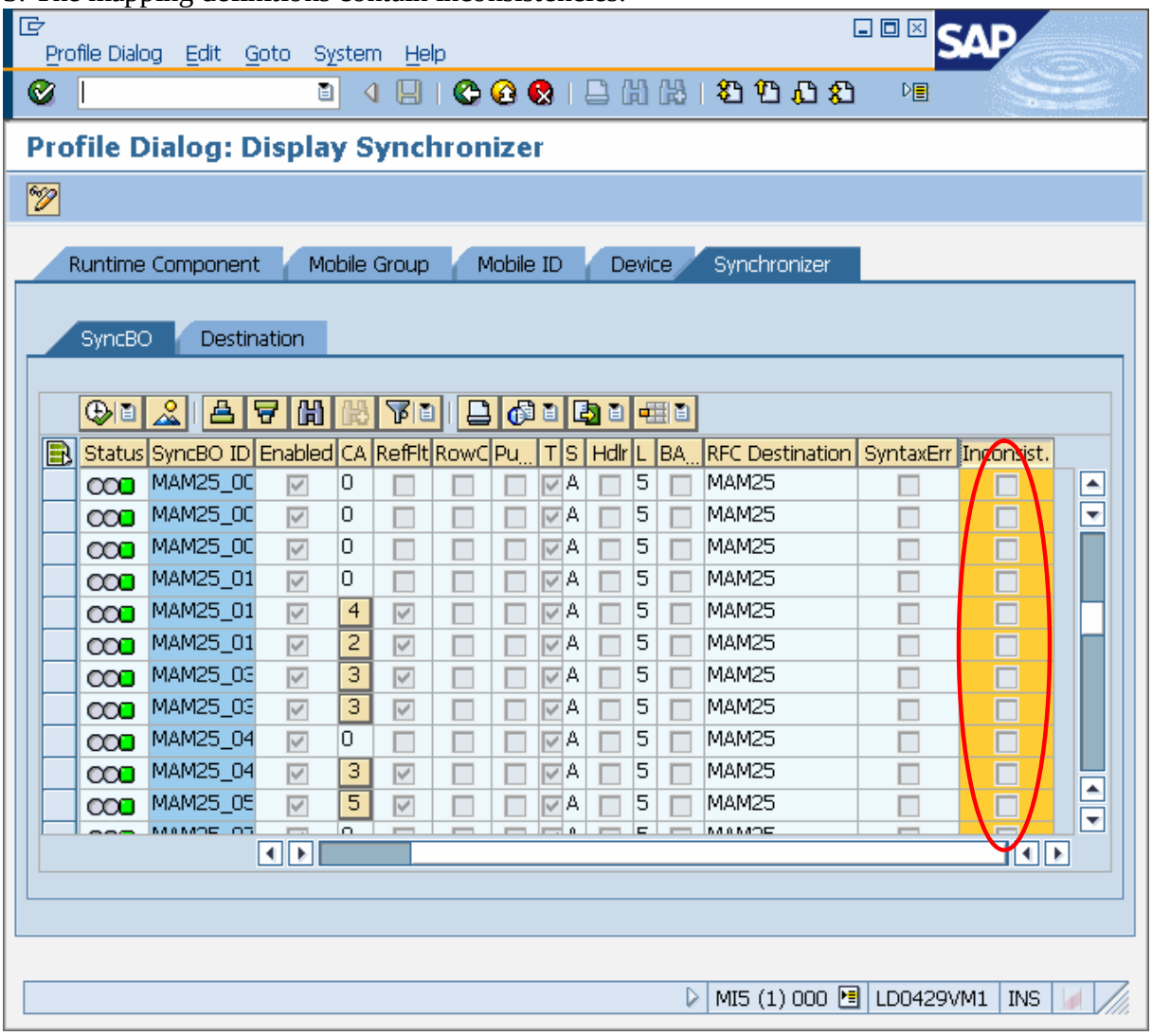
Hint 3: The latest Mobile ID assigned to the device is always the biggest number in the list. The Mobile ID can be used to filter the messages in Monitoring Tool.



Hint 4: If the “SyntaxErr” box is selected it means that there is a syntax error in the generated synchronizer of the SyncBO (usually in the exit of the SyncBO).



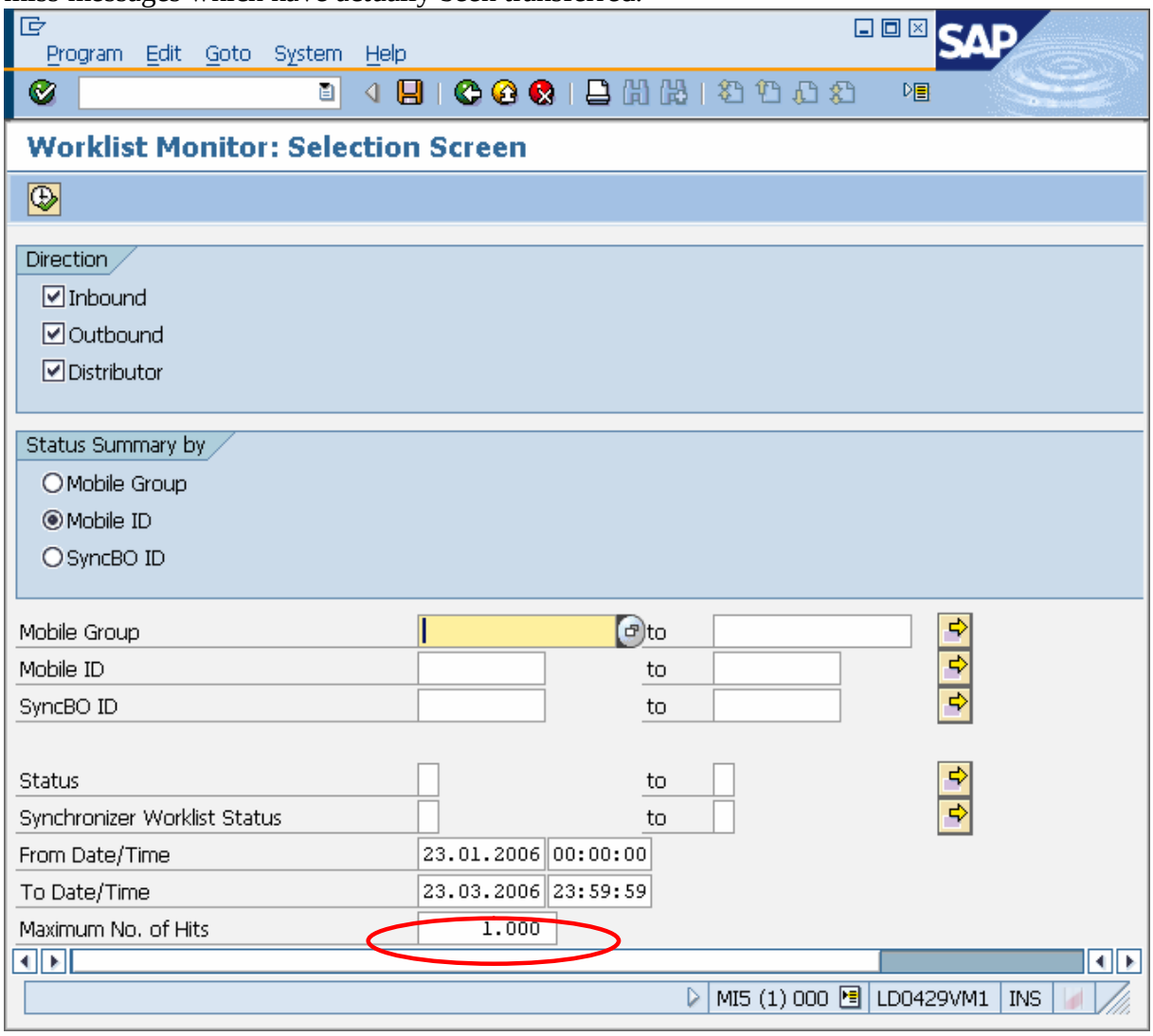
Hint 5: If the “Inconsist” checkbox is selected, it means that there are metadata definition errors in the corresponding SyncBO definition. To solve this problem, run the “BAPI Wrapper Interface Check” function from the SyncBO Builder for the corresponding SyncBO.
This problem could be caused by (for example):
1. The BAPI wrapper interfaces do not match the interface you retrieved when you created the SyncBO or when you modified the SyncBO the last time.
2. The BAPI wrapper interfaces do not satisfy the prerequisites for BAPI wrappers.
3. The mapping definitions contain inconsistencies.



Monitoring Tool (MEREP_MON)

You can display all the synchronization messages in various views. These include, for example, a statistical overview and a handler worklist view.
Using the inbound and outbound messages, you can display logs of the processes and determine what business data was uploaded and downloaded.

Hint 1: Change the “Maximum No. of Hits” to a larger number to check the entire message log from and to the devices. The default value “Maximum No. of Hits” is 1000. This value is usually too small. If the value is not large enough, the administrator can miss messages which have actually been transferred.



Log Tool (MEREP_LOG)

You can display all the editing logs for Smart Synchronization using the log tool. The logs are displayed by process.
You can display all the log data in chronological order. In contrast, the monitoring tool allows you to display the logs according to the synchronization messages.

Purge Tool (MEREP_PURGE)

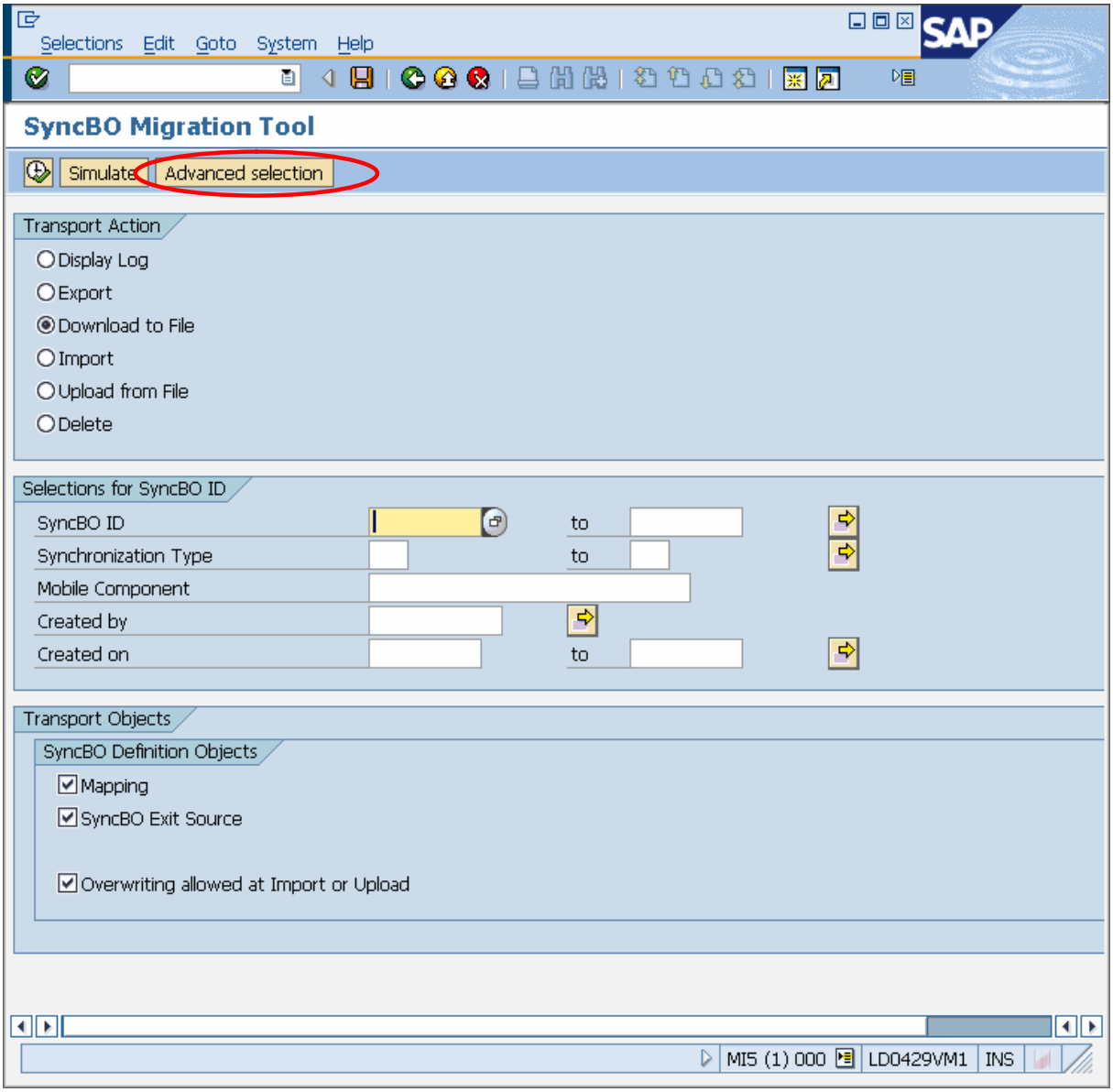
You can delete obsolete, historical runtime data about Smart Synchronization from the system. Since a large amount of data can be transmitted between the SAP MI Server Component and the SAP MI Client Component, it is essential that you, as system administrator, control the overall data volume in the system.
As administrator, you can execute the purge tool manually or automatically using periodical batch jobs.

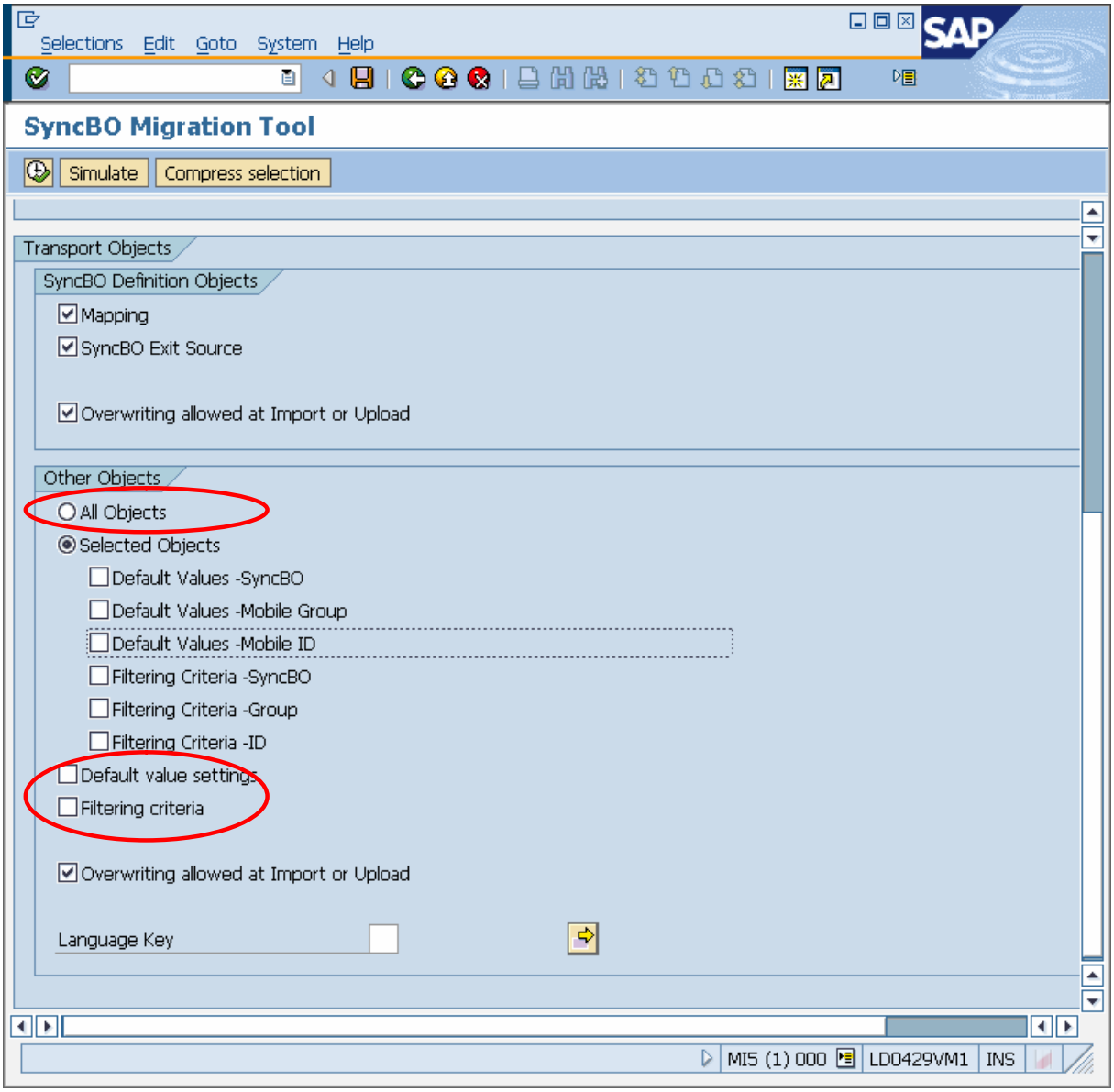
Migration Tool (MEREP_MIG)

You can transport the SyncBO definitions from one SAP NetWeaver Application Server to another. For example, you can transport from the test system to the production system when you have completed your tests.
The following options exist:

- Export of SyncBOs (to a flat file or a transport file)
- Import of SyncBOs (from a flat file or from a transport file)

Hint 1: When exporting the SyncBO definition file, the “Other Objects” section is not displayed by default. You usually have to make some settings in this section (see below). If you do not make these settings, an incomplete SyncBO definition file is imported into the target system and this causes synchronization errors,



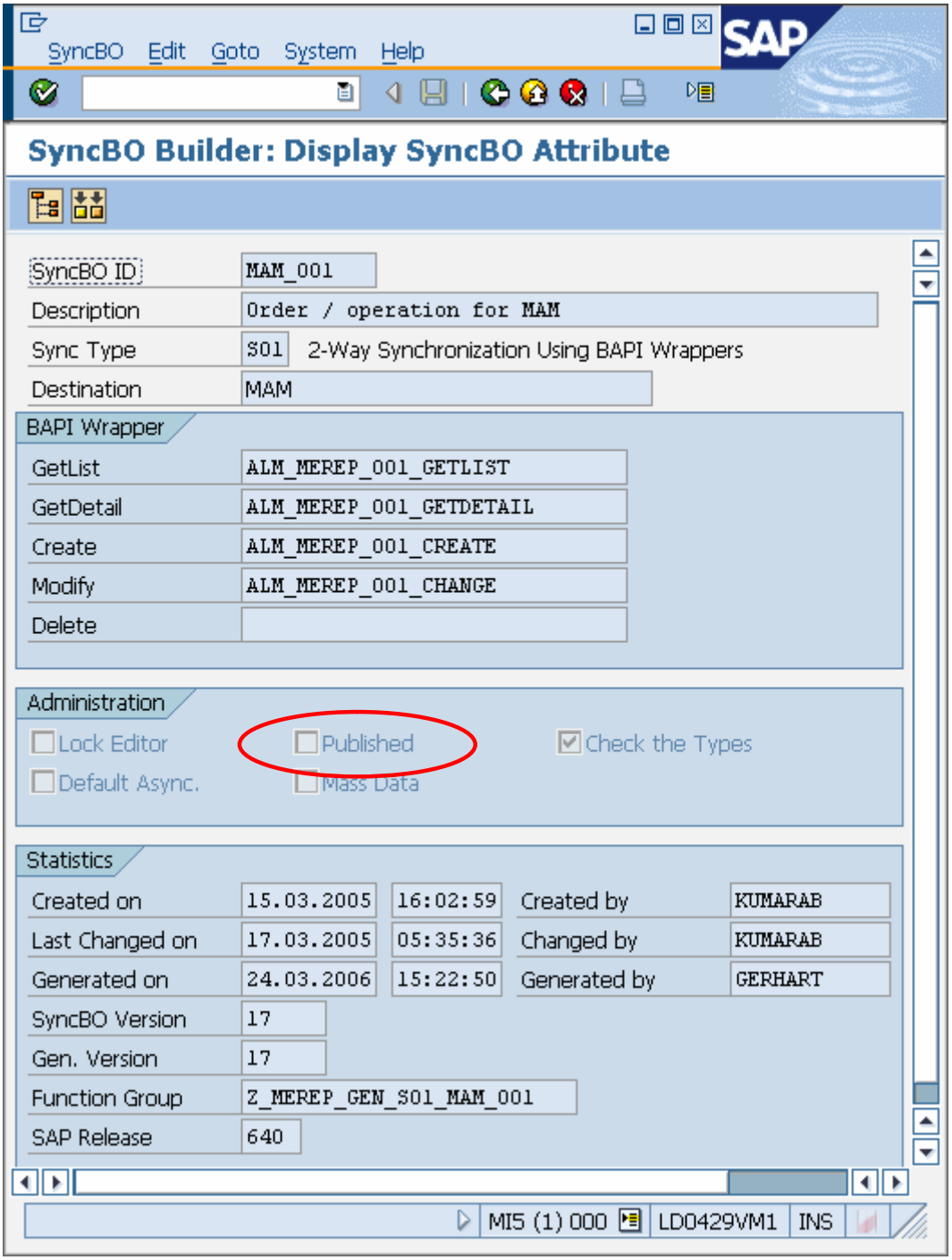


3.2.2.2 Development Toolkit

The development toolkit provides tools to enable you to develop the data model for mobile applications based on Smart Synchronization. Each application-specific data model that is defined and generated with this toolkit consists of one or more synchronizer business objects (SyncBO). The SyncBOs are the foundation of all synchronization processes initiated in Smart Synchronization.

SyncBO Builder (MEREP_SBUILDER)

Hint 1: Do not select the “Published” box during development or when testing. Once this box has been selected you cannot make any further adjustments to the SyncBO.



Synchronization Emulator

The Synchronization Emulator can be called from the “SyncBO” menu of the SyncBO Builder or by entering the program name “MEREP_EMULATOR” in transaction SE38.

3.2.3 User Exits

Custom logic can be added in a user exit to modify the default behavior of Smart Synchronization. The implementation of the user exit is specific to the SyncBO and written in ABAP as it is included in the generated synchronizer code.

SyncBO specific custom logic can be added, for example, to handle the following cases:

3.2.3.1 Dynamic default value assignment

The default values are usually entered in the SyncBO Builder with constant values or a few system parameters. By implementing a user exit, it is possible to fill the default values dynamically at runtime.

3.2.3.2 Dynamic filtering criteria

As with the “dynamic default value assignment,” the filtering criteria values are usually filled with constant values or a few system parameters in the SyncBO Builder. By implementing a user exit, it is possible to fill the values of filtering criteria dynamically at runtime.

3.2.3.3 To call GetAllDetails BAPI Wrapper for all header records instead of calling GetDetail BAPI Wrapper for each header record

You can find more detailed descriptions in the chapter “Performance Tips” in “Back-End System Development”.

3.2.3.4 To avoid returning replace messages for upload messages for performance reasons

You can find more detailed descriptions in the chapter “Performance Tips” in “Middleware Development”.

3.2.3.5 To avoid conflict check during upload for performance reasons

You can find more detailed descriptions in the chapter “Performance Tips” in “Middleware Development”.

3.2.3.6 To call back-end system function modules, for example, to set the replication status for synchronization management handled by the application

You may want an application to control the synchronization process, for example, to record the status of data replication from the back-end system to the middleware on the back-end system for error recovery. This can be done by implementing a user exit to call the back-end system function modules after data replication is finished on the middleware.

3.2.3.7 To check if there is error in the current synchronization

You may want your application to report errors that occurred during synchronization to the end users or administrator. This can be done by implementing a user exit (see example).

3.2.3.8 User Exit Example

An example of a dynamic filtering is to filter the downloaded business objects, where the value of a date field is in the range between the date of the synchronization and one month later.

The following example shows how to determine whether to download each entry in the replication DB to a client device: Add an exit at <X:400000> of the downloader of a download, 2-way or timed 2-way SyncBO.

```
*<X:400000> Synchronization block: Start position
* ldt_rdb_h = Filtered    (based on filtering criteria) HEADER data from Replica
DB
* ldt_rdb_p = ITEM    data under the HEADER data
* The following LOOP will remove the data that should not be downloaded to a
client device

LOOP AT ldt_rdt_h INTO lds_rdb_h .
* Accessible    parameters:
* ldt_rdb_h, ldt_rdb_p
* Import parameter or Tables parameter that will be used as an input to the
downloader
* Add a logic here to remove an entry from ldt_rdb_h and/or ldt_rdb_p unless the
data should be downloaded to the device

END LOOP .
*</>
```

Additional work required when implementing user exits:
After upgrading to a higher version of MI or applying a new MI Support Package on the middleware, it may be necessary to implement the user exits of SyncBOs. This can be checked with the help of a MI consultant.

3.2.4 Performance Tips

3.2.4.1 Avoid returning replace messages for upload messages

If, after upload, the updated data is no longer required on the device (for example, downloaded data is filtered by today's date, and there will always be new data downloaded tomorrow) the replace messages for the upload messages can be ignored. This reduces synchronization time on the middleware server and on the device.
To avoid returning replace messages for upload messages, the user exit of SyncBO has to be implemented.
The following example is sample code for a two-way SyncBO.
SyncBO type: S01
Function module type: S02

Exit key	Sample Codes
399	lds_customizing-write_outbox = gcf_off .

3.2.4.2 Call GetAllDetails BAPI Wrapper for all header records instead of calling GetDetail BAPI Wrapper for each header record

You can find more detailed descriptions in the chapter "Performance Tips" in "Back-End System Development".

3.2.4.3 Avoid conflict check during upload

Since the last synchronization took place and the last data download was completed, the data in the back-end system tables may have been updated. To resolve this issue, MI calls the GetDetail BAPI Wrapper during the upload process before calling the respective

Create/Modify/Delete BAPI Wrapper and compares the return values with the data in the replication DB. If a difference between the two sets of data is apparent, MI generates error messages in the Monitoring Tool.
If the customer's scenario allows changes made in the back-end system tables to be ignored, the GetDetail BAPI Wrapper does not have to be called during the upload process. This reduces the synchronization time by implementing the user exit of the SyncBO.
The following example contains the sample codes for a two-way SyncBO.
SyncBO type: S01
Function module type: S01

Exit Key	Sample Codes

399	`IF NOT i_inbox_data[] IS INITIAL.`
	`  APPEND lds_syncwl  TO e_upload_syncwl.`
	`  READ TABLE ldt_h_from_mbl INTO lds_h_from_mbl INDEX 1.`
	`  PERFORM dsd_coci_build_upload_data`
	`  USING  lds_syncwl`
	`  lds_h_from_mbl`
	`  ldt_p_from_mbl`
	`  CHANGING e_upload_data[] .`
	`  PERFORM dsd_coci_end_of_downloader`
	`  USING e_conflict_r3data[]`
	`  e_conflict_rtn[]`
	`  e_upload_data[]`
	`  ldc_badi_parms`
	`  CHANGING ldc_sync_generic`
	`  e_download_data[] .`
	`  EXIT.`
	`ENDIF.  "IF NOT i_inbox_data[] IS INITIAL.`

3.3 Client

3.3.1 Required Skills

The client development for mobile applications based on Mobile Infrastructure is Java based. The developer must, therefore, be familiar with Java.

The MI client offers multiple APIs, making development easier. The developer must have a strong understanding of these APIs to be able to successfully implement them. These APIs are explained in detail in the Mobile Development Kit (MDK), which can be downloaded from Service Market Place or from SAP Developer Network (SDN).
The following technologies are used to implement a JSP-based mobile application and the developer should be familiar with these technologies as well:

- HTML
- JavaScript
- Java
- JSP/Servlet API
- Servlet containers
- AWT (for AWT based applications)

3.3.2 Tools

3.3.2.1 Development Environments (IDEs)

There is no strict rule for which tools you should use for developing mobile applications based on SAP Mobile Infrastructure. Any Java IDE, such as IDEA, JBuilder or NetBeans, could be used.

However, there are some tools provided by SAP that work in these two development environments:

1. Eclipse & MDK
   Eclipse is a free open-source Java IDE that was developed mainly by IBM. It includes a lot of features and offers a very good Plug-In architecture. There are Plug-Ins for many different APIs and technologies, including the MDK. The MDK is a Plug-In from SAP that allows you to easily develop mobile applications based on SAP MI. You can use it to import your SyncBO settings (metarep.xml) and create a small, working mobile application that allows you to browse your data and that can be used as a starting point for your project.

2. Netweaver Development Studio

   The Netweaver Development Studio (NWDS) is the SAP development environment for all Java-based SAP products. It is based on Eclipse and offers perspectives for Web DynPro, XI, iViews and much more. One of these perspectives is for the MDK.

If you want to develop other Netweaver applications as well, the NWDS is the right choice for you.
However, if you want to develop mobile applications only, the better choice is Eclipse. The NWDS includes several Plug-Ins and requires a lot more system resources than Eclipse. Finally, for PCs with less then 2 GB of RAM, NWDS is not suitable at all.

3.3.2.2 Useful Tools

Ant

Ant is a very well known and powerful build tool that allows you to build your application. It is perfectly integrated into Eclipse and can be used to create your war file or run your unit tests.

FingBugs

FindBugs (http://findbugs.sf.net/) is a tool that can analyze your Java source code and tell you where errors occur. It searches for uninitialized variables and statements that are always true or always false, for example. FindBugs, which was developed by the University of Maryland and is open source, comes with a small UI, but can also be run through Ant.

3.3.3 Architecture for an Easily Maintainable Mobile Application

As with every well designed application, it is important that your application is created in accordance with the Model-View-Controller (MVC) paradigm.

To explain the MVC concept would be beyond the scope of this document. However, to summarize, the concept describes how to separate the business logic (model) from the user interface (view) and the application flow (controller).

If you follow this concept, it is easy to migrate your application from JSP to AWT, from Generic Synchronization to SmartSync, or from laptop to PDA.

The SAP standard MI applications, such as MAM or MTT, are created with a framework known as CAF, which is also based on MVC. This framework also focuses on extensibility (changing the application without modifying any of the files of the standard application). This is important for customers to ensure that they do not lose support if they make enhancements as opposed to modifications.

Enhancements involve overheads: Configuration files, based on XML, which need to be read and parsed. This causes problems for PDAs and should be avoided in your own applications.
You can find more information about the MVC concept at
http://en.wikipedia.org/wiki/Model-view-controller

3.3.4 Deprecated MI APIs

The following is a list of MI APIs that should not be used anymore. These are all APIs that will not be maintained in the future and that you should avoid using in your application:

- Com.sap.ip.me.api.persist.query.Query → use Com.sap.ip.me.api.persist.query.JQuery instead
- Com.sap.ip.me.api.persist.core.Transaction → use com.sap.ip.me.api.persist.core.PersistenceManager instead
- Com.sap.ip.me.api.persist.core.TransactionManager → use com.sap.ip.me.api.persist.core.PersistenceManager instead
- Com.sap.ip.me.api.persist.app.PackageEntityFactory → use Com.sap.ip.me.api.persist.app.EntityFactory instead
- Com.sap.ip.me.api.persist.app.Entity → use Com.sap.ip.me.api.persist.app.PersistableEntity instead

3.3.5 MI APIs in Detail

3.3.5.1 Logging API

The logging API is simple to use, and its integration into the synchronization mechanism and the middleware makes it a really powerful tool.
It is important to consider the ideal number of entries to be written to the log file. If you do not write enough entries, you may not get the required information you need to solve the problem concerned. However, if you write too many, this has a negative effect on performance. The ideal balance depends on the hardware, your application, the amount of data on your device, and on other parameters.

Here are some tips to consider in your client code:

Use caution when tracing in frequently used coding.
Tracing can be very useful to analyze the source of a problem. However, it also uses resources. Tracing statements to the necessary level. Do not use trace statements in program loops.

Concatenate strings only when tracing is activated.
If the description of the trace is more than one string, concatenate the strings only when tracing is activated. String concatenation consumes more resources than an if-statement.

Example:

Wrong:

```
Trace trace = Trace.getInstance(“MyComponent”);
trace.log(Severities.DEBUG, “Some “ + “information”);
```

“Some” and “information” will always be concatenated, even if the trace is off or not logging at DEBUG level.

Right:

```
Trace trace = Trace.getInstance(“MyComponent”);
if (trace.isLogging(Severities.DEBUG))
trace.log(Severities.DEBUG, “Some“ + “information”);
```

Performance Trace

The performance trace can be used to trace the performance of your application. This can be very helpful to find out which parts of your code require the longest time to execute. If you have this information, you can then focus on optimizing the code.

To activate the performance trace, you have to add the following line to your MobileEngine.config:

```
MobileEngine.Trace.PerformanceLog.Enabled=true
```

Below is an example of how to trace the performance in your code:

```
Public String some Method() {
    // Performance Log entry at the beginning of the method:
    Object pTag = PerformanceLog.methodStarted(this);

    try {
      this.doSomeCoding();
      String bla = SomeHelper.doSomething();
      // .... some more coding
    }   catch ( SomeException e ) {
      handleSomeException(e);
    } finally {
```

```
      // Performance Log entry at the end of the method:
      PerformanceLog.methodFinished(pTag, "method someMethod");

    }
  }
```

The performance trace file can be found in the MI_HOME/log folder. Use the same rules for the performance log as for regular logging.

3.3.5.2 Generic Sync API

Synchronization of large amounts of data

If you download a large amount of data to your MI client, especially if you run on a PDA, this may result in an OutOfMemoryError.
To avoid this, you could implement your Generic Sync in such a way that the server does not return all data at one time, but firstly splits the data into packages, and then downloads them separately.

- This only applies if you synchronize in synchronous mode.
- As you have to call SynchronizeWithBackend several times, an installed SmartSync application on the same device would be triggered to synchronize multiple times.

Use `readNextElement()` on InboundContainer instead of `getAllElements()`, `getElementsWithFieldName(String fieldName)` and `getElement(String fieldName, String lineNumber)`

The methods getAllElements(), getElementsWithFieldName(String fieldName) and getElement(String fieldName, String lineNumber) need the entire container in the memory.

The method readNextElement() does not consume so much memory. Note the following points:

1. Do not mix the method readNextElement() with the methods getAllElements(), getElementsWithFieldName(String fieldName) and getElement(String fieldName, String lineNumber).
2. The method readNextElement() has to be called in InboundProcessor.process(InboundContainer). Do not store references to the InboundContainer and process them outside the method InboundProcessor.process(InboundContainer).

3.3.5.3 SmartSync API

The MI SmartSync APIs have been changed considerably over recent years. Each new API version and function has represented an improvement on its predecessor, and as such, you should always use the latest APIs in your application. Here are some hints about what to use:.

Use `SyncBoReplyObserver` instead of registering to `SyncBoInDeltaObserver.`

When using `SyncBoInDeltaObserver` instances, every inbound delta has to be instantiated as an `Object` that holds all fields as value objects. Therefore, the data cannot be passed on to the persistence layer in the faster stream-based mode.
A more efficient solution to get conflict messages from the back-end system-processing of a SyncBoDelta is to setup a `SyncBoReplyObserver` for `SyncReplyType.ERROR / SyncReplyType.CONFLICT / SyncReplyType.TECHNICAL`.

Do not use untyped read.

An untyped read forces SmartSync to look in all the relevant persistence tables for an entity with the specified key.
Example for an untyped read:

```
SyncBo.getRow(String);

SyncBoDataFacade.getSyncBo(String);
SyncBoDataFacade.getRow(String);
```

Performance improvement using RowDescriptor:

```
SyncBo.getRow(RowDescriptor,BigInteger);
SyncBoDataFacade.getSyncBo(SyncBoDescriptor, BigInteger);
SyncBoDataFacade.getRow(RowDescriptor, BigInteger);
SyncBo.getRow(RowDescriptor,String);
SyncBoDataFacade.getSyncBo(SyncBoDescriptor, String);
SyncBoDataFacade.getRow(RowDescriptor, String);
```

Use values > 18 digits only when absolutely necessary.

Complex numeric values require more resources. The decision about the precision of the numeric values has to be made at the BAPI wrapper design time.

Use `Row.modifyFieldValue(Object)` instead of `Row.setFieldValue()`.

Every time the Row.setFieldValue() method is called, a full DB-update with transaction commit is performed.
To overcome this problem, an unlinked row has been introduced that needs fewer resources, although the row is cloned and various other precautions have to be taken.

Row.modifyFieldValue(Object) modifies a single field without triggering a DB-operation.
SyncBo.modifyRow(Row) finally performs the DB update for all changes.

Use type specific methods to get field values.

The Row.getFieldValue(FieldDescriptor) / Row.getFieldValues() methods instantiate and return value objects, according to the “old” mapping model, such as N>String, P>FixedDecimal and D,T>Calendar.
The persistence stores values according to the “new” mapping model (that is N>BigInteger, P>BigDecimal, D>Date, T>Time) and, therefore, a field value conversion is required.

Use the methods Row.getCharacterField(), Row.getDateField(), Row.getDecimalField(), Row.getNumericField(), Row.getTimeField() to get the field value.

Use `SyncBoDataFacade.getRows(Query)` / `SyncBoDataFacade.getSyncBos(Query)` before using `SyncBoDataFacade.size(Query)`

The `SyncBoDataFacade.size(Query)` method scans the entire database table. This operation does not need as many resources as a query operation because no result set is created.
If, however, a query operation has to be performed anyway, you can save resources by invoking the query first and then calling the `size()` method.

Avoid Query condition where possible

Queries are resource intensive. Check if different cases can really occur, based to the data definition in the back-end system. Try to convert user input strings to uppercase or lowercase and make one individual query.
Example: Query to find a part number
The user input can be in upper or lower case, or both. The application uses a query with AND to cover all possibilities.
Solution:

1. The part number is a numeric value (characters between 0 and 9).
2. The character case is defined by the back-end system. Values will generally be handled in uppercase only.

Request only entries you want to display

Use the SmartSync API to get the entries you want to display (defined by starting index and number of entries). We recommend the following:

1. Use JQuery API to select the entries required for display. Cache the keys to read the corresponding row on demand (for example, if you want to edit it).
2. Use the result iterator to scroll forward. If you must store the list yourself, do not store the entire list.
3. Execute a separate query using the COUNT (number of entries) operator.

Applications must handle synchronization at any time.

Synchronization is a framework-driven process. Synchronization is started when a network connection is available, the user chooses the synchronization button, and all transactions on the client are closed. The application must be able to handle synchronization at any time.

3.3.5.4 AWT UI

Reuse Screen Elements

As PDAs have limited memory, we have to be careful about which and how many objects we create. If you develop an application that has an AWT UI, you should reuse screen

elements (such as buttons and labels) wherever possible. Even if they are on different screens, use a pattern that allows you to reuse them.

3.3.5.5 JSP UI

Performance: Visuals?

If you develop MI JSP applications for a laptop, these applications can resemble your usual Web applications. However, if you are developing for a PDA, you are more restricted in what you can do.

One example of this is the use of graphics. Elaborate graphics slow down performance, mainly because it takes longer to render the HTML/graphic with the Internet Explorer.

If you have no alternative but to insert graphical elements, they should be referenced in the file system and not by means of the MI Web server.
Example:

If you reference the image as shown in the first example, it is read from the file system by the MI Web server and then sent to the Web browser, which then renders the image.
In the second example, the image is read from the file system directly by the Web browser. This is much faster, as it does not load the MI Web server.

Another problematic area for PDA applications are tables. They offer a common way to structure HTML pages and to create layouts. On a PDA, however, the rendering of tables is particularly slow (especially if a table is nested inside another table). We recommend, therefore, avoiding using tables where possible.

To summarize:

- Keep your pages simple
- Avoid graphics
- Avoid nested tables

The “Double Click Issue”

The “double click issue” refers to a problem that occurs when the user clicks on a link twice on the PDA or when he or she clicks two or more links on the UI. This will bring your application into an undefined state.

To avoid this problem, you must make sure it is impossible for the user to do this. You do this using JavaScript:

```
<a href=”#”
onClick=”goTo('/MAM/Controller?id=42');”>link</a>
```

Instead of going directly to a page, you execute a JavaScript function, such as the following:

```
var linkActivatedFlag = false;

function goTo(url) {
  if (linkActivatedFlag == false) {
    document.location.href = url
    linkActivatedFlag = true;
  }
}
```

The same has to be done for forms. If the user clicks on the submit button, the form needs to be submitted and the button needs to be deactivated:

```
<form name="myMiApplicationForm">
<input type="button" name="myMiButton" value="Click to Submit"
onClick="doSubmit();">
<script language="JavaScript">
function doSubmit()
{
  document. myMiApplicationForm.myButton.disable = true;
  document. myMiApplicationForm.submit();
}
</script>

</form>
```

Adapt this pattern for all the JSPs/HTML pages of your MI application.
The disadvantages of this approach are:

- JavaScript is browser-dependent. Your JavaScript implementation may not work with all browser versions.
- This approach does not cover illegal navigation, such as returning to an outdated page by pressing the browser’s return button.

3.3.6 Best Practices

There are many ways to optimize the performance of your Java application. However, in many cases the effort that needs to be made to improve, for example, the speed of your Web or desktop application by a few milliseconds seems to outweigh the benefits gained.

As PDAs do not have the same resources as desktop machines, when you start developing mobile applications for PDAs, these small changes can really improve your application. . Here are some examples:

Use StringBuffer instead of string concatenation.
The concatenation of more than 2 Strings with the + operator in frequently called methods consumes a lot of resources. This is a general Java issue. When concatenating strings with +, a new string object is instantiated, together with memory-allocation for the underlying char array, for every single concatenation.

Example: Following command instantiates nine objects and allocates char-arrays.

```
String s = “hello ” + “world ” + “here ” + “I “ + “am “;
```

Example: Ineffective way to a add trailing spaces.

```
private String addLeadingSpaces(String st, int len)
{
  while (st.length() < len)
  {
    st = "0"+ st;
  }
  return st;
}
```

Pass final objects into methods when possible
If you are sure that a parameter of a method will only be read and not modified, then you should declare it as final. If defined as such, your code is executed faster.

Optimize your for-loops
Avoid creating a loop with a limit that is a method. For example, for(i=0; i< obj.bigMethod(); i++). In this case, the bigMethod is executed with each loop.

Example
Incorrect:

```
for (int i=0; i<someMethod(); i++) { …}
```

Correct:

```
s = someMethod();
for(int i=0,i<s; i++) {…}
```

In the first example, the method someMethod() is called for each loop. In the second example, it is just called once.

Remove your System.out.println() calls
Many developers continue to include System.out calls in the code.during development. When the application is rolled out to users, these calls should be removed.
The System.out calls can lead to performance bottlenecks and result in a slow application. If the application is running on a PDA, the output that is written to System.out is written into a file called jscpout.txt in the root of the PDA file system. As file system access is slow on PDAs, this should be avoided at all costs.

Use Arrays instead of Collections
Whenever possible, you should use Arrays instead of Collections. The former are much faster.

Free resources quickly

If you open file handles or URL connections inside your application (for whatever reason), make sure that the memory allocated by them is released once you have used them. The best way to do this is in the finally-block to make sure that even when a exception is thrown, the resources will be freed.

3.3.7 Testing

Testing is not generally closely integrated with development. This prevents developers and project managers from measuring the progress of development. It is very difficult to determine when something starts working or when something stops working. Incrementally writing unit tests helps to measure progress, highlights unintended side effects, and increases the quality of the application. There are several commercial and non-commercial testing frameworks available to choose from, for example, junit,.

Rules:

- Unit tests must be created before the implementation has started, or at the same time for each iteration.
- Create test methods for at least each public method of your MI application-classes. Relevant private/protected methods should be covered by unit tests too.
- Execute the unit tests on the target platforms. If you are using a PDA, the test cases must be executed on the PDA platform.
- Integrate the unit tests into your build process.
- Test performance regularly on the target device during development. If performance problems are recognized during development, they can be fixed. If problems are not recognized until the end of the development cycle, it is much harder to rectify them.

4 Packaging/Development Phase

4.1 Required Skills

You should have a strong understanding of the MI deployment mechanism.

Experience of basic Java class-loading and of the creation of jar and war files would also be helpful.

4.2 Tools

To create war files that are used to deploy MI applications, you can use your development environment (NWDS or Eclipse) with the MDK view or PlugIn. You can also use Ant.

To deploy your installation packages, you have to use the server-side MI tools (the Web Console for pre-SP15 landscapes or the NetWeaver Administrator (NWA) (SP15 and higher)). A detailed description of these tools can be found in the MI Installation and Administration guides.

4.3 Precompiling JSPs

Under normal circumstances, the JSP pages of your MI client application are compiled when a JSP is entered for the first time. This task takes a long time and, as such, can have a negative effect on the overall application performance, in particular with PDAs. It is strongly recommended to compile the JSPs before the application is exported as a Web archive. If you are using the MDK to create the Web archive, select the box “Add JSP as compiled classes” and the JSPs are added as compiled classes (see screenshot). If you are using a central build script, you can employ jasper (for example in an ANT script).

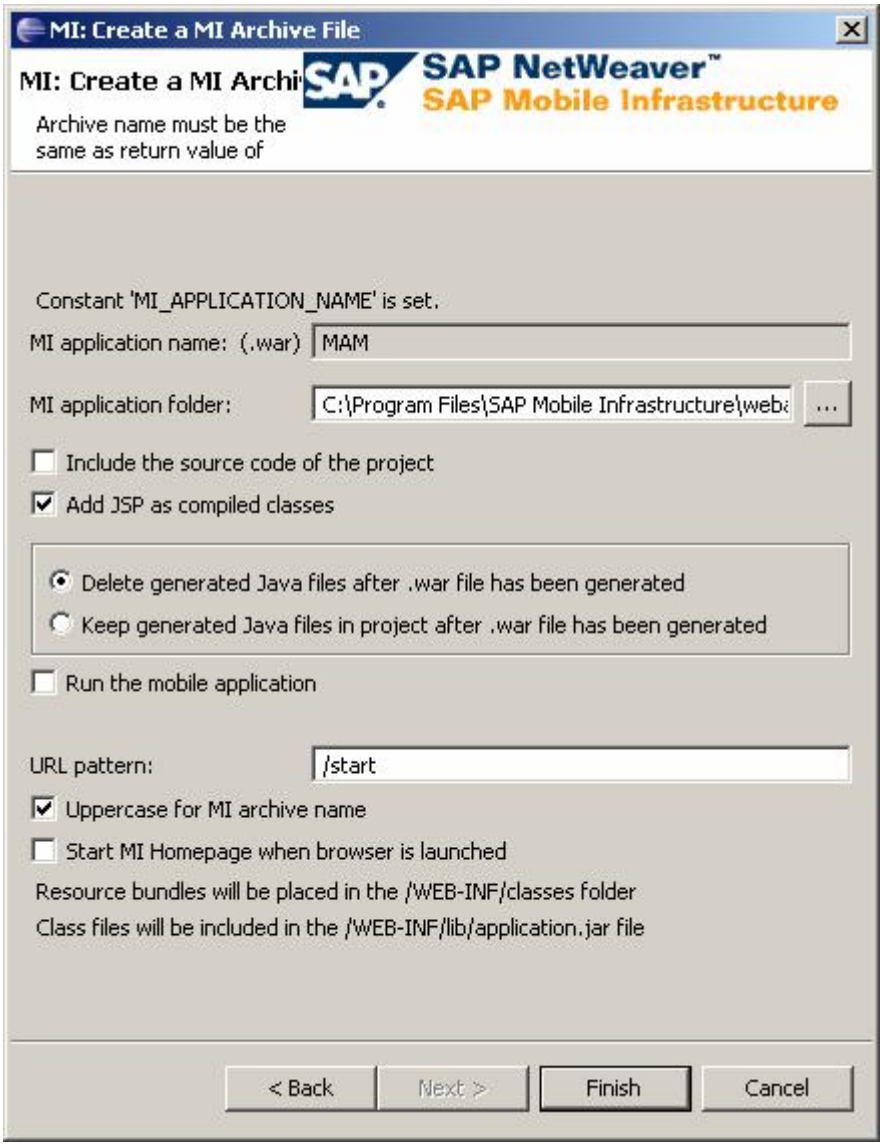


Figure 1: Precompiling JSPs in the MDK

4.4 Creating Installation Packages

MI applications are distributed as standard Web archives (WAR) files. The easiest way to create the war for an MI application is to use the export function of the MDK plugin (for more information, see the MDK documentation).

Nevertheless, we recommend integrating an automated central build into the software development process that builds the client application on a daily basis (at least). This central build can be provided by, for example, NWDI.

Note: The MDK export function works as follows: All the compiled Java classes for the application are put into a Java archive. This archive is added to the folder WEB-INF\lib of the Web archive. When the application is started for the first time, the archive has to be extracted by the MI client framework. Depending on the complexity and size of the MI

client application, this can be a very time-consuming task. Performance can be improved by placing all the compiled Java classes into the folder WEB-INF\classes instead.

4.4.1 Update, Patches and AddOns

When an MI client application is changed, we must consider the best way to update productive MI devices. Depending on the type of fixes that are contained in the update, either a full redeployment of the application or just a replacement of some files may be required.
In general, we can distinguish between two kinds of updates:

- Updates that include changes to the SyncBO model
- Updates that do not include changes in the SyncBO model on the condition that the application itself does not externalize any Java class by object serialization, and that the application itself does not register any application class as a SyncEvent-Listener. (SyncEvent-Listeners are externalized automatically by the MI framework).

4.4.1.1 Updates with SyncBO Model Changes

As soon as a new version of an application implements a changed SyncBO model, a full update of the MI client application has to take place. The current version has to be removed from the mobile device (delete the application assignment using the Web Console) before the new application version is assigned to the mobile device and deployed by means of synchronization. Export your application using the MDK, upload the created Web archive to the Web Console and execute the required deployment steps.

4.4.1.2 Updates Without SyncBO Model Changes

Whenever an application has to be deployed again because a new version is available, a new initial data download is required. This can be avoided if the application version does not contain any SyncBO model changes and if there is no externalization/serialization of some of your application classes.
For such an update, you have to export your new application Web archive as above.. Take the war file and create either an addon or a client installer archive.

- Client Installer Archive:
- Create a temporary folder <<TEMP>>
- Create a subfolder named <<TEMP>>/<<APPNAME>>
- Create a subfolder named <<TEMP>>/<<APPNAME>>/<<webarchive>>
- Copy your war named <<app.war>> to <<TEMP>>/<<APPNAME>>/<<webarchive>>.
- Create a file named install.xml in <<TEMP>>
- Copy the following instructions to the install.xml file:

```
<?xml version="1.0" encoding="UTF-8"?>
<ARCHIVE  name="<<APPNAME>> update version x" description="update"
uimode="full">
<PACKAGE  name="<<APPNAME>>" description="Installation and Migration Package">
<TASK name="Delete old application file">
<!-- Delete all files of deployed app in webapps-->
```

```
<DELETE targetfolder="%MI_HOME%/webapps/<<APPNAME>>" pattern="**"
mi_running="false"/>
</TASK>
<TASK name="Delete compiled jsps">
<!-- Delete all files of deployed app in work-->
<DELETE targetfolder="%MI_HOME%/work/localhost_8080%2F<<APPNAME>>" pattern="**"
mi_running="false"/>
</TASK>
<TASK name="Deploy new web archive">
<!-- Copy the new <<APPNAME>> war to webapps -->
<COPY source="webarchive" pattern="**" target="%MI_HOME%/webapps"
mi_running="false"/>
</TASK>
</PACKAGE>
</ARCHIVE>
```

After that open a shell or a DOS window in <<TEMP>> and type **`jar cvf <<APPNAME>>.zip .`** Upload the created zip archive as an addon using the Web Console and assign it to all the devices that are to be updated.

http://www.sdn.sap.com/irj/sdn/howtoguides