Designing and developing mobile applications for SAPConsole using ABAP

Best practices and lessons learned

by W. Patrick Tunney

W. Patrick Tunney
SAP Business Analyst,
WPT Consulting Inc.

W. Patrick Tunney began his SAP career working for SAP at its Karlsruhe, Germany facility in 2000. Since then, he has become a recognized expert in the areas of supply chain logistics and warehouse management. As founder and president of WPT Consulting, Patrick helps clients get the most out of their investments in their SAP systems through the application of best practices that ensure maximum ROI. Patrick currently resides in Toms River, New Jersey. You may reach him at patrick.tunney@wptconsulting.com.

Now that the use of wireless barcode scanning equipment is common worldwide, companies are challenged to adapt existing warehouse management software to fit on the smaller screens of handheld devices. Furthermore, these mobile applications need to be basic enough that users on the shop floor — often with little or no technical knowledge — can work with them. SAPConsole, available with SAP R/3 Release 4.6B and higher, allows you to access data on mobile devices. It renders the SAPGUI interface on smaller screens using buttons and plain text.

However, companies often find that the standard set of small-screen applications provided with SAPConsole is insufficient for their needs. As a result, these companies develop custom small-screen programs for wireless data entry to meet business goals and drive process efficiency. Although custom development is generally discouraged due to maintenance concerns, a successful custom SAPConsole application can improve your warehouse processes by allowing users to capture back-end data in real-time easily and accurately. This extends the reach of back-end data to the shop floor, which not only leads companies to achieve greater supply chain visibility, but also provides floor personnel with an explicit, repeatable set of steps to run SAPConsole transactions from start to finish.

Designing small-screen SAPConsole applications for non-technical shop floor personnel presents a unique set of challenges. You need a broad range of design skills that include a strong understanding of business processes and insight into application usability engineering. I'll show you some best practices for creating custom mobile SAPConsole applications for non-technical users. I'll explain my philosophy for designing mobile data entry applications by using an example SAPConsole application that involves a material inquiry transaction. I'll then go over some of the constraints you may encounter when designing mobile data entry screens.

Finally, I'll provide some design recommendations and show you how to apply the design patterns. In addition to describing a general mobile data entry design philosophy, I'll show you some specific coding principles and lessons I've learned in my seven years of SAP mobile application development. By the end of the article, you should have a broad overview of the design challenges associated with mobile development as well as a general understanding of many of the techniques used in these designs. For more information on custom SAPConsole development, see the sidebar on the next page. Let's begin by looking at what a simple SAPConsole application does and how it is structured.

Note!

Although SAP Mobile Infrastructure (SAP NetWeaver Mobile in SAP NetWeaver 7.0) is the latest mobile technology offering from SAP, for SAP teams more comfortable with ABAP, SAPConsole, in my opinion, is the most viable way to develop online mobile applications.[1]

[1] For more on SAP NetWeaver Mobile, see the article "Best practices for planning, deploying, and maintaining mobile applications with SAP NetWeaver 7.0" (*SAP Professional Journal*, September/October 2007).

SAPConsole application example

Figure 1 shows an example of a simple material inquiry transaction, standard SAPConsole transaction LM12. The left screenshot is the first screen that appears. The middle screenshot shows data that the user entered, either through manual entry or via barcode scanning. Clicking on the F7 button in this screen validates the data. The system then displays source bin inquiry information in the right screenshot, which is the second screen the user views.

From a SAPGUI environment in a production system, the SAPConsole engine renders the screens into simple text elements at runtime and displays them on an industrial-grade radio frequency (RF) mobile computer (**Figure 2**). Depending on the RF hardware used, you can access the buttons through dedicated function keys, touch screen interaction, or both.

Mobile data entry design philosophy

Given the constraints outlined in the previous section, a mobile data entry application must be as simple as

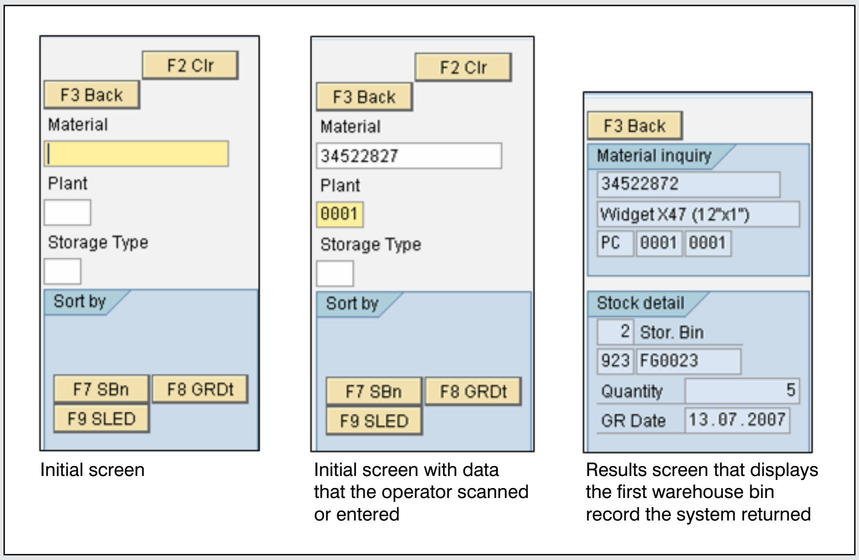


Figure 1 Sample SAPConsole application

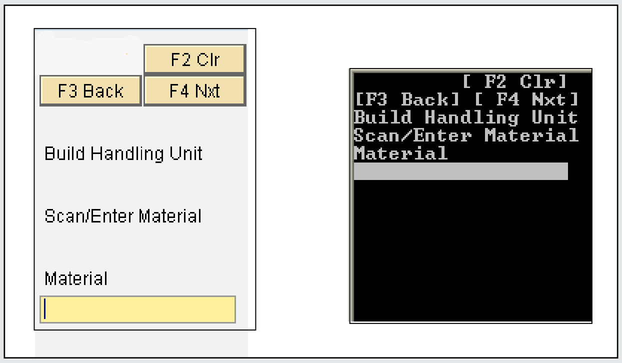


Figure 2 A custom SAPConsole application screen shown from both the SAPGUI and from an RF telnet client

SAPConsole and general custom development guidelines

SAPConsole transactions are often limited in scope, but they are necessary to drive business process efficiency. However, it's important to keep in mind the basic fundamentals inherent to the development of any custom code in the SAP landscape. For example, it is a well-understood principle that SAP installations should always favor customization over custom development. SAP offers such a wide range of business functionality that you can frequently meet business requirements by following a simple three-step process: 1) carefully analyze business process requirements; 2) match the business requirements with the functionality SAP offers; and 3) work through configuration activities to change the way the standard SAP transactions collect and record data functions. Generally you must perform these steps in multiple iterations with refinements to the business process based upon new information gathered regarding available SAP functionality.

Note!

The decision to develop custom SAPConsole transactions is based on the ability of the existing transactions to support shop floor business processes. For new SAP installations, make sure that SAPConsole requirements roll-in follows the business process customizing phase. If customizing isn't locked in, you'll spend a lot of time and money redesigning and recoding your SAPConsole transactions.

SAP implementation teams regularly benefit from using this three-step process. Most importantly, the SAP functionality analysis helps businesses identify ways in which they can improve their existing business processes. SAP has crafted business processes to meet the needs of a wide variety of industries and constantly improves them. As such, the standard SAP processes reflect successful best practices that companies use globally. If you're using SAP's systems but routinely ignoring SAP Best Practices in favor of homegrown ones, you're costing your business money in both lost productivity and unnecessary development costs.

Continues on next page

Continued from previous page

Keep this in mind any time you set out to design a new SAPConsole transaction. The table below contains a list of SAPConsole transactions delivered in the standard system. (Note that this is a partial list of the SAP transactions included in the SAP table TSTC — transaction SE16.) Usually you'll find that the return on investment (ROI) of a custom SAPConsole program tailored to your specific business needs far outweighs the risks and associated costs of customer development. Nevertheless, sometimes you'll find that the standard version is just what you need.

Note!

Sometimes you'll be tempted to modify existing SAPConsole transactions to add business-specific functionality. This is advisable only if the modification is minor. The existing SAPConsole applications support multiple screen sizes and customizing configurations, but modifying these transactions correctly can be difficult and time consuming. If you do decide to create a modification, I advise you to copy the program into your own custom namespace before making changes.

Code	Activity	Code	Activity
LM00	Logon Radio Frequency	LM27	Put Away by Delivery – w/o Selection Screen
LM01	Dynamic Menu	LM30	Load Control – Load by Shipment
LM02	Select by Storage Unit – Put Away	LM31	Load Control – Load by Delivery
LM03	Put Away – by Transfer Order	LM32	Load Control – System Guide Load
LM04	Put Away – System Guided	LM33	Load Control – Unload by Shipment
LM05	Picking by Transfer Order ID	LM34	Load Control – Unload by Delivery
LM06	Picking – by Delivery ID	LM35	Load Control – Detail by Shipping Unit
LM07	Picking – System Guided	LM36	Load Control – Detail by Delivery
LM09	Put Away by Delivery ID	LM37	Load Control – Detail by Shipment
LM11	Posting Change	LM45	Pick and Pack
LM12	Material Inquiry	LM46	Pick and Pack by Delivery
LM13	Put Away Clustered	LM47	Nested Handling Units
LM18	Handling Unit Inquiry	LM50	Count Inventory by System Guided
LM19	Handling Unit – Pack	LM51	Count Inventory by User Selection
LM22	Handling Unit – Unpack	LM55	Print Storage Bin Labels
LM24	Packing Handling Unit by Delivery	LM56	Select by Storage Unit – Interleaving
LM25	Unpack Handling Unit by Delivery	LM57	System Guided Putaway – Interleaving
LM26	Picking by Delivery – w/o Selection Screen	LM58	System Guided Dynamic Inventory Count

Continues on next page

Continued from previous page

Code	Activity	Code	Activity
LM59	User Initiated Dynamic Inventory Count	LM71	Goods Receipt by Delivery
LM60	User Guided Dynamic Inventory Count	LM72	Goods Receipt by Staging area
LM61	Goods Issue by Delivery	LM73	Goods Receipt by Shipment
LM62	Goods Issue by Staging Area	LM74	Goods Receipt by ALL Criteria
LM63	Goods Issue by Shipment	LM76	Goods Receipt by Handling Unit
LM64	Goods Issue by ALL Criteria	LM77	Queue Assignment
LM65	Goods Issue by Group	LM80	Serial number capture
LM66	Goods Issue by Handling Unit		

possible and designed for a specific task. Operators should have to enter a minimal amount of data and be able to do so either through barcode scanning or by selecting a single function key. The transaction should guide the operator down an easy-to-follow path of data entry steps that prompt the operator for data at each stage. Once completed, the operator should be able to wash, rinse, and repeat over and over without any surprises. Any hiccups along the way should be expressed as simple error messages that are easily understandable and that help operators resolve the underlying issue.

Whereas each of the aspects that make up an ideal transaction seem straightforward enough, combining them in a single application that meets a specific business goal is easier said than done. Often you need to make some tough decisions, such as adding functionality at the expense of simplicity. Although each business scenario is different and presents unique challenges, I'll share some guidelines that you can use for reference. I have compiled the following four high-level design recommendations to help you:

- The three screen rule
- Directed data entry
- Immediate input validation
- Post by item, not by document

You'll find that valuable mobile data entry transactions, including the standard ones that SAP provides, combine two or more of these elements.

The three screen rule

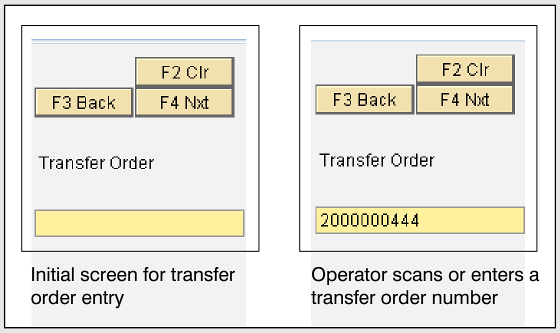
A best practice for mobile applications is to minimize screen flow — the logic that controls the user experience by determining which data output/entry screen to display based on operator input. In turn, the easiest way to minimize screen flow is to minimize the total number of screens. In general, the best applications are ones that get the job done in three screens or less:

- Screen 1 — Enter SAP document
- Screen 2 — Review/select document items
- Screen 3 — Enter or confirm data

An example of a three screen application is a transfer order picking transaction. This transaction might include an initial screen in which the operator scans or enters a transfer order, a second screen that displays individual transfer order items for inspection and selection, and a final transfer order confirmation screen in which an operator confirms a selected transfer order item.

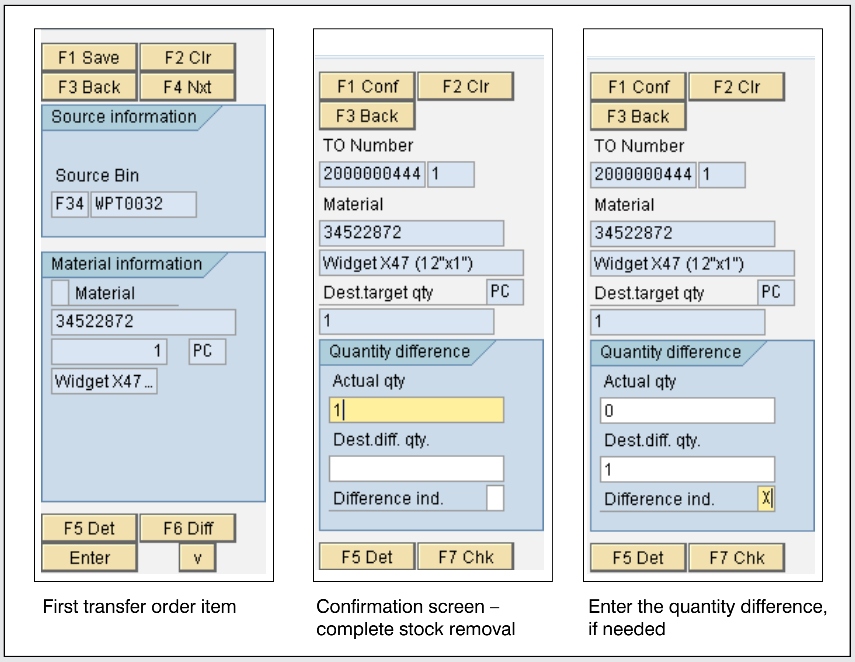
Let's walk through an example of entering an SAP document number, reviewing document items for selection, and entering the confirmation data. **Figure 3** shows the first of three screens from transaction LM05 (Picking by Transfer Order ID) — before and after data entry. The left screenshot shows the initial transfer order data entry screen. The screenshot on the right shows what the screen looks like after the operator enters the transfer order document number.

Clicking on the F4 button or scanning the transfer order displays the second screen shown in **Figure 4**. The screenshot on the left shows the first transfer order item that the operator must pick. The operator reviews the source information to find the storage bin on the warehouse floor, and then clicks on the F6 button to confirm the order. If the total required


F2 Clr
F3 Back
F4 Nxt
Transfer Order
Initial screen for transfer order entry
F2 Clr
F3 Back
F4 Nxt
Transfer Order
Operator scans or enters a transfer order number

Transfer Order: 2000000444

Figure 3 Entering transfer order data



First transfer order item: Source Bin F34 WPT0032; Material 34522872; 1 PC; Widget X47 ...

Confirmation screen: TO Number 2000000444 1; Material 34522872; Widget X47 (12"x1"); Dest.target qty 1 PC; Actual qty 1; Dest.diff. qty. (blank); Difference ind. (blank)

Quantity difference screen: TO Number 2000000444 1; Material 34522872; Widget X47 (12"x1"); Dest.target qty 1 PC; Actual qty 0; Dest.diff. qty. 1; Difference ind. X

Figure 4 Process for posting an item

quantity is found in the bin, the operator removes the material from the bin and enters the total quantity in the middle screenshot. If the quantity available in the bin differs from the required pick quantity, the operator enters the difference and selects the difference indicator in the right screenshot. In both cases, the operator follows up data entry by clicking on the F1 button to post the item.

Transaction LM05 demonstrates an uncomplicated, repeatable shop floor process that operators can use to pick materials from a warehouse storage bin. Each screen has a single purpose and data entry steps that drive the screen navigation independent of document data. Shop floor operators without any SAP knowledge can quickly step in and start working to ensure that data entry and shop floor processes are in sync.

Depending on the level of business process automation and application requirements, you can create some applications in fewer than three screens. A material/bin inquiry transaction might consist of a single screen to scan a bin with a follow-up screen to display bin stock totals. An optimized transfer order putaway application might automatically select transfer order items for putaway based on transfer order queuing priority. This could reduce the application to a single transfer order item confirmation screen.

If you find yourself in a situation in which your design contains more than three screens, see if you can break the transaction down. If you can't, it may be worth considering whether the business process in question is a good candidate for mobile data entry or whether it is better suited to the SAPGUI environment.

Directed data entry

When creating a mobile data entry application, ensure that you have a well-defined path from the initial screen to the final screen, regardless of the number of screens in the application. With the limited amount of input and output available to the operator, there isn't a whole lot of room for flexibility.

Consider the SAP MIGO transaction — the mother of all Inventory Management (IM) goods movement transactions. This transaction provides functionality that allows a user to perform most IM goods movements against most IM documents. As result, the interface can be daunting even for a seasoned SAPGUI specialist. Translating that functionality to a handheld application is practically impossible, especially considering that the default screen display is only 16x20 characters.

What's more, the realization of the entire transaction would require at least 10 screens with complex screen flow logic to manage the application state and screen transitioning. The end result would be an application with a significantly higher learning curve (especially for operators with little SAP system experience) containing built-in logic complexity that would be difficult to test thoroughly and maintain.

A more reasonable alternative to this approach is to select a MIGO sub-task, such as goods receipt for purchase order, as shown in **Figure 5** (on the next page). The handheld transaction could consist of a screen flow that drives the operator directly from document entry, to the purchase order item review, and then to the individual item goods receipt. This approach makes the final application more palatable to your end users and minimizes software development and maintenance overhead.

Also, in a mobile data entry application, avoid creating a search function. Directed data entry relies on guiding an operator down a specific path. Searching indicates that the application entry point is unknown. It can introduce transaction ambiguity and make it difficult to design a concise user experience.

From a practical perspective, searching means that you need to add two screens, one for entering search criteria and one for selecting from a list of results, in addition to whatever screens you need to accomplish your business task. Chances are that adding these two screens will cause you to violate the three screen rule. Obviously, at times searching is justified, but as a general rule you should avoid including a search option.

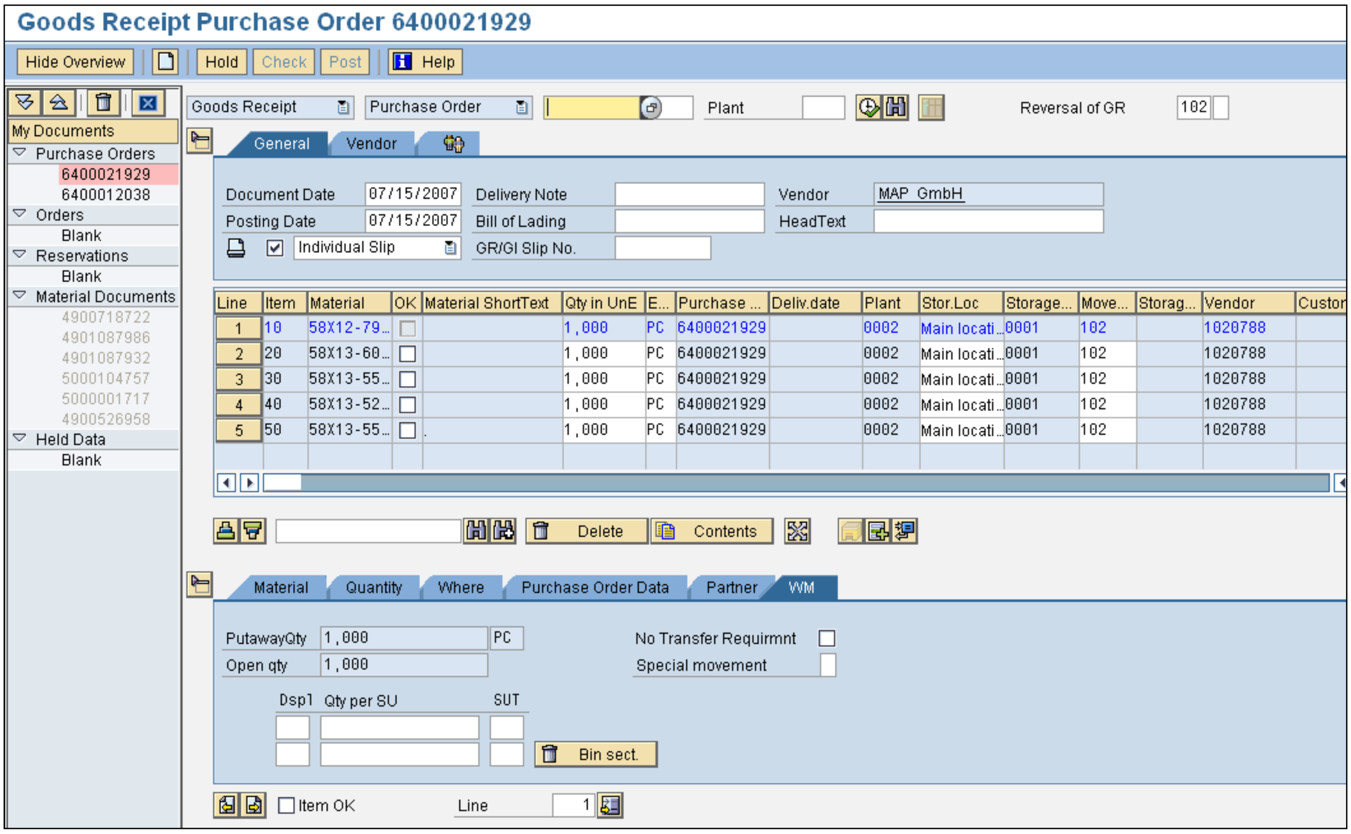
Immediate input validation

The directed data entry approach relies on guiding a user down a path based on data entry. As you guide your operators down that path, it is important to verify the validity of data as soon as the operator enters it. This differs from the way many SAP ABAP applications operate. Although SAP provides programming mechanisms that validate individual fields upon entry in response to a Process After Input (PAI) event, most PAI events capture multiple field data changes. As a result, the system validates data en masse when you transition from one screen to the next. In a mobile data entry environment, the system should validate data as it receives it, capturing each field entry as a PAI event.

Figure 6 shows a series of handling unit (HU) screens that collect a source bin, source HU, and a destination bin. The system validates each element upon entry to guide the operator down a predefined data entry path.

To illustrate this point, consider a warehouse bin-to-bin movement transaction that creates and confirms a transfer order to move a pallet of material from a source bin to a destination bin. This transaction requires the operator to scan a source bin and an HU number (pallet license plate). The operator then confirms the movement by scanning the destination bin, which triggers the transfer order posting.

In this example, the mobile data entry version of the application must account for space concerns that SAPGUI does not consider. An operator physically removes the pallet from a source bin and moves it to a destination bin. To save time, when the operator scans the source bin and HU number, the system should detect any impediments to creating the transfer order. Such validations include sanity checks, such as


Goods Receipt Purchase Order 6400021929

Goods Receipt	Purchase Order
Plant	
Reversal of GR	102

My Documents
- Purchase Orders
  - 6400021929
  - 6400012038
- Orders
  - Blank
- Reservations
  - Blank
- Material Documents
  - 4900718722
  - 4901087986
  - 4901087932
  - 5000104757
  - 5000001717
  - 4900526958
- Held Data
  - Blank

Document Date	07/15/2007	Delivery Note		Vendor	MAP GmbH
Posting Date	07/15/2007	Bill of Lading		HeadText	
✓	Individual Slip	GR/GI Slip No.			

Line	Item	Material	OK	Material ShortText	Qty in UnE	E...	Purchase ...	Deliv.date	Plant	Stor.Loc	Storage...	Move...	Storag...	Vendor	Custor
1	10	58X12-79...	☐		1,000	PC	6400021929		0002	Main locati...	0001	102		1020788	
2	20	58X13-60...	☐		1,000	PC	6400021929		0002	Main locati...	0001	102		1020788	
3	30	58X13-55...	☐		1,000	PC	6400021929		0002	Main locati...	0001	102		1020788	
4	40	58X13-52...	☐		1,000	PC	6400021929		0002	Main locati...	0001	102		1020788	
5	50	58X13-55...	☐	.	1,000	PC	6400021929		0002	Main locati...	0001	102		1020788	

PutawayQty	1,000	PC	No Transfer Requirmnt	☐
Open qty	1,000		Special movement	☐

Dspl	Qty per SU	SUT

☐ Item OK   Line 1

Figure 5 MIGO transaction in which the operator entered a purchase order for goods receipt reversal (movement type 102)

verifying the existence of the HU number or ensuring that the source bin is not blocked for inventory.

The decision to check these fields individually or together doesn't matter in a SAPGUI environment, but it can have serious cost implications for a shop floor with mobile devices. If one of the primary purposes of mobile data entry is to drive business process efficiency, then it is important to validate data as soon as possible to ensure that you are not inadvertently introducing inefficiency into your shop floor business process (e.g., the operator physically moves the HU and finds out when he attempts to post that the HU is blocked for movements within the warehouse).

Post by item, not by document

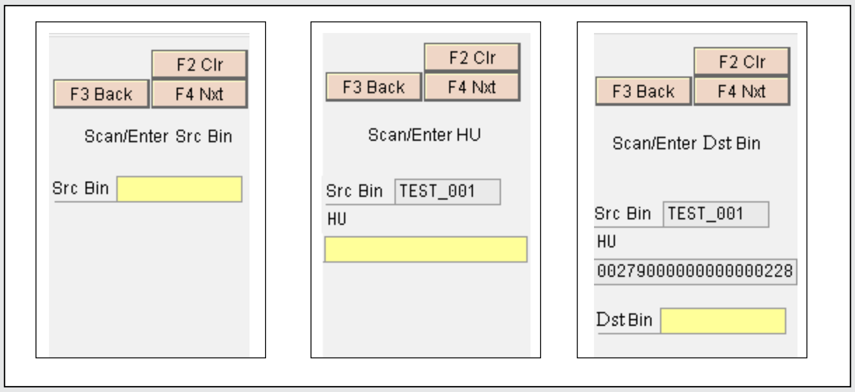
Many SAP interfaces allow you to perform data postings against individual document items. For example, in the goods issue to process order application, a SAPGUI operator enters a process order, enters the issue quantity for one or more process order items, and then executes the posting function. The end result of a successful posting is a single material document with multiple items — one for each process order item issued.

Even though this approach is technically possible in a mobile data entry transaction, I recommend that your transaction post each item as soon as the item data is entered. The reason for this is twofold. First, by enforcing data posting immediately upon data entry, you provide your operators with an interface that clearly identifies the transaction state — you either posted the data successfully or the system returned an error message. If you encounter an error for one item, you can address it immediately. This is important when operators can only review one item at a time, as opposed to a SAPGUI environment in which an operator has an overview of all process order items.

Second, implementing multiple item document posting is more complex in a mobile data entry environment with limited input and output. You spend more time deciding how to respond to partial item data entry. Also, often errors returned from SAP posting interfaces do not clearly indicate the item to which they belong. In these cases, it's difficult to direct your operator to the source of the error.

Mobile data entry design constraints

When developing transactions for mobile devices, such as warehouse barcode scanning equipment or fork-lift mounted mobile computers, you must consider three unique design constraints. The most obvious constraint is output display size. A handheld barcode scanning device has a significantly smaller screen size when compared to a standard 17-inch display. The same is true for forklift-mounted computers. The limited



Src Bin: TEST_001

HU: 00279000000000000228

Figure 6 Data entry screens for moving a handling unit within the warehouse with source bin verification

screen size means that you only have room for the most important information required to complete a transaction. What's more, the character length of a data field is often wider than the width of the device screen. Not only do you have to choose which data to display, you also need to consider how to display it.

Another important consideration is the limited ability to perform data input. Most mobile devices have either a small, built-in keyboard or a touch-screen interface. The smaller the device, the more painful and error prone character-based data entry becomes. You can't reasonably expect a user to engage in any significant free-text entry; otherwise the user spends more time in front of a desktop SAPGUI, defeating the purpose of the mobile application. Furthermore, some user input mechanisms, such as drop-down menus and value helps, don't function the same on all mobile devices. Then again, other mechanisms, such as check boxes and radio buttons, only work when you use a mouse or touch-sensitive display. Fortunately, barcode scanning equipment helps automate the data entry process when barcode labels are readily available for scanning.

Finally, the third constraint is the users' lack of familiarity with the warehouse management system. Applications must validate data immediately to prevent data-entry errors. In my experience, if an application provides a chance for an operator to introduce bad data inadvertently, operators will find a way to do so.

The ability to recognize and code for mobile device constraints in an application is the defining characteristic between a good ABAP developer and a good SAPConsole developer. In the next section, I'll show you how these constraints affect design decisions.

Development recommendations and applied design patterns

Now that we have examined the mobile data entry problem set and reviewed some of the high-level design concepts, let's take a look at the development phase. In the following sections I'll discuss some of the applied design patterns and coding tips that you can use to keep your code clean and manageable while enforcing some of the high-level concepts introduced in the previous section. These tips include:

- Minimize top include sharing.
- Define screen structures.
- Validate fields and screens.
- Avoid flag variables.
- Use standard function key mappings.
- Use standard function module interfaces exclusively.

Minimize top include sharing

When used correctly, top includes that contain common code and form routines can simplify development and maintenance in complex development projects. They save time by reducing the amount of necessary development and debugging because programs developed in the same package can share resources. However, unless you are developing a significant number of transactions to introduce into the production landscape as a group, you should keep include sharing to a minimum. Think of each mobile data entry transaction as a single self-sufficient unit. Ideally, it's made up of the following:

- One to three data entry screens
- Validation routines for individual screens
- Control logic for screen mapping
- Posting logic for SAP interfaces and table data

That's all there is to it — four simple bullet points. The SAPConsole menu model does an excellent job of handling menu dependencies that are external to the individual transactions, so you don't have any overhead. If you follow the suggestions laid out in the previous section, each of these bullet points, with the exception of possibly posting logic, should be as lean as possible. If that's the case, there isn't much room for code sharing between programs.

What's more, as with most custom programs, SAPConsole transactions evolve over time. By keeping the use of top includes to a minimum, you're writing yourself an insurance policy that ensures that changes to one transaction do not adversely affect others in your production environment. The stability benefits outweigh any reduced development effort obtained from code sharing.

Define screen structures

Recall that your custom mobile data entry transactions are self-sufficient units. You can take that model a step further and apply it to individual screens. An SAPConsole screen may contain the following elements:

- Display-only output fields
- Data entry fields or verification fields
- Function key and button mappings

These are the only elements your transaction should have. No radio buttons. No check boxes. No ActiveX controls. Nada. Your transaction has to survive not only a SAPGUI to text conversion, but also various hardware-specific renderings post-text conversion. (SAPConsole works by converting ABAP programs to text at runtime and sending them to a handheld device via a third-party transport mechanism — usually Telnet.) With that in mind, I recommend grouping your input and output fields together in a single screen-specific structure.

> ***Note!***
>
> If it sounds like I'm belaboring the point of keeping your transactions simple, it's because I am. As hard as it is to believe, one of the biggest challenges I've faced over the years is getting my clients to understand that in the mobile data entry space simpler is not only better — it's paramount. Don't learn this lesson the hard way.

Figure 7 contains a code snippet taken from a single-screen transaction that moves an HU stored in warehouse inventory to an operator-entered destination bin. You can add this code to the program's top include file that declares the variables used throughout the

```
REPORT   ZMDE_MOVE_HU.

TABLES: LAGP, VEKP, LEIN.

DATA: BEGIN OF screen_1000,
          hu            TYPE LENUM,
          src_bin       TYPE LGPLA,
          src_bin_type  TYPE LGTYP,
          hu_type       LIKE LTAP-LETYP,
          dest_bin      TYPE LGPLA,
          dest_bin_type TYPE LGTYP,
       END OF screen_1000.

* User warehouse
DATA LGNUM LIKE LQUA-LGNUM.

* Current cursor position used for screen 1000
DATA cursor_1000(20) TYPE C VALUE 'SCREEN_1000-HU'.
```

Figure 7 Top include file from a single-screen SAPConsole transaction

program. Notice the declaration of the data structure `screen_1000`. I like to group input and output fields together in one, screen-specific package. This helps me keep a handle on all of my data, which is especially helpful when multiple screens contain similar fields.

Validate fields and screens

As discussed in the previous sections, the system should validate data as soon as the operator enters it to maximize efficiency and minimize lost time. I recommend that you validate individual fields as soon as the system receives the data.

Figure 8 shows the PAI module for my single-screen transaction that moves an HU within the warehouse. The system calls this module after input on the data entry screen and handles all possible input events — in this particular application this includes BACK (exiting the transaction), OK (ENTER or barcode scan the event), and SAVE (posting of data back to SAP). As in all standard ABAP applications, these values are populated in the standard system field `sy-ucomm` in response to a user interaction. In this particular module, I used `sy-ucomm` to drive a CASE control flow statement.

Let's take a look at the specific data validations that take place in response to the OK event. The cursor position that is available in the global variable `cursor_1000` drives my data validations. This is important in SAPConsole applications because companies usually configure barcode scanning to append an ENTER event to the data that is scanned into the field with cursor focus.

Note!

All industrial-grade wireless barcode handhelds support the option to append an ENTER event to a barcode scan, triggering data validation. This feature automates your data entry in SAPConsole. If you are buying new equipment that doesn't support this feature, this is a red flag that the equipment is not industrial grade.

```
MODULE USER_COMMAND_1000 INPUT.

  GET CURSOR FIELD cursor_1000.
  CASE sy-ucomm.
    WHEN 'BACK'.

      ...

    WHEN 'OK'.
      CASE cursor_1000.

        WHEN 'SCREEN_1000-SRC_BIN'.
          PERFORM CHECK_BIN USING    screen_1000-src_bin
                            CHANGING screen_1000-src_bin_type.
          PERFORM CHECK_BIN_REMOVAL USING screen_1000-src_bin
                                          screen_1000-src_bin_type.
          cursor_1000 = 'SCREEN_1000-HU'.
```

Continues on next page

Figure 8 PAI field-level validation and cursor control

```
        WHEN 'SCREEN_1000-HU'.
          PERFORM CHECK_HU_NUMBER   CHANGING screen_1000-hu.
          PERFORM CHECK_HU_REMOVAL  CHANGING screen_1000-hu.
          PERFORM GET_HU_DATA USING screen_1000-hu
                              CHANGING screen_1000-src_bin
                                       screen_1000-src_bin_type
                                       screen_1000-hu_type.
          cursor_1000 = 'SCREEN_1000-DEST_BIN'.

        ...

        WHEN 'SCREEN_1000-DEST_BIN'.
          PERFORM CHECK_BIN USING    screen_1000-dest_bin
                            CHANGING screen_1000-dest_bin_type.

          PERFORM CHECK_BIN_PLACEMENT  USING screen_1000-dest_bin
                                             screen_1000-dest_bin_type.
          cursor_1000 = 'SCREEN_1000-DEST_BIN'.
      ENDCASE.

    WHEN 'SAVE'.

        ...
```

Figure 8 (continued)

It's important to notice that each field has a specific set of validations that the program performs on data entry, as indicated by the second CASE statement used for variable `cursor_1000` in **Figure 8**. The validations are specific to the field in which the system captured the data. Also, I set the cursor field destination to SCREEN_1000-DEST_BIN after successful validation. This allows me to maintain control of the data entry process and drive data entry down a single path to completion.

For example, in the code in **Figure 8**, whenever a user enters in the HU number and presses Enter, the system executes the validations under the CASE statement for SCREEN_1000-HU. In this situation, the system then executes the form routines CHECK_HU_NUMBER, CHECK_HU_REMOVAL, and GET_HU_DATA. Assuming each of these routines executes successfully, the system sets the cursor variable `cursor_1000` to SCREEN_1000-DEST_BIN, which returns control to the operator, who then enters or scans the data for the next field.

Note!

I didn't use the standard PAI module approach to control how the operator enters data and to avoid dealing with error messages and field locking that don't render well on mobile devices. The process I outlined reduces testing overhead and ensures that the program executes consistently across Telnet clients and RF devices.

In addition to validating individual fields, you should also consider validating the entire screen after the system collects all the data. This typically happens prior to transitioning to a new screen or before posting. The code in **Figure 9**, a continuation of the code from **Figure 8**, shows the PAI module processing block that the system executes when the operator selects the posting function key.

As you can see, the first form routine called in the SAVE processing block is titled CHECK_ SCREEN_1000. **Figure 10** displays the code for FORM CHECK_SCREEN_1000. In this particular example, you don't need to perform an exhaustive screen data validation because the subsequent BAPI posting picks up any process-specific data issues.

I've found that field- and screen-level validations are good mechanisms for defining my SAPConsole applications. Although in some cases it can lead to duplicate data checks — for example, above when the operator scans the destination bin immediately before posting — I find the stability and clarity it brings to my code is worth the additional overhead.

Avoid flag variables

There is nothing more frustrating than attempting to reverse-engineer code written by someone else that is littered with what appears to be control flow decisions based on ambiguous flag variables. What's more, even with good variable documentation, you're likely to forget what each flag variable represents and how it affects subsequent processing decisions.

Consider the bin-to-bin example discussed in the "Immediate input validation" section. After performing the bin blocking check on picking, a novice programmer might be tempted to set a flag variable that the system references before posting data back to the underlying subsystem through a function module call. Instead, a better approach is to immediately reject invalid source bins by clearing the user-entered input and displaying an error message. As a result, a source bin is valid if it exists in the input field and you don't need any flags to know a bin's validity. By checking the source bin value every time it changes and clearing out invalid values, you no longer need to collect state data using flags.

Look at the form routine code from the form include file in **Figure 11**. This system calls this code from a PAI module in response to an HU entry. This form routine checks if the HU entry exists in the standard table LEIN. If it does, then the system does not invoke any error processing and control flow continues back to the calling PAI module. If it does not, the system clears the entry, displays an information message, and aborts all subsequent processing.

```
    ...

  WHEN 'SAVE'.

    DATA l_tanum LIKE LTAK-TANUM.
    PERFORM CHECK_SCREEN_1000.
    PERFORM POST_HU_MOVEMENT CHANGING l_tanum.
    MESSAGE I012(ZMDE) WITH l_tanum.
    LEAVE TO TRANSACTION sy-tcode.

    ...
```

Figure 9 PAI module snippet executed upon posting request

```
FORM CHECK_SCREEN_1000.

* Basic sanity checks, BAPI will cover any conditions
* not checked here.

  PERFORM CHECK_HU_NUMBER    USING screen_1000-hu.

  PERFORM CHECK_BIN USING    screen_1000-src_bin
                    CHANGING screen_1000-src_bin_type.

  PERFORM CHECK_BIN_REMOVAL USING screen_1000-src_bin
                                  screen_1000-src_bin_type.

  PERFORM CHECK_BIN USING    screen_1000-dest_bin
                    CHANGING screen_1000-dest_bin_type.

  PERFORM CHECK_BIN_PLACEMENT  USING screen_1000-dest_bin
                                     screen_1000-dest_bin_type.

ENDFORM.
```

Figure 10 Form routine that performs screen-specific validation on SCREEN_1000 data

```
FORM CHECK_HU_NUMBER CHANGING P_HU.

  SELECT SINGLE * FROM LEIN
    WHERE LGNUM = LGNUM
      AND LENUM = P_HU.

  IF sy-subrc NE 0.
    CLEAR P_HU.
    MESSAGE I003(ZMDE).
    LEAVE TO SCREEN sy-dynnr.
  ENDIF.

ENDFORM.                     " CHECK_HU_NUMBER
```

Figure 11 Simple form to validate an HU input value

With this technique, the input field passes validation if it is populated. If it is empty, the operator either neglected to enter a value or must correct the entry. This reduces the need to rely on flag variables and allows you to minimize the amount of code needed to get the job done.

Use standard function key mappings

You'd think that this suggestion would go without saying. After all, sticking to SAP standards is a fundamental rule for any application. New SAPConsole developers, however, are often unfamiliar with the standard set of SAPConsole applications, which lay down the de facto guidelines concerning how you should use function keys in small-screen transactions. Furthermore, new SAPConsole developers frequently overlook the significance of function key mappings. After all, in a standard SAP application, typically only power users apply function keys as shortcuts to mouse navigation.

In a mobile data entry transaction, function keys are the fundamental glue that allows operators to navigate screens, enter and clear data, and scroll through list results. Function key mappings are important because they allow for easy data entry and navigation on RF devices. I've seen operators carry out their day-to-day workload by memorizing function key sequences — even though they have little understanding of the underlying SAP business processes. You should use the function keys listed in **Figure 12** for the tasks described.

Note!

The target audience for your SAPConsole applications differs from the usual end user (or consumer) of your custom ABAP applications. Many companies use temporary employees for shop floor activities (particularly if business demand is seasonal), so using easy-to-remember function keys becomes especially important.

As a final note, even if you do not use the F1 and F4 function keys in your application, be sure to override their default behavior. SAPConsole is notorious for short-dumping when an operator inadvertently calls up an F1 help dialog or an F4 search help that it can't handle. Override their default behavior either by handling the ON HELP-REQUEST and ON VALUE-REQUEST events in the PAI and Process Before Output (PBO) definition section of each screen or by defining a custom dummy processing code for the F1 and F4 function keys in the SAPGUI status. You don't need to define any processing when you override the default behavior — by overriding them, you prevent the system from calling the default data dictionary-defined search helps and dialogs.

Note!

If you're buying new RF hardware for your warehouse or distribution center, make sure units come with a row of prominently displayed, easily accessible function keys.

Function key	Button label	Description
F2	F2 Clr	Clears all input fields on the screen
F3	F3 Back	Return to the previous data entry screen or menu
F4	F4 Nxt	Initiate processing of current on–screen data
F6	F6 Prt	Execute printing

Figure 12 Typical function keys and tasks

Use standard function module interfaces exclusively

Anyone can post data to SAP R/3 by running a CALL TRANSACTION to populate expected data entry fields and invoke application processing by sending processing codes to step through screens. Although this approach appears to work well at design time, it is prone to numerous unexpected problems during production execution. Dialogs are famous for this. Here's a short list of what could cause your well-crafted transaction to fail to post when using a dialog:

- The operator enters a material whose master data is configured differently than the master data used during design. As a result, the CALL TRANSACTION fails when this material causes an unexpected screen to pop up.
- The operator changes master data in production after you have deployed the transaction. Now applications that worked fine before are failing. What's more, the error messages that end users receive are unintelligible because the underlying cause is technical, not functional.
- An SAP patch or SAP Note changes the screen layout or function key mapping, resulting in unexpected processing.

A better approach is to use standard SAP function modules. If possible, use BAPIs because their locked interfaces ensure backwards compatibility across releases. Even if SAP changes the underlying processing logic, the locked interface guarantees you still receive concise functional information regarding why the posting failed.

The downsides to this approach are that not all function modules are well documented, the available documentation can sometimes be limited or cryptic, and most standard interfaces have enough optional input fields to make your head spin. A former colleague once quantified the extreme number of input fields with what he called the 80-20 rule. Using just 20% of the input fields, you can achieve 80% of all processing scenarios. From another perspective, 80% of the input fields are needed only for a small set of specialized posting scenarios or are unnecessary — they exist only for interfaces' backwards compatibility.

Note!

Always check the BAPI documentation to ensure it is released for use and verify that it requires an external COMMIT WORK after execution. Most BAPIs need an external COMMIT WORK. If you don't execute one, the system does not post the data back to the relevant database tables.

The challenge with standard interfaces is knowing which fields to populate, with what data, and under which circumstances. A logistics expert can usually get to the heart of the interface quickly, mostly due to the countless hours spent on past projects tinkering with input values and reverse-engineering business logic. Unfortunately, no quick fix is possible here. However, the following two transaction codes can help ease your pain:

- **BAPI** lists all released BAPIs by function module and business object, which lets you quickly isolate potential interfaces.
- **SE37** provides the "Test Tool" feature, which allows you to execute test processing with different sets of test data after you select the candidate interface.

If you manage to track down the correct interface parameters, you can rest assured that your code remains stable. Even if posting fails, you'll always receive a failure message that pertains to the root functional cause and is not the result of a random technical error.

Conclusion

Armed with this knowledge and a "can do" attitude, you too can master the subtle but substantial design challenges inherent to SAPConsole development. In this article, I've covered topics from general development philosophy, to high-level design constraints, to the practical application coding constructs in real-world logistics solutions. In summary, here are some of the key design and development concepts described in this article:

- Keep your SAPConsole transactions to three screens or less.
- Direct users down a well-defined path to completion — field by field.
- Perform data validation as soon as data is scanned or entered.
- Minimize or avoid searching.
- Post data by individual document item as opposed to collecting item data and attempting to send all data back to SAP together in a single posting.
- Minimize the use of flag variables and top include sharing.
- Validate data at both the field and screen level.
- Mimic the standard SAPConsole function key mappings where possible.
- Favor standard function module interfaces for data posting.

This will help you design SAPConsole applications that are as simple to use as they are to maintain. If you stick to the guidelines, you'll be well on your way to becoming an SAPConsole guru.