Editor's Note: With all the talk about NetWeaver and SAP® Java Connector and other hot tools, you’d think ABAP was a thing of the past. But not so, as BW expert Pat Pesa can attest. In this excellent "best of" article, Pat shares the most valuable ABAP techniques he has used on BW projects. He starts by addressing the key questions: What are the uses of ABAP in BW? How is programming in BW different than programming in R/3? What ABAP tools are available in BW? Then, Pat digs into some technical examples of ABAP-BW in action. This article provides technical managers with a handy overview of Pat's ABAP-BW methodology, and gives ABAP programmers a technical guide they can refer to on their BW initiatives.

This article will focus on the concepts and strategies that constitute best practices regarding the use of ABAP programming in BW. On each project I have been involved with, the use of ABAP programming has had a profound effect on our ability to model the correct business processes within BW. During the design and blueprint stage, a core functional process is the determination of the corresponding business logic found in R/3 or legacy data systems and how to accomplish the same logic in BW. Having said that, we can now examine what possibilities exist for using ABAP to customize our BW environment.

In order to understand how ABAP is used in BW, we should have a general understanding of the infrastructure of BW.

This article will begin with an overview of the prospective involvement of ABAP code, where we can find this code, and how best to include coding for specific areas without impacting other functionality. Our examples for this article will include R/3 data-source modifications and the creation of generic extractors in R/3 for use with BW. To get the ball rolling, I would like to start off with the following considerations. Remember that in SAP BW, most everything is interconnected, and we share a great deal of data amongst several data targets. Therefore, we will need to be specifically aware of how we proceed with our data or record updates and where we decide to place ABAP programming that will manage these updates. Having said that, we can now examine what possibilities exist for using ABAP to transform our data as required.

What Can We Do With ABAP in BW?

As with anything, in SAP the words and phrase "ABAP programming" routinely surface as topics and questions during implementation. However, between SAP R/3 and BW, there are distinct differences between how we can use ABAP programming in these environments and what we can expect as an outcome of development. In order to understand how ABAP is used in BW, we should have a general understanding of the infrastructure of BW. Generally speaking, in BW, we are dealing with Objects that we connect with some type of configuration, such as the linking of an ODS with an InfoCube, or the creation of source system definitions and datasources. What we will NOT be doing on a general basis in BW is building screens and multiple user exits in ABAP to gain additional functionality. This is the distinct difference in BW as it relates to R/3. So if we’re not using ABAP for user exits and screens, then how will we use ABAP? In BW, we will use ABAP to transform our data as requirements dictate, as well as provide additional processing during extraction and updates.

1 One thing to note: we are provided with one user exit in SAP R/3 that will enable the management of data being extracted and one user exit in BW that will enable the management of data being queried.
Within the ABAP language are certain constructs that allow us to get work done within this environment. In SAP, we will be using the ABAP Workbench to create our code. This is true for each instance we are writing code, whether it is from within a transfer rule, update rule, or start routine. It is important to understand the fundamental concepts of the ABAP workbench in order to proceed. First, we should be familiar with the constructs of programming, such as understanding what a sub routine does, how a function module works, etc. Also, we should understand the different avenues we may use in order to debug our code. This last measure is of extreme importance, since it will be necessary to debug code from time to time in order to create what our requirements dictate.

For the purposes of this article, we will assume that the reader already has a grounding in these basic ABAP concepts, including the ABAP Workbench.

Our approach to ABAP development in BW will follow the simple guidelines found in Figure 1.

Where Are We Able to Create ABAP Programming in BW?

An often-asked question prior to the beginning of a BW project revolves around how much ABAP programming will be necessary. Following the construct of a typical R/3-BW integration, one would assume that all ABAP would be done by a team of programmers and that the BW consultants would furnish documentation around the changes to be made. However, in BW, the line between functional and technical is often much more gray than the distinction in R/3. To cite one example: we can configure objects in BW, thus gaining a great deal of functionality, without writing an array of user exits as found in R/3.

Let’s take a look at where we are able to augment our BW development with ABAP and what that may mean to the system performance. Generally, when adding code, we can assume some type of run time overhead in the form of processing time or memory impact. However, when programmed in an expedient and concise manner, the overhead may be insignificant. At this time, we should familiarize ourselves with where BW will permit customer-provided ABAP programming.

Now let’s look at the options found within BW that allow the additional implementation of ABAP programming (see Figure 2).

The list in Figure 2 represents the areas within BW that may require ABAP expertise during a BW implementation. This list cov-
ers 99 percent of the examples of ABAP development in BW. I say that because there is always that one exception in which a customization has been done outside this matrix. Now that we have covered the what's and where's of ABAP programming in BW, we can have a look at the how's.

How Do We Manage Our Programming During the Development Phase?

Let's start at the beginning. By the beginning, I mean the point on our BW project where we first start looking in R/3 and figuring out what we can modify prior to or during extraction. As a rule of thumb, the concept here will be as follows. In R/3, we have access to the raw transactional data set we will want to modify. We have several options here, but primarily, we can either modify an existing data extractor or create a new customized extractor from the ground-up. Let's look at the two options in the table in Figure 3.

Option 1
This is the most straightforward modification in ABAP and is commonly employed to add data requirements that do not exist in the standard extractor. We do this using the basic concept of append structures for our additional fields. Once we have added our fields and activated our extract structure, we will be able to see these fields in the user exit.

A commonly overlooked step in adding fields to extract structures and filling them with data is actually making the newly added fields visible to the BW target system. Once the fields are added to the extract structure and you have saved and activated both the append structure and the extract structure, you will need to go to transaction code RSA6 -> Post Processing Datasources and find the datasource you modified. Double click on that datasource and make sure you uncheck the "hide" check box for the added fields. Otherwise, these fields will not show up in the newly replicated datasource in BW.

At this point, we can look at how we can populate these fields with data in the R/3-to-BW user exit in R/3. To provide some background, I will briefly mention how to navigate to the user exit we want to use. Using transaction code CMOD, go to your BW project, i.e., "Z_BW". This project will be found within the customer namespace. Once you have found the correct project, choose "display components" and hit F7. This will display the BW user exit components. You should find four user exits. The configuration of these user exits may differ from system to system. Figure 4 contains the descriptions of the four user exits found in a CMOD BW Project.

To provide an example, let's consider a standard-delivered data extractor found within Logistics Cockpit. This extractor is responsible for extracting purchase order item data found in R/3. For our example, let's assume you have added several fields to the 2LIS_02_ITM extract structure.

<table>
<thead>
<tr>
<th>User Exit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAPLRSAU_001</td>
<td>Transactional data user exit</td>
</tr>
<tr>
<td>SAPLRSAU_002</td>
<td>Master Data User exit</td>
</tr>
<tr>
<td>SAPLRSAU_003</td>
<td>Optional</td>
</tr>
<tr>
<td>SAPLRSAU_004</td>
<td>Optional</td>
</tr>
</tbody>
</table>

Figure 3: ABAP Programming Options in R/3 for BW

Figure 4: R/3 User Exit Examples in BW
You now want to add your programming logic to fill these fields and then re-initialize BW. You will double click on SAPLRSAU_001 and you will be taken to the correct include file for the top-level program SAPLRSAU_001. Once there, you will double click on the include ZZRSAU01. This include is where all your customer logic will be written. Let’s examine the structure of this include and lend some insight into how and where to add our code within this include. This include, or user exit, is based on case logic, meaning we will use a case statement to determine where we will land within the exit during runtime. Please refer to the structure found in Figure 5 within the include. Note that you may have to create the case logic prior to adding your additional logic.

From this point, you will use "when" statements to control entry into your logic. An example would be Figure 6.

Once you begin extraction and the extract structure has been filled with the initial data from standard extracts, you will find yourself within the When logic and ready to begin to append data to each record. In this user exit, you will loop through each record and append the necessary data. Then, in the final step within the loop, you will add all the new data elements to the actual extract structure and return to the next record. Loop like this through each record until you are finished with each record in the initial extract structure. Figure 7 illustrates ABAP written to append account assignment data to a purchase order item.

While the code in Figure 7 is not all-inclusive, it clearly illustrates how to integrate your ABAP programming into the correct user exit.

**Option 2**

Now let’s look at Option 2 (reference Figure 3). In Option 2, we are going to create a generic data-
source using standard SAP R/3 technical components. Much of the actual extract code will be created for our use; we will need only to specify several aspects, such as the extract structure and function module. Or, we can use a view and allow R/3 to generate everything for our use. Please keep in mind that once we complete this step, we can gain the same access as illustrated in option 1 during extraction. Before we get into examples, let’s review our options in creating a generic extractor found in figure 8.

Creating a view in R/3 permits easy definition of our extract based on several tables. Here are several important considerations in creating a view in R/3:

1. When using multiple tables, you will have to provide the joins for the tables based on the keys found in each table. If you don’t, then the system will make each field within your view a key for the view. This will impact view performance during runtime.

2. You will have to include unit references for quantity and price data elements. You should use the same reference field found within the data element itself. This reference field is simply added to the view.

3. You should consider using the top-level tables within your view even in the event you will not use fields from these tables. For example, when creating a view from "Material Plant data", MARD, you should join the "Material Master general data", MARA, document number with the MARD document number and MANDT "client" to ensure that you will get only the records you want and not multiples on the same record across clients.

Now, taking into consideration these tips on creating a view, we can build our view so that it contains the fields we have identified and want in the generic data extract.

1. Calling transaction SE11 and setting the radio button to View, we input the Z****** name for our view and click Create.

2. When prompted for a development class, input the correct development class and click save. This will bring us to the initial view definition screen.

3. Begin by adding the fields you need, supplying the base table names as well. You will need to complete all the steps required to finish the view.

4. Once done and active, check to see how many records your view returns and compare that number to the number of records found in the base table(s).

5. Once you have your view completely finished, you can go to transaction RSO2.

6. Input the view name, making sure the correct options are checked for using a view instead of an extract structure/function module and click the save button.

7. If you’re successful, you can go to RSA3 and run the extract checker for your new datasource.

8. You should double check the RSA3 output with the view definition as a unit test. This will give you a preliminary fitness test for the generic datasource.

Now let’s consider the second option in creating a generic datasource. First, you will need to define a structure that will serve as an extract structure for your program. In creating an extract structure please refer to the following rules:

- Make sure you include units for quantities and price data.
- Make sure you include the appropriate keys for the data you will be working with, such as document numbers and/or item numbers.

As for the extract function module, you should plan on creating a simple function module and following the guidelines found in the example function module RSAX_BIW_GET_DATA.
Following this example will ensure compatibility creating the generic extractor with the structure/function module combination. In this article, we will not detail the creation of a function module based on RSAX_BIW_GET_DATA, but I will note that it will be relatively simple to create a datasource once the function module is complete. The extractor definition is created in the same manner as the first example using RSO2, but we will specify our structure name as well as the function module instead of the view. Once it is saved and active, it is ready. Also remember that you may make changes to the logic in both the function module itself or within the user exit SAPLRSAU_001.

**Conclusion**

In the end, creating a customized user experience is really what it is all about, and it is up to the development team to implement the correct changes within the environment that make a positive user experience a reality. The use of ABAP in the BW environment is widespread and is impossible to cover in its entirety within this article. However, it is most certainly possible to get a solid grasp of the concepts required to understand what, when, and how of ABAP programming in BW.

I have attempted to layout, in logical order, the basic areas in which ABAP is used, when it is appropriate to create customized logic, and how to go about programming the environment to get what you want out of R/3 and into BW. Specifically, we covered the relevant aspects of the R/3 environment, such as the user exit SAPLRSAU_001, and how to implement basic modifications. We looked at how the user exit works and how it is called during runtime. We detailed the basic examples of creating generic extractors using base tables in R/3 and in views or function modules, and we laid out specific tips regarding the successful creation of these components and how they interact.

Remembering the many things that can be done in BW using ABAP is the key to successfully addressing the requirements of customized reporting. We have furnished you with the basic concepts and included examples of real situations and logic used in production environments. It is now up to your team to determine how you may move your environment to the next level.

**Arthur "Pat" Pesa, arthurpesa, inc.** Pat is a senior independent consultant specializing in SAP Business Warehouse and Application Link Enabling. He has over nine years of SAP R/3 experience, specializing in ABAP/BAPI development, Internet based order-processing systems, and Demand Management integrations with third-party Demand Planning systems. Pat has spent the last three years as a Business Warehouse consultant, specializing in technical leadership roles and solution design and development. Pat’s email address is Arthus.Pesa@SAPtips.com.
The information in our publications and on our Website is the copyrighted work of Klee Associates, Inc. and is owned by Klee Associates, Inc. NO WARRANTY: This documentation is delivered as is, and Klee Associates, Inc. makes no warranty as to its accuracy or use. Any use of this documentation is at the risk of the user. Although we make every good faith effort to ensure accuracy, this document may include technical or other inaccuracies or typographical errors. Klee Associates, Inc. reserves the right to make changes without prior notice. NO AFFILIATION: Klee Associates, Inc. and this publication are not affiliated with or endorsed by SAP AG. SAP AG software referenced on this site is furnished under license agreements between SAP AG and its customers and can be used only within the terms of such agreements. SAP AG and mySAP are registered trademarks of SAP AG. All other product names used herein are trademarks or registered trademarks of their respective owners.