INTRODUCTION TO MY 1988 SMP ARTICLE

My article, which explains the basic concepts of SMP and SMP/E, was written in 1988, and is still almost completely applicable today. I'm going to show you, in a few pages, how to understand SMP/E. Let's first just say a few vitriolic, but true, introductory words.

IBM has a habit of "only touting its latest stuff". They tell us: "Here's our product SMP/E, which helps you maintain your MVS system!" What they don't tell you are the concepts that are necessary to grasp, so an intelligent person can get a basic understanding of SMP/E ideas in a reasonable amount of time.

In order to understand SMP/E, you have to understand where it came from. SMP/E came from an effort to automate and keep track of the SYSGEN process, and how to put subsequent fixes on an already-generated MVS operating system. It is my fervent and passionate belief, that if you don't understand the SYSGEN process, you'll never have a clue to really understanding what SMP/E is all about.

Problem: We don't do SYSGENs any more today, to construct an MVS system. The reason is that IBM has done them for us, before they shipped us the new system. The information from their SYSGEN of our system is buried in the JCLIN information, in the target SMP/E control datasets, of the system that they ship us.

Previously, we could not dig that information out. Now we can. In the newer SMP/E releases, after Release 1.3, there is an "un-JCLIN" process in SMP/E, otherwise known as the GENERATE command. Using GENERATE, we can re-create assembly and link-edit information which is buried in the SMP/E datasets--the same information, like the SYSGEN information, that tells us how the pieces of our system were put together in the first place.

So, we begin our journey as I take you back in time--to a time that IBM will never admit ever existed--to the (ancient) time when SYSGENs were done, to build MVS systems. We must go back there. Because, if we don't ever go back there, we'll never have a clue to understanding what IBM is selling us now--this nice, neat, clever, intricate SMP/E product.

This is the truth. Isn't "official IBM" great? They don't give us what we need, because it's based on history, and they can't admit that history exists, because they want to sell us their newest stuff!

That's why we have USERS helping each other, and that's why I'm helping you. It took me 4 long years to learn what could have taken 3 weeks, with a decent explanation, that wasn't there. Once I learned the stuff, I vowed to provide the explanation too!

So let's go learn about SMP/E from the beginning--the beginning of time!

SMP DE-MYSTIFIED

Sam Golob
MVS Systems Programmer

Many otherwise-expert MVS systems programmers shy away from doing operating system maintenance. Many new MVS systems programmers have a very hard time getting started with doing this necessary work. That is not because the work is intrinsically difficult, but because IBM, until recently, has not provided to-the-point introduction to their very good SMP maintenance products. We propose to do so. After these several pages, the reader should feel much more confident with IBM's SMP products and SMP literature.

SMP stands for "System Modification Program". It is exactly that. SMP does every operation necessary to maintain the MVS operating system: source updates, assemblies, linkedits, copy operations, and zaps. It keeps a thorough accounting of everything it did, and it also allows the user to back out bad changes to the system. Every facility provided in SMP is for this purpose, and this purpose only.

The PURPOSE OF SMP must be constantly borne in mind when trying to navigate its forbidding vocabulary. All of the strange words: JCLIN, UCLIN, FUNCTION, DLIB, RESTORE, RELFILE, and so forth, are only labels for things that are NECESSARY TO KEEP TRACK OF THE OPERATING SYSTEM MAINTENANCE. THEY ARE NOTHING MORE THAN THAT. We'll try to straighten you out on most of them, so they are put in their proper perspective and their purpose is understood.

Just for the record, IBM distributes two types of SMP. SMP Release 4, known as SMP4, is the old version. It is provided free with the operating system, and it works. It just doesn't have so many bells and whistles. IBM has a much more enhanced product, known as SMP Extended, or SMP/E. A large majority of MVS installations use SMP/E, and its use is recommended. Most of the principles I will discuss apply equally to both versions, since my purpose is to provide enough insight and perspective for the reader to be comfortable with the IBM literature. The name "SMP" will suffice for reference to things applicable to both versions. Features particular to SMP/E will be labeled as such.

We must begin by describing how IBM has historically distributed its MVS Operating System releases. That process is known as "SYSGEN". How the SYSGEN process works must be clearly understood before one can hope to get a clear picture of how SMP works. IBM does not describe the contrast and connection between SYSGEN and SMP clearly enough in its SMP literature. Therefore, we will now discuss how the SYSGEN process works.

MVS was always (and is still) extremely flexible to the individual customer's needs and hardware configuration. Each customer's operating system was intended to be configured by the customer, and not just dumped in by the manufacturer, as is true with many other operating systems. IBM therefore distributes the individual pieces of the system, and it is up to the customer to put them together so that he creates a complete working system that is appropriate to his needs. Since the customer does not have the smarts of the IBM system designers, IBM gives him the means to accomplish the feat of system construction in a rather straightforward way. This is what the SYSGEN is.

The individual components of MVS, whether they be assembler macros, source, linkedited csects, sample JCL, or other ELEMENTS, are distributed to the customer in individual partitioned datasets or libraries. These are called DISTRIBUTION LIBRARIES appropriately. Members of Distribution Libraries can be pictured as the BRICKS FROM WHICH THE BUILDING WILL BE BUILT. Where then is the PATTERN or DESIGN for the building? The pattern is put into a system of assembler macros known as "SYSGEN MACROS". The SYSGEN MACROS contain the smarts for building the operating system, and we must trust that IBM supports them correctly.

What must the customer do, then? The customer must code a system configuration deck of assembler statements which contain the SYSGEN MACROS that are appropriate for his installation's hardware and software configuration. IBM refers to this deck as the STAGE I SYSGEN DECK. A working operating system, called the TARGET SYSTEM, cannot be made unless a STAGE I SYSGEN DECK has been coded, either by IBM in a sample generated system, or by the customer.

What next? The STAGE I DECK is assembled, using the DISTRIBUTION LIBRARIES as the source for the SYSGEN MACROS. The assembler output is a large JCL stream divided into a number of JOBS, usually six of them. This output, which has assembly, linkedit, copy, and other steps in it, is called the STAGE II DECK. The STAGE II DECK has the actual JCL which creates the executing operating system libraries (called the "TARGET LIBRARIES") from the building blocks in the distribution libraries. In essence, the STAGE II DECK BUILDS THE BUILDING FROM THE BRICKS. This is the gist of the SYSGEN process. Most of its smarts come from IBM, which coded the SYSGEN macros to correctly build the TARGET OPERATING SYSTEM.

The logical next question is: What if a piece of the system doesn't work? How do you correctly replace that piece in the context of the whole system? Do you have to do the entire SYSGEN all over again? The answer of course is NO, but the principle of what happens here is of utmost importance to our topic. THIS IS WHAT SMP AND ITS PREDECESSORS ARE ALL ABOUT.

Let us begin to answer the question of how to replace a bad piece of the system--a bad brick, so to speak. Put a good brick into the same place. If the bad component is in source form, an update to the source and a reassembly and relinkedit must be done. If the component was distributed in object code form, the new object code must be relinkedited to replace the old csect in the appropriate load module(s). Again, how do we know the PATTERN where the new csect fits into the system? We know it from the SYSGEN STAGE II DECK. That's how that csect got there originally. IBM fixes to the system, called "PTFs" or "Program Temporary Fixes" were originally applied to the system libraries by pulling out the appropriate job steps from the SYSGEN STAGE II deck and replacing the offending csects by IBM's replacements. A program called AMAPTFLE, which is probably not used today, was an aid in that process. This historical anecdote, if you will, gives insight into what SMP was designed to do automatically.

We have to answer one more question before we get into how SMP operates. What if IBM changes the PATTERN of how it is putting the system together. Suppose for example, that IBM breaks one module into two or three modules. In that case, IBM will change the appropriate SYSGEN MACRO(S), and a new SYSGEN STAGE II DECK will have to be assembled. The STAGE I DECK that was coded by the user does not have to be changed. It must only be reassembled against the changed macros. The part of the SYSGEN STAGE II that is affected by the module change will have to be rerun. The new TARGET LOAD MODULES to be executed will then be linkedited correctly according to the new pattern designed by IBM.

Now we are in a position to start talking about what SMP does. SMP automatically accomplishes the processes we just talked about, with the exception of assembling the STAGE II SYSGEN DECK. Once the STAGE II DECK is created and fed into SMP (through a process known as "JCLIN"), SMP "knows the pattern" of how the TARGET SYSTEM is put together, and it can apply system changes in an automated and audited way.

I must emphasize from the start that SMP is designed primarily for APPLYING FIXES TO AN ALREADY CREATED SYSTEM. Its accommodation to creating a completely new system is just that--a necessary accommodation, but once the structure of a new system is fed into SMP, using the SMP process called "JCLIN" of the SYSGEN STAGE II DECK, fixes can be applied to that new system speedily and efficiently.

There is one more fact of great value here. IBM made an important decision once SMP (which is an efficient way of applying system fixes) was invented and debugged. It was no longer necessary for them to REPLACE ENTIRE RELEASES OF MVS, and require a NEW SYSGEN each time. With SMP comfortably in place, many extensive system revisions could be accomplished as A SUM OF INDIVIDUAL MODULE REPLACEMENTS, or PTFs. The application of a large number of PTFs thus ELIMINATED THE FREQUENT DISTRIBUTION OF NEW MVS RELEASES. Therefore the PRESENCE OF SMP CHANGED MVS MAINTENANCE from NEW RELEASE orientation to INCREMENTAL PTF organization. This means, in other words, a change in emphasis from RUNNING ENTIRE SYSGENS to the SUCCESSIVE APPLICATION OF PTF FIXES TO THE SYSTEM.

Now, onward and upward to the mechanics of SMP, both in handling fixes, and in handling a major system change.

The single basic unit of work in SMP is called a "SYSMOD", or "system modification". There are four kinds of SYSMODs: these are "FUNCTION" sysmods, "PTFS", "APARS", and "USERMODS". Every SYSMOD has a seven-character ID, which must begin with an alphabetic character.

A FUNCTION SYSMOD basically represents A SEPARATE PRODUCT. One piece of MVS, such as "data management", may be composed of a number of separate FUNCTIONs, each of which owns a distinct "piece of the piece". The currently consolidated data management package of MVS/370, known as DFP, consists of three FUNCTIONs. DFP replaces a conglomeration of over thirty separate FUNCTIONS which covered the data-management area before. The seven-character SYSMOD ID of a FUNCTION has a special name. It is called an "FMID", or "Function Modification IDentifier". EVERY SYSMOD OF ANY TYPE MUST BE OWNED BY AN FMID. In other words, every SYSMOD must belong to a product (a FUNCTION) which OWNS it. Every piece or ELEMENT of the system MUST ALSO BE OWNED by a unique FMID.

A FUNCTION SYSMOD can belong to another FUNCTION or FMID, or it may be a PRIMARY FUNCTION. A PRIMARY FUNCTION SYSMOD is the only SYSMOD that is not owned by another FMID, since it is, by definition, the base level of a program product.

A real illustration of this is the control program of MVS/370. The BASE FUNCTION of MVS/370 has the FMID of EBB1102. EBB1102 is the original MVS release 3.8, and it does not belong to any other product. MVS SP 1.3.0 is a rewriting of many (but not all) of the modules of MVS 3.8. MVS SP 1.3.0 has its own FMID of JBB1326, which BELONGS TO THE FMID EBB1102. The modules that were rewritten BELONG to FMID JBB1326. Now MVS SP 1.3.3 is in turn a partial rewriting of MVS SP 1.3.0. Therefore, the FMID of MVS SP 1.3.3, which is JBB1329, BELONGS to the FMID JBB1326, which in turn belongs to EBB1102, the base MVS release. The MVS SP 1.3.3 revised modules belong to FMID JBB1329. Finally, the current MVS/370 release, MVS SP 1.3.5, is a rewriting of some of the modules of MVS SP 1.3.3. Therefore all of its rewritten pieces belong to its FMID of JBB1356. JBB1356 itself belongs to JBB1329, which in turn belongs to JBB1326, whose primary FMID is EBB1102. Please notice that EBB1102 was never superseded by the higher levels of MVS/370. This illustrates the product relationship of FMID numbers, when whole sections of a base product have been revised by IBM.

On the other hand, separate non-overlapping products will have separate, completely unrelated FMIDS. The important fact is that every SYSMOD that is not a FUNCTION SYSMOD, and many FUNCTION SYSMODS themselves, MUST BE OWNED BY AN FMID. No SYSMOD or system objects (called "ELEMENTS" in general) can belong to more than one FMID. All ELEMENTS (macros, load module csects, etc.) must belong to one and only one FMID.

We are now at the point where we want to know several things. First, what does an SMP environment consist of? Second, how do SYSMODS get onto an SMP-controlled system. Third, how do the ELEMENTS, the PIECES OF THE SYSTEM, relate to the SYSMODS. Fourth, how does all this SMP stuff fit in with what we already know about the SYSGEN process, and how does SMP build an actual working operating system?

Answering these four questions constitutes the goal of this article. With an understanding of the answers, any person can easily negotiate IBM's SMP literature and thereby gain real competence in SMP use. (BOB, PLEASE PUT EMPHASIS ON THIS PARAGRAPH - - -.)

Let us discuss the SMP environment. Roughly, the concept of an SMP environment breaks down to two things: first, a set of LIBRARIES controlled by SMP, and second, special SMP CONTROL DATASETS which keep accounting of all the system changes that occurred. A sequential SMP LOG records and time-stamps all significant actions which are occurring during SMP processing.

The set of libraries controlled by SMP is broken down into two types of libraries. There are TARGET LIBRARIES, which contain copies of actual working system code. And there are DISTRIBUTION LIBRARIES, which have the pieces from which the system is built. The concept of DISTRIBUTION LIBRARIES and TARGET (or "SYSTEM") LIBRARIES corresponds exactly to what we discussed before, concerning the SYSGEN process. In fact, there is no difference. We have seen that with the SYSGEN process, the DISTRIBUTION libraries are SUPPLIED INITIALLY by IBM and the TARGET libraries are CREATED FROM THE DISTRIBUTION LIBRARIES using the STAGE II SYSGEN DECK. With SMP however, (once an initial operating system has been created) the TARGET LIBRARIES are supplied first, using material sent by IBM in the form of SYSMODS. These are plugged into the actual working operating system libraries by SMP. Then they are tested, and if the changes are good, they can then be put into the DISTRIBUTION LIBRARIES PERMANENTLY. Thus, the usual SMP flow is from IBM-supplied SYSMODS, to TARGET LIBRARIES, and then to DISTRIBUTION LIBRARIES. This is the opposite of the flow of changes during the SYSGEN PROCESS.

Our discussion of control datasets must be based on the three PROCESSES OF SMP FLOW. These are called "RECEIVE", "APPLY", and "ACCEPT". RECEIVE, APPLY, and ACCEPT are the SMP-language equivalent of: taking a new SYSMOD into the SMP environment, putting its pieces into the TARGET libraries into the proper places, and finally, storing its pieces in the DISTRIBUTION libraries for archival and possible later use. Again, the flow is: IBM-supplied SYSMODS, to TARGET libraries, to DISTRIBUTION libraries.

How do the RECEIVE, APPLY, and ACCEPT processes work? We shall discuss them briefly in order. Please bear in mind that our aim is to simplify the discussion and keep it conceptual. Once the concepts are understood, and the vocabulary words associated with them are learned, then the IBM literature will become quite readable.

RECEIVE basically involves STORING A NEW SYSMOD and recording some vital statitstics about it, so that it is pre-digested a bit by SMP. The text of the entire SYSMOD is stored in a partitioned dataset called the "PTS" or "PTF Temporary Store Data Set". In the older version of SMP, SMP4, the control information taken during the RECEIVE process is stored in the PTS also. However, in the new SMP/E, all control information is stored in VSAM files known as "ZONES". The VSAM file which stores the control information from the RECEIVE process is known as the "GLOBAL ZONE". The text of the SYSMOD thus goes to the PTS as before, but the control information is stored in the GLOBAL ZONE.

There are certain SYSMODS which are RECEIVED a bit differently. These are SYSMODS which have what are known in SMP language as "RELFILES". The word RELFILE roughly translates to "IEBCOPY". If a modification has large numbers of linkedited csects, macros, ISPF panels, or other ELEMENTS, which are suitable for direct IEBCOPY into TARGET LIBRARIES, then those ELEMENTs included in the SYSMOD which have SIMILAR RECORD FORMAT AND RECORD LENGTH can be loaded into a single pds. This pds is unloaded onto the SYSMOD distribution tape using IEBCOPY, and its file sequence order on the tape is very important. If the Sysmod Specification File on the tape (known as the "SMPMCS") calls for THREE RELFILES for instance, then the next three files on the tape, hopefully IEBCOPY unloaded, are known as RELFILE(1), RELFILE(2), and RELFILE(3) respectively.

The RECEIVE process on a SYSMOD with RELFILES is then the following: The SMPMCS, which contains the SMP control statements of the SYSMOD, is read from the tape. The number of RELFILES (3 in our example) is determined from the "RELFILES (3)" keyword in the SMPMCS. The three files on the tape which follow the SMPMCS file are then unloaded to a disk pack (determined by the "SMPTLIB" DD statement in the SMP job) and given special names, which SMP will understand during later APPLY and ACCEPT processing of the SYSMOD. These names are of the format: PREFIX.sysmodid.Fn, where n is 1, for the first relfile, 2 for the second, and so forth. The prefix is set in the global zone for SMP/E (or in the PTS system entry in SMP4) and is not changed during the entire process. Therefore, when a SYSMOD with RELFILES is RECEIVED, the relfile libraries are loaded from tape to disk. The elements they contain are thus readied, so that the APPLY and ACCEPT processes to follow can selectively copy them to where they will be needed. Meanwhile, the SMPMCS is itself copied to the PTS, and its control information is dealt with as we discussed before.

With SMP/E there is a bit more to the RECEIVE process. Besides the RECEIVE of SYSMODS, one must also RECEIVE SYSMOD ERROR INFORMATION. This is known as "HOLDDATA". HOLDDATA consists of a list of "++HOLD" and "++RELEASE" control statements in sequential order. Each such statement points to a PTF or an APAR SYSMOD. The purpose of the ++HOLD statement is to prevent SMP/E from APPLYing a PTF that is in error, known as a "PE PTF". A ++RELEASE statement that is RECEIVED AFTER a ++HOLD statement against the same SYSMOD will UNDO THE EFFECT of the ++HOLD statement. The ORDER in which the statements are RECEIVED is of paramount importance. SMP/E will honor the ++RELEASE only if it occurs AFTER ALL ++HOLD statements for that particular SYSMOD. The HOLDDATA is kept in the SMP/E GLOBAL ZONE.

A quick word about UN-doing the RECEIVE of a SYSMOD. A RECEIVE can be undone by an SMP process called "REJECT". To REJECT a SYSMOD that has been RECEIVEd causes the SYSMOD to be erased from the PTS and its control information associated with the RECEIVE process to be wiped out. If the SYSMOD has RELFILES, the disk-loaded copies are deleted. The SYSMOD cannot then be RE-APPLIED or RE-ACCEPTED unless it is RE-RECEIVED. REJECTing a SYSMOD WILL NOT AFFECT THE STATUS of a SYSMOD that has ALREADY BEEN APPLIED OR ACCEPTED. It stays in the TARGET libraries and in the DLIBs.

OK, we've talked about getting the SYSMOD into SMP. Now is a good time to mention the syntax of SMP MODIFICATION CONTROL STATMENTS, before going to the details of APPLY and ACCEPT processing.

Rule number one is that all SYSMOD control statements (known in IBMese as MCS or "Modification Control Statements") must begin with the characters "++" in columns 1 and 2 of the SYSMOD's card-image records. The other keywords and parameters in SYSMOD syntax are free-flowing. Extra blanks don't count. Columns 73 to 80 are not used in the SYSMOD statements themselves, but they are required for IEBUPDTE source and macro update cards, so you have to be careful and one can't renumber a SYSMOD. Comments are as in the PL/I language. They begin with the characters "/*" and end with the characters "*/". Everything in between these specific strings, even on many successive lines, is treated by SMP as a comment. All SMP statements must end in a period, ".". The period is the delimiter for statements in SMP syntax, and one must be EXCEEDINGLY careful with them.

Every SYSMOD must begin with the statements: "++ FUNCTION", "++ PTF", "++ APAR", or "++ USERMOD", followed by the seven character SYSMOD ID enclosed in parentheses, and delimited by a period. We've already discussed FUNCTION sysmods. PTF sysmods or "PTFs" are (after the invention of SMP and the change in IBM's maintenance philosophy) really PERMANENT SYSTEM FIXES, although their name stands for "Program Temporary Fix". The TEMPORARY fixes are called "APARs", which is really a short term for "APAR FIX". APAR stands for "Authorized Program Analysis Report", and it really refers to a problem that was reported to IBM. IBM assigns a number to each problem, and when the problem is fixed, the temporary fix itself is assigned the same number. This number (or something very close to it) is what SMP uses for the APAR SYSMOD ID. APAR SYSMODS are intended to be replaced, or "SUPERSEDED" by permanent fixes, or PTFs. They are therefore not usually "ACCEPTED" into the distribution libraries. More about this later.

USERMODs are packaged SMP SYSMODS written by the individual installation for its own needs, usually to modify some IBM code. They look like PTFs or APARs, but SMP puts them in a special category so that they stay a bit distant from the real IBM maintenance. The user has flexibility to assign almost any seven character SYSMOD ID to his USERMOD, but each different SYSMOD must have a unique ID, and it is wise to steer clear of IBM-type names. USERMODs are also usually not ACCEPTed into IBM distribution libraries (also known as "DLIBS") because they are likely to overlay IBM code there. USERMODs should only be ACCEPTed if you really know what you are doing.

We'll quickly explain the concepts of the other strange keywords you're likely to find at the top of SMP SYSMODS. I don't intend to be exhaustive. The idea is to cut through the ice and give you some idea of what is happening.

"++ VER" means "version of the operating system", or perhaps "domain of SMP activity" would also be an appropriate explanation. The values for this parameter have 4 characters, and there aren't too many valid choices. "Z038" means MVS 3.8 and upwards thru XA. ("Z037" or MVS 3.7 probably isn't used anymore.) "C150" means CICS 1.5 and upwards. There are a few more of them. These values almost never change, and within the same domain of activity, you always find the same value. The ++VER values were originally intended to differentiate between similar maintenance on different MVS releases, but because of the evolution of the SMP product since those "old" days, it doesn't have that importance any more. The purpose of the ++VER statement has been largely taken over by the FMID. It is necessary and required, however; it must be included in all SYSMODs.

Inside of the ++VER sentence (and before the period) you'll most likely see the FMID, PRE, REQ, and SUP keywords. FMID has been discussed. Every SYSMOD must belong to a unique FMID. In the parentheses following the FMID keyword, the FMID which will own the SYSMOD must be specified. PRE and SUP are quite simple. In the parentheses following the PRE keyword, and separated by spaces or commas, is a list of SYSMOD IDs which have to be present before OUR SYSMOD can be put onto the system, or "APPLIED". The "prerequisite SYSMODs" must have been already APPLIED themselves, or else they have to be APPLIED together with this new SYSMOD. Otherwise if all the PREs or prerequisites are not present, the new SYSMOD will not go on.

REQ is like PRE, but the SYSMODs that are REQuired for the new SYSMOD, must BE APPLIED IN THE SAME RUN as the new SYSMOD. It is seldom necessary to use the "REQ" keyword. Ususally "PRE" will suffice.

Following the SUP keyword is a list of SYSMODS that are effectively replaced, or "SUPERSEDED" by the new SYSMOD. This keyword is used when the intent is that the new material to be added to the system COMPLETELY OBSOLETES all the material from all of the SYSMODs in the SUP list. If only SOME of the material in a previous SYSMOD or PTF will be replaced, that SYSMOD should be placed in the PRE list or "prerequisite list". For a SYSMOD to be superseded or "SUPed" (to use the common parlance) its purpose in the operating system should be COMPLETELY REPLACED by the later SYSMOD.

Our new SYSMOD may have a sequence of "++IF FMID(fmidnam) THEN REQ(sysmdid)." statements. This will happen if we have different levels of the same product, as we mentioned earlier regarding the various MVS/370 product levels. If our SYSMOD is for a lower level of the product, and the installation has both that level and a higher level, it may be necessary to apply another fix to satisfy the requirements of the higher level. SMP is informed of this by means of a coded ++IF statement in the lower level SYSMOD. If the higher level referred to (in the ++IF statement) is not present on our system, then our fix suffices for us, and it will go on to our system normally. If our installation has the higher level of the product also, the lower level SYSMOD will not go on without the sysmod id in the REQ keyword of the ++IF statement also being present. The FMID name within the ++IF statement is usually for a higher level of the product than the current SYSMOD is for.

Below all this "version", PRE, SUP and IF stuff is the actual fix. A "++MOD" statement followed by object code signifies an object module replacement of a CSECT. A "++SRC" statement followed by source code is a complete source code replacement. A "++SRCUPD" statement or "++ MACUPD" statement followed by IEBUPDTE control cards signifies updates to an existing source module or macro. A "++MAC" statement is followed by the complete replacement for the macro. Keywords in these statements supply necessary additional instructions so that the change is done according to the author's or IBM's specifications and requirements. Specification of the destination libraries for the ELEMENT to be changed is also accomplished by these keywords.

A single SYSMOD can contain fixes for many system ELEMENTS. Each element to be changed must have its own ++SRC, ++MOD, +SRCUPD, ++MAC or ++MACUPD control card, followed by the new replacement or additional material for that element. An approximate limit to the number of element fixes in one SYSMOD is set by a global SMP parameter called "PEMAX". It is advisable to set PEMAX to a high number, usually to 9999.

One more important note. Libraries in SMP control statements are referred to by their DDNAMES ONLY. These must be one to eight characters long. It is safe and wise practice in SMP work to ALWAYS MAKE THE DDNAME OF THE LIBRARY CORRESPOND TO THE LOWEST-LEVEL QUALIFIER OF ITS DATASET NAME. DATASET NAME PREFIXES DON'T COUNT TO THE SMP PROGRAM. It is therefore advisable to use the SAME SMP JCL PROCEDURE when doing system modifications to one system. The JCL of that PROC will uniquely and unchangeably determine the destination libraries of the SMP action. Thus, the SMP control information and the actual contents of the affected libraries will always be kept in synchronization. (SMP/E has a facility for determining dataset names by dynamic allocation. These dynamically determined library names are called "DDDEFs". I want to keep the discussion here simple, and I will not dwell on DDDEFS, except to say that they DYNAMICALLY accomplish what DD cards do in a JCL procedure.)

This will take some of the mystery away from what a SYSMOD looks like. Now once it is RECEIVEd, how do we APPLY it to the TARGET LIBRARIES?

When thinking about APPLYing SYSMODS to our system, we must never forget that the PURPOSE OF A SYSMOD IS TO SUPPLY NEW COMPONENTS for our operating system or our product. THE APPLY PROCESS PUTS THE NEW PIECES OR ELEMENTS INTO THEIR PROPER PLACES IN THE EXECUTING LIBRARIES OF THE SYSTEM, the TARGET LIBRARIES. IT ALSO KEEPS DETAILED TRACK OF WHAT IT HAS DONE.

APPLY can ONLY be done to a SYSMOD that has been RECEIVED. The RECEIVE process has put the SYSMOD into the SMP staging areas, the PTS and possibly the UNLOADED RELFILES, and has done some preliminary accounting in the SMP/E GLOBAL ZONE or the SMP4 PTS. The APPLY process will pick the SYSMOD up from there. Accounting for the APPLY process is done in SMP/E using a VSAM cluster called the TARGET ZONE, and in SMP4 using several partitioned datasets, the most important of which is called the "CDS" or "Control Data Set".

A word about ELEMENT ACCOUNTING. If a SYSMOD will replace an ELEMENT (a macro or a module or source code - a MAC, a MOD, or SRC), that ELEMENT acquires an "RMID" (or "Replacement Module ID") equal to the SYSMOD ID of the SYSMOD which replaced it. Since the piece was completely replaced, each MAC, MOD, or SRC ELEMENT can have ONLY ONE RMID. If on the other hand, the SYSMOD will UPDATE A MACRO, UPDATE SOURCE, or ZAP A LOAD MODULE, then that ELEMENT acquires a "UMID" ("Update Module ID") equal to the SYSMOD ID that updated it. One macro or module can have MANY UMIDS, because it is possible to update a single ELEMENT many times, and in many ways.

Now back to APPLY processing. Many SYSMODS can be APPLIED together in one run. This is one of the great conveniences of the SMP product. When APPLYing more than one SYSMOD, the programmer can SELECT a list of individual SYSMODS by their seven-character SYSMOD IDs. This is done in the SMPCNTL DD statement, where one enters the "APPLY" parameter into the SMP job. One simply states "APPLY SELECT (sysmdid,sysmdid, ...).". We can also select just one SYSMOD to APPLY, if that is what we want.

Before we get too far, I must mention "APPLY CHECK", which is a dry run of the APPLY process. By inclusion of the word "CHECK" in an APPLY request, SMP will do a dry run. All SMP control information will be verified, just as if the real APPLY was being done. With APPLY CHECK however, no real library updates are done, and the SYSMODS will not really be APPLIED to the system. A report will be generated that tells us what WOULD be done. "ACCEPT" and "RESTORE" processing, to be mentioned later, also have the "CHECK" facility. IT IS ALMOST ALWAYS ADVISABLE TO DO APPLY CHECK BEFORE DOING THE REAL APPLY.

We continue. It is also possible to do what is called a MASS APPLY. MASS APPLY works as follows. By simply stating the word APPLY in the SMPCNTL DD statement of an SMP job, SMP will ATTEMPT TO APPLY ALL RECEIVED SYSMODS THAT ARE ELIGIBLE FOR THE SYSTEM but have not been APPLIED yet. SMP will look at ALL THE UNAPPLIED RECEIVED SYSMODS and will BUILD A SYSMOD SELECT LIST. It will then proceed to APPLY ALL THE SELECTED SYSMODS to your system. It is possible to EXCLUDE a list of SYSMODS specifically from a MASS APPLY, by using the EXCLUDE or "E" parameter in the APPLY control statement. In SMP/E (but not in SMP4) there is an additional way to exclude a SYSMOD from an APPLY. This is through the use of a "++HOLD" modification control statement which has been RECEIVED previous to this APPLY request. MASS APPLY is the normal means of APPLYing IBM's periodic system maintenance known as "PUTs" (Program Update Tapes) to the system, because this maintenance consists of large numbers of PTFs, perhaps several hundred of them, and it is inconvenient to SELECT them individually.

There is another APPLY option called APPLY with the GROUP parameter. "APPLY GROUP" works very much like "APPLY SELECT" for APPLYing a list of SYSMODS. The difference concerns missing prerequisites. If "APPLY GROUP(sysmdid,...)" is coded instead of "APPLY SELECT", and if a necessary prerequisite SYSMOD has not been included in the explicit SELECT list, SMP will go to the trouble of adding all such necessary prerequisites to the SELECT list before doing the APPLY. It is only necessary that the added SYSMODS have been previously RECEIVED. "APPLY GROUP" is good if you are not sure what other PTFs are necessary to include, when you are APPLYing new PTFs that you really want. It does use extra overhead, but it should be used for avoiding multiple SMP runs when some prerequisites may be missing.

BYPASS. This is a useful facility of SMP APPLY processing, especially during APPLY CHECK, when you're determining if the APPLY should work. The BYPASS parameter of APPLY (and ACCEPT) allows SYSMODS to be APPLIED to the system, even though they are missing prerequisites or other requirements. During a real APPLY situation, the BYPASS parameter will allow the putting on of a SYSMOD in an exceptional circumstance, when there is no other way of getting it on. During a CHECK situation however, it is advisable to BYPASS ALL OR MOST ERROR CONDITIONS on the first try. This is because SMP WILL OFTEN STOP FURTHER ACTION when it encounters the FIRST ERROR.

For instance (to use an SMP/E example), suppose SYSMODS B and C both require SYSMOD A, and SYSMOD A has a ++HOLD against it. When we try to APPLY CHECK SELECT SYSMODS A, B, and C together, and we don't have a BYPASS (HOLDERROR) coded, SMP will not tell us how the three SYSMODS will APPLY, and what ELEMENTS they will affect. It will simply say that it cannot APPLY SYSMODS B and C because it could not APPLY SYSMOD A. On our next try, if we did code BYPASS (HOLDERROR) in the APPLY CHECK, SMP/E would give us a complete report of the ELEMENTs affected by all three SYSMODS, because the BYPASS allowed the simulation of a completed APPLY, and all the consequences of the APPLY will show in the report.

As a matter of procedure, most people BYPASS every error condition during APPLY CHECK, but some try not to use BYPASS on the real APPLY. They exert much effort to completely clean up the APPLY CHECK so that they can avoid coding a BYPASS for the real APPLY run. I personally leave the BYPASS in during the REAL APPLY runs too. I just make sure that the APPLY CHECK will show the exact result that I want to achieve. My approach avoids an excessive number of APPLY CHECK runs, which can take a long time. This is a matter of personal preference. The important thing is that THE REAL APPLY SHOULD ALWAYS BE DONE WITH THE SAME PARAMETERS AS THE LAST SATISFACTORY APPLY CHECK RUN.

Our final word on APPLY processing will be to show how the PATTERN OF THE SYSTEM is determined or changed during APPLY. SYSTEM PATTERNS are communicated to SMP by means of the "JCLIN" facility. JCLIN consists of MODEL JCL statements for ASSEMBLY, LINKEDIT or COPY steps. SMP reads these statements, and uses them to determine how source can be reassembled after a MACRO change, or how load modules of one or many csects (called LMODs by SMP) are linkedited from component parts, including all linkedit control statements and attributes. JCLIN is only associated with TARGET LIBRARIES and the TARGET ZONE, not with DISTRIBUTION LIBRARIES, which have only separate pieces of the system. JCLIN tells SMP how to construct the working programs of the system from their component pieces.

This is reminiscent of our discussion of the SYSGEN process. In fact, JCLIN is the means of communicating the contents of the SYSGEN STAGE II DECK to SMP. A JCLIN stream can be put into SMP in two ways. The first way is by means of a dataset of JCL, which is referred to the SMP job by the SMPJCLIN DD statement. The second means is from within a SYSMOD itself. This second way is called "INLINE JCLIN". The JCL stream is included in the text of the SYSMOD and preceded with a "++ JCLIN." statement. This will cause the JCL pattern to be read into the TARGET ZONE or CDS when the SYSMOD is APPLIED.

The APPLY process is reversed with the RESTORE process. RESTORE takes off SYSMODS that have already been APPLIED but NOT ACCEPTED. SMP provides a RESTORE CHECK facility so the user can determine in advance if the RESTORE will work. RESTORE allows you to back off bad SYSMODS if they are causing trouble during your system tests, after APPLYs have been done.

We shall talk about ACCEPTing the APPLIED SYSMODS into the DLIBS. The ACCEPT process of SMP takes the ELEMENT REPLACEMENTS which have already been APPLYed to the system (and presumably tested), and puts them into the DISTRIBUTION LIBRARIES to archive them. This also allows them to be used in a subsequent SYSGEN or IOGEN (a partial SYSGEN to rearrange IO-configuration related ELEMENTS). ACCEPT is thus a very important process.

Control information for the ACCEPT processing is kept, for SMP/E, in a VSAM cluster known as a DLIB ZONE. Each DLIB ZONE must be paired with a corresponding TARGET ZONE. This is because SMP will normally ACCEPT only SYSMODS that have already been APPLIED, and the TARGET ZONE has the control information for the APPLIED SYSMODS. It is possible for ONE GLOBAL ZONE to control SEVERAL PAIRS OF TARGET AND DLIB ZONES, so that several sets of system libraries can be maintained out of one SMP/E configuration.

For SMP4, control information for ACCEPT is kept in two partitioned data sets, the "ACDS" or "Alternate Control Data Set" and the "ACRQ" or "Alternate Conditional Requisite Queue Data Set". (There is also a "CRQ" dataset for APPLY information.) These datasets correspond to the information that is kept in the DLIB ZONE for SMP/E.

ACCEPT control parameters are similar to those for APPLY, as we discussed above. It is always advisable to do ACCEPT CHECK runs before doing a real ACCEPT run, just to make sure the job will achieve the desired result. Doing ACCEPT CHECK runs will make it possible to fix errors before the ACCEPT run. Once a SYSMOD has been ACCEPTed on the SYSTEM, it cannot be removed, except by another SYSMOD that will SUPERSEDE it and replace all its ELEMENTS.

In order for an ELEMENT to be used in a subsequent SYSGEN or IOGEN, its SYSMOD must first be ACCEPTED. This loads the ELEMENT into its proper DISTRIBUTION LIBRARY. As we stated before, the SYSGEN PROCESS uses the material in the DLIBS to build the system, and one doesn't want back-level code to creep into his system after an IOGEN has been done. It is therefore mandatory to do a MASS ACCEPT before a SYSGEN or IOGEN is done.

There is a special case of ACCEPT processing which allows SMP to simulate the SYSGEN process. This type of ACCEPT processing is called "ACCEPT NOAPPLY", and its purpose is to replace SYSGEN MACROS and other pieces of the DISTRIBUTION LIBRARIES so a newly updated SYSGEN STAGE II DECK can be created. This new STAGE II DECK will reflect the NEW PATTERN that IBM has planned for the new version of the operating system. ACCEPT NOAPPLY processing replenishes the DLIBS directly from RECEIVED SYSMODS, bypassing the TARGET LIBRARIES entirely. After the ACCEPT NOAPPLY has been done, and the DLIBS reflect the state of the new system, a STAGE II SYSGEN DECK is assembled from the installation's STAGE I DECK, and the new STAGE II DECK is made known to SMP through the JCLIN process. When a subsequent APPLY of the same SYSMODS is done, all their ELEMENTS will fit into the system according to their proper new pattern.

We shall finish our discussion with the topics of "UCLIN" and "LIST". Then I'll let you try your hand at the SMP books. You'll probably find them clear and well-written now, because you're past the "vocabulary barrier". Any words I haven't defined here can probably be found in the glossaries of the SMP publications, and you can be on your way.

UCLIN for SMP CONTROL DATASETS roughly corresponds to ZAP for data. UCLIN provides the facility for ARBITRARILY CHANGING SMP CONTROL INFORMATION. Sometimes SMP control entries must be adjusted to allow a PTF to be APPLIED or ACCEPTED properly. For example, suppose PTF B is intended to replace ELEMENT A, but our installation has modified ELEMENT A with a USERMOD. Our intention is to replace ELEMENT A with IBM's new version of it, supplied by PTF B. After the element is replaced, we will refit our USERMOD to the new version. This is what we want, but SMP will not let us do it. That is because PTF B doesn't "know", in its "PRE" and "SUP" keywords about our USERMOD, which tagged ELEMENT A with a UMID equal to the USERMOD's SYSMOD ID. The APPLY CHECK run to APPLY PTF B will report an ID CHECK, stopping PTF B from going on.

One way to get around this situation is to use UCLIN processing to remove the strange UMID from ELEMENT A before doing the APPLY for PTF B. Then PTF B can be APPLIED without any trouble, and it will replace ELEMENT A on the system, achieving the result that we want. (Another approach is to BYPASS ID CHECKS and force PTF B on. The UCLIN approach seems cleaner to me.)

UCLIN can also be used to replace large pieces of SMP control data. SMP has a facility called UNLOAD, that converts entire SMP entries, or even an entire ZONE into UCLIN control statements. This may prove useful to some users. The point is that the UCLIN facility gives us fine control on the SMP entries, outside of the RECEIVE, APPLY, ACCEPT milieu. We must obviously be very careful with UCLIN, and we should only use it when we know exactly what we are doing. Sometimes an IBM-supplied PTF will recommend the application of UCLIN. This should also be done with care, because a previous SYSMOD may have done the proper adjustment already.

"LIST" allows the SMP user to DISPLAY ALL SMP CONTROL INFORMATION, and it is obviously the tool to use when determining if UCLIN is necessary, and if your UCLIN worked properly. LIST has much more general-purpose use in SMP. It is THE WAY TO DO SMP INQUIRY. With SMP4, it is practically the only way find out SMP control information. (There are a few other tools around, including a "LISTCDS" TSO command on file 300 of the CBT tape.) With SMP/E, there is an extensive ISPF INQUIRY SYSTEM that takes the place of the LIST function for most routine lookups. LIST can still be used for big inquiries in batch. The SMP books have much information on the use of the LIST command.

That's all for now, but I hope it's enough to achieve our purpose. Now you can hit the books, and use SMP carefully, but with confidence.