

FEPI: My Favorite Four-Letter CICS Word

Or, What I Learned While Helping the Nation's Largest Public School System Enroll Students More Efficiently

A HostBridge White Paper

By Russ Teubner, CEO



866-965-2427

info@hostbridge.com

Copyright Notice

Copyright © 2011 by HostBridge Technology. All rights reserved. No part of this publication may be reproduced, transmitted, transcribed, stored in a retrieval system, or translated into any computer language, in any form or by any means, electronic, mechanical, optical, chemical, manual, or otherwise, without prior written permission. You have limited permission to make hardcopy or other reproductions of any machine-readable documentation for your own use, provided that each such reproduction shall carry this copyright notice. No other rights under copyright are granted without prior written permission. The document is not intended for production and is furnished "as is" without warranty of any kind. All warranties on this document are hereby disclaimed including the warranties of merchantability and fitness for a particular purpose.

Revision date: 1/3/2011

Trademarks

HostBridge and the HostBridge logo are trademarked by HostBridge Technology. All other trademarks mentioned are property of their respective owners.

FEPI: My Favorite Four-Letter CICS Word

Or, What I Learned While Helping the Nation's Largest Public School System Enroll Students More Efficiently

Every now and then a customer has a “special” requirement. We love these situations: they are usually time critical and have clear business value. We like the challenges. And who knows? Perhaps we can show off a bit.

Such was the case recently when a customer asked if they could use HostBridge to integrate with non-CICS applications. The initial request went like this: “We have applications that run under Model 204. Can we use HostBridge to XML-enable them?”

Wow! Model 204! What a blast from the past. In a former life, I actually evaluated, acquired, and administered Model 204 for a project. For its time, Model 204 was very innovative. So the request didn't catch me off-guard; it did raise an eyebrow.

Most Model 204 apps were written in their own proprietary language, styled as terminal-oriented (end-user) apps, and accessed via their own communications monitor (not CICS, etc.). HostBridge, however, runs under CICS – historically, that is; watch this space for news. So the key questions regarding the request were: what are the operational characteristics of the Model 204 apps, and what are the customer's specific integration requirements? The key questions regarding our response were: could HostBridge adapt to these characteristics and satisfy these requirements? Conference calls ensued.

We learned that, true to form, the customer's apps were written in the Model 204 language, styled as terminal-oriented apps, and accessed via the Model 204 communications monitor. The customer's requirement was to interact with these apps from a .NET application via HostBridge in order to optimize a student enrollment process, i.e., allow it to proceed faster and cheaper (and with more than one million students, the customer had skin in the game).

Our answer was, “Yes, HostBridge could support access to these applications, but we would need to use our ‘generic interface’ to non-CICS terminal-oriented transactions.”

But what does *that* mean? It means I get to throw around my favorite four-letter word in the CICS lexicon: FEPI, or Front-End Programming Interface. Or as it's often spelled below – F#*%!

FEPI...A Four-Letter Word?!

I'll admit right off the bat that I've had a love/hate relationship with FEPI over the years. I love it because it's inside of CICS. And when used appropriately, it can help

meet some targeted CICS integration requirements. I've hated it at times because FEPI was sometimes used indiscriminately to create some rather "brittle" CICS integration solutions. And, inevitably, CICS integration solutions that are brittle, or perform poorly, give a black eye to the underlying CICS applications, CICS itself, or System z as a whole. Needless black eyes hurt us all.

Over the last decade, HostBridge has doctored its fair share of black eyes. We've replaced some customer-written FEPI apps – and vendor-created FEPI-based solutions – that were inflexible or performed poorly. Still, a nod to customers who exploited FEPI. Why? Because they didn't really have any alternatives! That is, until CICS TS 1.3 when the 3270 Bridge interface became formalized and available. But even then, the Bridge interface didn't do what FEPI does.

FEPI is a full-fledged 3270 terminal emulator/simulator running under the covers of CICS; it creates the appearance of an SNA Logical Unit (LU) to VTAM. The 3270 Bridge interface is just that: an *interface* to run 3270-based CICS transactions without a real 3270 terminal. The Bridge interface was game changing because it was the first time that CICS included a supported interface that allowed a program (like HostBridge) to interact with a CICS terminal-oriented application DIRECTLY – that is, NOT via FEPI. And if the target CICS application used BMS, the 3270 Bridge allows you to circumvent the entire process of 3270 data stream generation and interpretation. But as you can see, FEPI and the 3270 Bridge are very different.

To me, the introduction of the 3270 Bridge interface was the final step in creating the necessary and sufficient conditions to achieve the death of screen-scraping in many, many cases. And given the rise of XML as the *lingua franca* of the Web, Scott Glenn and I designed HostBridge and wrote the application for US Patent No 6981257: "System, method and apparatus to allow communication between CICS and non-CICS software applications."¹ Enough history for now....

Bottom Line: FEPI was revolutionary for its time. However, with the advent of CICS TS 1.3, FEPI's proper role became more limited. Using FEPI for programmatic access to CICS terminal-oriented apps (within an interconnected set of CICS regions) became at best passé, and at worst a bad idea (depending on circumstances). However, for programmatic access to non-CICS terminal-oriented apps, FEPI was still the only game in town. Additionally, since FEPI had its own EXEC CICS commands, it was easy for customers to construct FEPI-based CICS apps. However, the capabilities of the 3270 Bridge are expressed as a *programming interface*. This is fitting given its purpose and benefits, but it's not as easy to use as FEPI (that's partly why products like HostBridge exist).

3270 Bridge, FEPI, and HostBridge

HostBridge began life as a product whose sole means of interacting with CICS apps was the 3270 Bridge interface. We were undisputed early-adopters and advocates of this technology – still are. These are our roots and it's why our customers probably represent the largest users of the 3270 Bridge interface. In the early days, when CICS TS 1.3 and HostBridge were both new, I was something of a 3270 Bridge

¹ Canadian Patent No. 2,435,263, and EU Patent Pending.

“evangelist.” I found myself in countless conversations about certain topics: (a) what’s the difference between the “3270 Bridge interface” and the “3270 Web Bridge”?, and/or (b) what’s the difference between 3270 Bridge and FEPI? Only after I got past those questions could I get to the value proposition of HostBridge: integrating with CICS BMS apps without screen-scraping. And that was the important point!

Given that FEPI is a 3270 terminal emulator, it is axiomatic that one of its primary functions is to generate and process 3270 data streams. The EXEC CICS FEPI commands allow a program to interact with either the raw 3270 data stream, including presentation logic, or a formatted version of the screen image. Regardless, the implication is clear: all FEPI applications “screen-scrape.” Whatever processing or integration is being performed has a binding relationship to the underlying 3270 data stream presentation logic. Publicly, we were often “anti-FEPI” and “pro-Bridge Interface” because it seemed to be the only way to draw a clear distinction between these two interfaces and articulate the value proposition of HostBridge. We wanted HostBridge to become synonymous with high-fidelity, high-performance CICS integration. That meant no screen scraping, and that meant no FEPI.

It worked. But as the use of HostBridge grew, what did customers want to do? Use HostBridge to integrate with non-CICS apps! And what did this imply? FEPI! So as of HostBridge v4.12a (which was quite a while ago) we, the self-proclaimed FEPI antagonists, added FEPI support to our product. We describe HostBridge support for FEPI as providing access to “special” types of applications, for example:

- CICS applications that cannot be executed under a bridge facility, usually because they circumvent the CICS terminal control interface
- Non-CICS applications, such as those that run under IMS or other subsystems
- Applications provided by an external service provider, and that can be accessed as a VTAM application from within the customer’s SNA network.

With this functionality in place, we began to see customers use HostBridge to develop CICS-based integration processes that incorporated an even broader array of System z applications.

Details, Details

When you use FEPI to interact with a terminal-oriented VTAM application, it is very important that you understand how the application (or communications monitor) interacts with the terminal. Inevitably, this draws you back into the cruel reality that all VTAM applications (CICS, IMS, TSO, whatever) communicate with terminals according to SNA LU2 protocols. Remember those? They are the “laws of physics” that govern how a VTAM app communicates with a terminal. Perhaps we would like to think that SNA Bind parameters are a thing of the past. (Most of us have already recycled the brain cells that decoded phrases like “half-duplex contention,” “half-duplex flip-flop,” “bracket protocol,” and “change direction protocol.”) But lo and behold, if you are going to use FEPI to interact with a VTAM application, you better issue a refresh on those brain cells.

Actually, if the target FEPI application runs under CICS, you can be a less concerned about SNA LU2 protocol matters. That's because CICS has always been one of the most "well behaved" SNA applications on the planet. Oh, but wait... After CICS TS 1.3, and the enhanced functionality/performance of the Bridge Interface (not to mention the advent of products like HostBridge), why would you use FEPI to integrate with CICS applications? You probably wouldn't. Thus, at this point in time the easy cases are somewhat irrelevant, and we are left with the value proposition of FEPI being associated only with the hard cases.

F#*%!

Being Well Behaved

Above, I gave CICS credit for being "well behaved." What do I mean? In practical terms, there are two aspects of being "well behaved" that are relevant when discussing FEPI. Posed as questions specific to our customer's VTAM environment, they are:

1. Does the VTAM application send any "asynchronous" messages, violating a mannerly "your-turn-my-turn" view of the world and sending messages out of turn?
2. Is the VTAM application id (applid) to which you initially connect different from the ultimate VTAM applid that you end up connected to and communicate with? Stated another way, does the VTAM application pass your terminal LU to a "third party" application LU?

If the answer to the first question is "yes," you have a number of details to attend to (sometimes nasty, but not always). If the answer the second question is "yes" take a deep breath and ask your boss how much time you can spend on the project (better yet, call HostBridge).

By the above standard, CICS applications are well behaved because: (1) asynchronous messages are rare and usually limited to applications that exploit ATI (Automatic Transaction Initiation), and (2) CICS does not pass the terminal from a primary applid to a third-party applid. IMS applications are a mixed bag because: (1) they do have a penchant for sending asynchronous messages, but (2) IMS doesn't use third-party applids. The poster child for a VTAM application that is NOT well-behaved is TSO because the answer to both questions is "yes." (To be fair, TSO exploits the SNA LU2 protocols as designed, and squeezes every ounce of functionality out of them.)

So, whether you are an industrious CICS customer (you), or an enterprising CICS business partner (us), if you are going to try and use FEPI to communicate to a VTAM application, your first thought should be: Is it more like CICS, IMS, or TSO?

Model 204 Under the Microscope

At one time I supposedly knew the answer to the above two "behavioral" questions for Model 204. But my attempt at memory refresh failed, and I had to resort to

another trusty procedure. At the customer's site, my first step was to configure CICS/HostBridge/FEPI to access Model 204 and see what happens. FEPI configuration can be complicated, but it's at least well documented. It's the "see what happens" that is usually a pain because it involves performing a VTAM buffer trace (captured via GTF) while running the test scenario. If you can control all the knobs yourself, this is not too difficult. However, many large organizations have adopted a division of responsibility between system programmers (or have outsourced their data center operations and system programming), which makes this process rather inconvenient. In such cases, doing coordinated tracing can result in yet another expressive moment .

F#*%!

Given the complexities of this customer's organization, being onsite turned out to be the key to coordinating the configuration/testing/tracing process. And what did it reveal? Model 204 behaves mostly like TSO because it DOES pass control of the terminal session to a third-party VTAM applid (e.g., from "M204" to "M204001"). Thankfully, it doesn't send too many asynchronous messages past the initial logon process. Still, for our purposes, Model 204 had to be deemed NOT "well behaved." Once we saw how Model 204 operated two things were clear: (a) we were going to have to do some additional testing (and possibly development), and (b) we were going to have to figure out a way to do this work on our own system. But wait.... We certainly don't have Model 204 running our z/OS system, so the question became: What is the best proxy for Model 204? Yep, you guessed it.... TSO.

F#*%!

TSO? Gulp!

As mentioned above, HostBridge support for FEPI was driven by certain customer requirements. And it just so happens that none of those customers wanted to access TSO (thankfully). It was usually IMS or odd CICS application environments. So now we were presented with a requirement to support Model 204, but our only viable proxy was TSO. We couldn't dodge the TSO bullet any longer. While supporting TSO was overkill, we knew that if we could support TSO via FEPI, we could probably support anything. Sounded enticing.

But wait. CICS support for FEPI was also driven by certain customer requirements. Did they include TSO? I had serious doubts since the term "TSO" only appears once in the FEPI User's Guide (in the context of "Accessibility features," which doesn't count). The Guide doesn't say FEPI *can't* be used to interact with TSO, but it certainly doesn't illustrate how to do it. Time to search the web: had anyone ever used FEPI and TSO together and left behind some bread crumbs of knowledge on the Internet? If you Google "CICS FEPI TSO" you conclude quickly that the answer is "no." It looked like we are going to blaze a bit of a trail (this document is my bread crumb for posterity).

Time to Roll Up Our Sleeves

I've already alluded to the most challenging aspect of supporting access to VTAM applications like TSO or Model 204: the VTAM applid that you initially connect to is different from the ultimate VTAM applid that you end up communicating with. For example, when you logon to TSO (the initial applid), TSO accepts the session but immediately passes it to a different applid: TSOnnnn. The programming macro that VTAM applications use to achieve this transfer of control is CLSDST, with the PASS option. Thus, the most important two pages of the FEPI doc are appropriately labeled "Handling CLSDST(PASS)." Read this information carefully; commit it to memory; every word has meaning.

Based on this information I modified our standard FEPI "unexpected event" handler to support Session events (caused by the unsolicited Bind from the third-party PLU). In response, the event handler needs to dynamically define a new FEPI target with the appropriate properties and add it to the appropriate pool (don't forget the cleanup process too). I found these steps a little surprising at first; in fact, they seemed rather magical. On the other hand, at least FEPI supports unsolicited Binds!

OK. So let's try and actually connect to TSO from HostBridge via FEPI. You know the drill: Type command; squint eyes; look away and press Enter. Nothing. Do it again. Silence.

I assumed I was doing something wrong in my event handler, so I began to go through the typical debugging procedures. However, nothing really indicated a failure in my code. In fact, I was not even seeing my code scheduled for Session events. Not good. At this point you might think a CICS event trace would be a good idea. Don't bother. The trace events generated by FEPI (the SZ domain) are rather obscure. Their obscurity is matched only by their volume. To me, the only clear and conclusive way to see what is happening between FEPI, VTAM, and a target application is to do a VTAM buffer trace. (If you are really adventurous, you can do a CICS event trace – with the trace data going to GTF – at the same time you do the VTAM buffer trace. You can then print them together and marvel at how the exterior world of VTAM meshes with the interior world of FEPI.)

OK. VTAM buffer trace in hand. Now...what the heck is going on?

Where Did the Unbind Go?

The first commercial software product I wrote was called A-NET (great code, strange name). It was my *magnum opus* as far as VTAM applications were concerned. Under the covers it did many of the things FEPI does, but for a different reason. I could sling around VTAM macros pretty well; I could read VTAM buffer traces in my sleep; and I certainly knew that a CLSDST(PASS) macro caused an SNA "Unbind, Bind forthcoming" Request Unit (RU) to flow. Wait! That was 25 years ago...and the VTAM buffer trace I was looking at now didn't have any such Unbind request! On the one hand, this was perplexing; on the other, it seemed to hint at why FEPI never scheduled my Session event handler.

Somewhere during the past decade (while I had my head buried inside of CICS), IBM decided that the traditional mechanism used by VTAM to implement CLSDST(PASS) was inefficient. True enough. The process involved first Binding a session between the Secondary (terminal) LU and Primary (application) LU. Once the session was bound, and the data flow started, the initial PLU would issue a CLSDST(PASS) macro which would immediately Unbind the session, indicating that another Bind would be forthcoming. What would transpire next is another Bind process between the SLU and the new PLU that handles the actual session. Pretty clunky, but it worked well. In order to improve this process IBM implemented a mechanism called "FASTPASS." FASTPASS is clearly more efficient, but it is completely different from the traditional Bind-Unbind-Bind mechanism (the RU flow is much different).

Might the problem be that FEPI doesn't support the FASTPASS mechanism? This was a good theory. However, to test it you have to turn FASTPASS off. But wait a second...

Turning FASTPASS Off

When I first realized that something was very different about the way VTAM was handling the CLSDST(PASS), I didn't know it had anything to do with something called "FASTPASS." There's nothing in the VTAM buffer trace that says: "WARNING – SOMETHING IS HAPPENING HERE AND IT'S CALLED FASTPASS!" So the first task was to figure out what this phenomenon was called so I could learn about it.

Long story short, the use of the FASTPASS mechanism is controlled by a parameter on a VTAM application program major node definition. Per the z/OS *Communications Server SNA Resource Definition Reference*:

[The FASTPASS parameter] determines how session establishment is performed for application programs that issue the CLSDST macroinstruction with the PASS option as part of their session establishment procedure. The application program acts as a secondary logical unit (SLU).

The primary logical unit (PLU) acknowledges the capability of the SLU during session establishment. If you have coded the logon performance enhancement in the PLU's application program, only a single bind is issued to establish the session.

Some SLUs are incompatible with this enhancement. If your SLU is incompatible with this enhancement, code FASTPASS=NO.

Note: FASTPASS applies only to application programs (acting as SLUs) which establish sessions with TSO.

FASTPASS=NO Specifies that an unenhanced logon procedure requiring multiple binds is performed for application programs that issue the CLSDST PASS macroinstruction with the PASS option as part of their session establishment procedure.

FASTPASS=YES Specifies that the logon performance enhancement is performed for application programs that issue the CLSDST macroinstruction with the PASS option as part of their session establishment procedure.

If you are not acquainted with SNA-speak, some of this reads like gibberish. However, did you catch that sentence: “Some SLUs are incompatible with this enhancement”? Yeah. That would be FEPI. So here’s the bottom line on this matter: FASTPASS=YES is the default on a VTAM APPL definition, and FEPI doesn’t support FASTPASS. Thus, the application major nodes used by FEPI to connect to TSO better specify FASTPASS=NO or you are never going to get to square one. Period.

Success! Sort of...

Once I got this whole FASTPASS issue sorted out, things began to work – sort of. My event handler was indeed being invoked for a Session event. And if I went through the gyrations of dynamically adding the correct target, with the correct properties, in the correct pool – BINGO! A session would be bound between my FEPI application node and TSOnnnn. But wait.... It was also being immediately terminated....

F#*%!

This particular issue was far more difficult to diagnose than anything up to this point. FEPI was successfully Binding to the TSO applid (TSOnnnn). However, for some reason, it was Unbinding the session after TSO sent the initial Logon prompt message. There was nothing in the VTAM buffer trace that indicated why CICS/FEPI was doing the Unbind. I racked my brain and could see only a few options to figure out what was happening: (a) scour the FEPI doc and try to ferret out some shred of information that I had missed, (b) hope something was mentioned about this behavior on IBMLink, or (c) stare at the FEPI trace data until a vision of its meaning appeared. I tried each. To no avail.

All I wanted was for my FEPI connection to enter the golden state of being acquired with no conversation. Was that too much to ask?

The FEPI Guru

Years ago, when we added our initial support to HostBridge for FEPI, we had a few questions. As an IBM business partner, HostBridge had access to the CICS development group in Hursley, England. They have always been very helpful; that time was no exception. As a result of that collaboration, I became aware of a gentleman named Robert Harris on the CICS team. I never knew his proper title; I just thought of him as the “FEPI Guru.”

Fast forward to the current project. About midnight one evening, out of sheer desperation, I dug back in my email archives and found Robert’s name. I then Googled “Robert Harris.” Immediate hit! It turned out that Robert had recently retired from IBM and had a Facebook page. Email away. Within 24 hours Robert and I reconnected, and what ensued over the next week was a delightful collaboration.² Most importantly, we were able to determine the reason for the unexplained Unbind between FEPI and TSOnnnn.

² See the Appendix below for a transcript of our conversation about the history of FEPI.

Brackets Anyone?

At the beginning of this paper, I referred to the importance of understanding SNA protocols if you are really going to venture far into the land of FEPI. One of those SNA protocols pertains to “brackets.” The concept of brackets is foundational to understanding how LUs communicate in an orderly manner.

The best definition, albeit abstract, of “bracket protocol” is in the *z/OS Communications Server SNA Programming Reference*:

A bracket is any “uninterruptible” unit of work that an application program and an LU have been programmed to accomplish. A bracket can consist of any combination of data requests and data replies, ranging from a single request in one direction to an elaborate exchange of requests and replies. (Reply is used here to mean a data request sent in answer to a previously received data request.) But, no matter how simple or complex the series of requests and replies can be, the characteristic that makes them all part of the same bracket is that they all pertain to the same unit of work.

This is a conceptual definition, and words such as “uninterruptible” or “unit of work” can have different meanings to different applications. To one VTAM application, a unit of work could include all interactions during the life of the session. To another VTAM application, a unit of work could be a related set of interactions (one of many) during the life of the session. Such is the case with TSO and CICS, respectively.

TSO has a strong bias toward “half-duplex flip-flop” protocol. Essentially, this means that TSO starts a bracket at the beginning of the session and encapsulates all terminal interactions within a single bracket. Direction of the conversation is controlled using Change Direction protocol. CICS has a different view of the world, and a strong bias toward “half-duplex contention” protocol. Let’s just say that CICS likes to try and group related terminal interactions together within a bracket. And when a transaction ends, CICS ends the bracket. The session is still active; it’s just “between brackets” rather than “in bracket.” The rules for who can begin the next bracket (and how) are controlled by other rules. This is not to say that a 1:1 relationship exists between a bracket and a CICS unit of work, but you can see how CICS’s transactional orientation influences things.

Brackets and Conversations

Everything I just said about CICS and bracket protocols applies to its interaction with terminals. But what does it imply about FEPI? Nothing, necessarily, but it serves as a good introduction to how FEPI thinks.

When a connection has been established between FEPI and another VTAM application, the connection is “in session” and it can be “allocated” for a “conversation” (those are all key concepts to FEPI). Conversations are the basis of all FEPI applications and, depending on the needs of the application, may be used in several ways. For our purposes, remember this: FEPI has a very strong opinion about how conversation states and bracket states should mesh.

Specifically, when a FEPI session is In-Bracket, you can PASS a conversation but you can't HOLD it. If you try to HOLD the conversation (via a FEPI FREE HOLD command), FEPI will immediately Unbind the session. That is, the FREE HOLD is treated as a FREE RELEASE and your session will be toast. To FEPI, holding a conversation that is In-Bracket is a protocol error worthy of session termination. Now, since TSO begins a bracket with its first message (the userid prompt) and does not end the bracket until the user logs off, that means that you can never perform a FREE HOLD on a TSO session. Or to put it another way, once you begin a FEPI conversation on a TSO session, your only option is to PASS the conversation from one transaction to another.

I won't try to connect all the dots, but this operation characteristic of FEPI has a number of implications regarding the initial state of the FEPI-to-TSO connections, the use of a "begin session" handler, and the overall scheme for allocating and passing conversations.

Success...Finally!

The designers of FEPI worked hard to mask many of the nasty details of SNA LU-LU sessions from users of the API. If you are using FEPI to interact with a well-behaved VTAM application like CICS, ignorance is your friend. However, in a post CICS TS world, there may be better ways to programmatically integrate with CICS terminal-oriented transactions. Thus, the well behaved cases are often irrelevant, and understanding how FEPI maps its actions/concepts to the underlying SNA protocols can be very important. In fact, there are certain things about FEPI that will never make sense unless you do. Hopefully, this article will add the accumulated public wisdom and lore about FEPI.

I doubt my successful outcome will kickstart a modern day FEPI revival – or entirely eliminate the four-letter F#*%!

But I can at least report that customers are now able to use HostBridge to XML-enable, Web-enable, or service-enable TSO, Model 204, or any other "badly behaved" VTAM application.

Is that the lid of Pandora's Box is hear opening?

Appendix: A Chat with FEPI Guru Robert Harris

As mentioned above, I recently had the good fortune of reconnecting with Robert Harris, one of the IBMers responsible for the design and development of FEPI.

If you know a little about FEPI, it's easy to feel confused. As you learn more, you start to get the picture that some people worked pretty darned hard to design it and bring it to life, especially given the limitations of CICS during that era.

Our recent work with FEPI made me curious about its history. With that in mind, I chatted with Robert hoping to fill in the gaps about FEPI's roots.

Russ: *What was the first version of CICS that included FEPI?*

Robert: FEPI first came out in CICS/ESA 3.3. This was the first thing that used the newly restructured CICS kernel with the ability to use non-QR TCBs. So the SZ TCB was the first one that was introduced for “foreign” usage within CICS. I called it the SZ TCB (although I wanted to call it the Remote Handler TCB!) because it was simulating VTAM. Since CICS’s terminal control modules used TC for BTAM and ZC for VTAM, SZ made sense.

FEPI was originally going to be called Terminal Simulation Facility – but that failed due to some naming issues. Hence, Front-End Programming Interface. (I still remember spending a very tedious two weeks renaming all the FEPI modules to SZ and correcting the comments.)

Russ: *Sounds like you guys were the pioneers of the whole “open TCB” concept. What drove the development of the FEPI API (clearly, it’s a bit unique)?*

Robert: Because the restructured kernel was so new, and we wanted FEPI to service many more things than the CICS Dispatcher could cope with, we decided upon an architecture that isolated QR from SZ activity as much as possible. So we evolved into a request/scheduling mechanism with XCF (EXEC CICS FEPI) activity turning into a request block, which then got queued to the SZ TCB for processing. This had an implication on the FEPI application coding style which I think was beneficial.

The SZ TCB was needed anyway because all the VTAM inbound activity on the SLU ACB/RPLs was asynchronous driven by VTAM exits. You can see in the SZ trace this effect – lots of requests suddenly appearing due to VTAM exits pinging away for RECEIVE(ANY) function. The QR (or CO) TCB would not have been able to cope with all that interference whilst doing useful work.

Russ: *So that explains all the events that are generated when tracing is active for the SZ domain. Why are FEPI resources defined differently from other “normal” CICS resources?*

Robert: Because FEPI was seen as “alien” to CICS (after all, it had its own TCB), I was not able to persuade anyone that FEPI resources should be RDOed. This is what gave rise to the dynamic resource management scheme. In the long run, this was a good thing because support the CLSDST(PASS) operation required some sort of dynamic action/response. It was also good because the ordering of TARGETs/NODEs into a pool gave different (and needed) operational properties which at that time would have been difficult to get into SZ domain initialization (this would be easier now as the Java domains required some of this function).

We decided on TDQs for things because that removed the need for the SZ TCB to schedule operations on the QR TCB which would affect existing operation. Still, usage of the TDQs required QR – but that was a quicker and more isolated operation than doing transaction starts. Tracing for the SZ TCB required all the usual region-wide locks. But that was business as usual, so no problem.

Russ: *Over what CICS versions (or time) did FEPI primarily evolve?*

Robert: Nothing really evolved with FEPI. Its LU2 and SLUP support was all there from the start. The only thing that was added was the ability to generate a PASSTICKET for a Connection so that automatic password signon could be accomplished. What we wrote for CICS/ESA 3.3 is still pretty much in place.

Because FEPI was not really a part of CICS, the doc was never up to the standard I wanted. So I wrote my book *The CICS Programmer's Guide to FEPI* which was published in 1994. It was my attempt at rectifying things. It was a bit of a struggle within Hursley to get it approved – but eventually it contained all the usability and hints/tips that I wanted, but failed, to put in the official pubs.

Russ: *What was the original vision or need that FEPI was meant to meet?*

Robert: I've described how FEPI was seen as an intruder into the CICS nucleus. That implied that FEPI was not really developed as part of the CICS organization. Well, 'tis true. FEPI was funded by MQ, and was actually developed alongside the first release of MQ (our offices were in an un-waterproofed temporary building in a parking lot at Hursley).

The idea was that MQ would provide asynchronous messaging (with that famously small API) – but that would not be interesting unless the messages actually got somewhere useful. That meant reusing existing applications, which in the 1990s meant mainly CICS (and a bit of IMS). It's taken from then to now for CICS development to take over the CICS/MQ Adapter and put MQCONN definitions into RDO!

Russ: *So MQ was the driving force behind FEPI. And it was all about integration.*

Robert: Yes, some way of getting into CICS applications, via MQ, was needed – and those applications were 3270-based. Thus, the need for a 3270 terminal emulator within CICS.

The idea was this: MQ would send a request into CICS (one of the reasons that TRUeS were developed); this would start a transaction which would screen-scrape an existing application; the results would be sent back via an MQ Response message. Thus the need for XCF FREE HOLDS (to keep the virtual terminal needed for the screen scraping) and the XCF FREE PASS (to transfer the virtual terminal to another part of the MQ interaction).

Russ: *OK... it's all starting to make sense. Anything else?*

Robert: There was also another sneaky use of FEPI. Under MRO you could not route anything back into a CICS region that was involved with the routing. So one cannot use DPL to go in a long loop and end up back in the region you actually issued the XCLINK PROGRAM() SYSID() command. FEPI did not have that restriction, and so some clever sysprogs used FEPI to remove this restriction (which is still around).

The SLUP emulation (a particular subset of LU0) was the one that IMS/DC used – and so that gave MQ the ability to get into IMS applications as well. LU0, being very basic, exposed all those cunning VTAM flags and protocol interactions.

The MQ requirement was only for XCF FORMATTed operation – but the SLUP operation required XCF DATASTREAM. Thus, we coded up the datastream layer first, and put the formatted presentation layer on top. All those VTAM indicators naturally arose from the datastream operation.

Russ: *How big was the original team, and who was on it?*

Robert: There were only four people who did the development work (plus a few contractors to do the testing). I mainly looked after the CICS side and stayed with CICS development for another 15 years or so until I left due to “early retirement” (since then, I have transformed myself into “dfhexpert”). I had the VTAM and CICS experience after doing a lot of work on terminal control and LU6.2 in CICS/XA 2.1 (added the STATE parameter on the Distributed Transaction Programming APIs, rewrote most of the then Intercommunications documentation, and caused the split into the Reference and Guide books).

Alan Webb majored on the VTAM side after joining IBM as an operator, turning into a CICS developer via SNA development, and afterwards doing a lot of the early Java work at Hurlsey (Alan then went to the USA as part of IBM Research and is now involved in grid computation).

Keith Andrews drew up the internal request protocol between an XCF command and the SZ TCB (Keith then went to MQ Development). There was another person who was lent from IMS who gave us input on LU0 programming.

Russ: *During your tenure on the CICS team, what other roles did you play?*

Robert: Well, first and foremost, I usually looked after FEPI even though I had moved on to other areas. Whilst I was in the CICS strategy and marketing team, I occasionally looked at the code while helping customers out with FEPI design and coding issues. Then I had a bit of a stay in CICS test and had fun coding up some regression tests for FEPI in PL/I. After that, I got involved with IBM Beijing Labs and mentored a load of bright Chinese doing CICS testing on what turned out to be the CPIC (I still persist in calling this MRO/IP) extensions for function shipping in all its variants. I also started learning Mandarin as a result. The very last thing I did at Hursley before my early retirement was to do a bit of design work on a project involving FEPI and CTG for a European customer. So I guess you can say that FEPI is alive and well, and still being used for new applications.

Russ: *Given all the new communication components that have been added to CICS TS over the last decade, what do you think the role of FEPI is now?*

Robert: Notwithstanding the fact that CICS has a full Web services/XML stack, FEPI is still very unique. It’s a full-function 3270 terminal emulator running inside of CICS. That’s what differentiates FEPI from the 3270 Bridge interface. Sometimes the 3270 Bridge interface is the best approach. But sometimes FEPI is the only approach. The thing I like about your product, HostBridge, is that it normalizes the underlying technical differences between FEPI and the Bridge interface and allows customers to focus on what they really care about: transforming existing 3270 applications into business services that create new business value today.