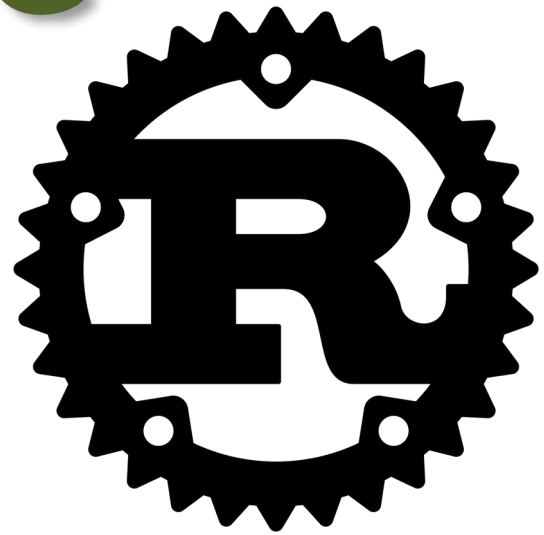
Getting a lil' Rust on the Mainframe

Clayton Slaughter, IBM
clay.slaughter@ibm.com



Agenda

There be dragons here!

- Who am I and what I do
- Problem statement
- Breaking down SMAPI
- Rust Lang
- Solution (in progress)
- Feedback (I hope)
- My *favorite* SMAPI Functions (Time Permitting)

Who am I?

Clayton Slaughter

IBMer, z/VM Imagineer

Things I'm good at:

- z/VM
- Linux
- Rust & REXX
- Video Games

Things I'm bad at:

- Seeing color
- Writing pretty presentations

What do I do?

I work for the team formerly known as the “Dallas ISV Center”.

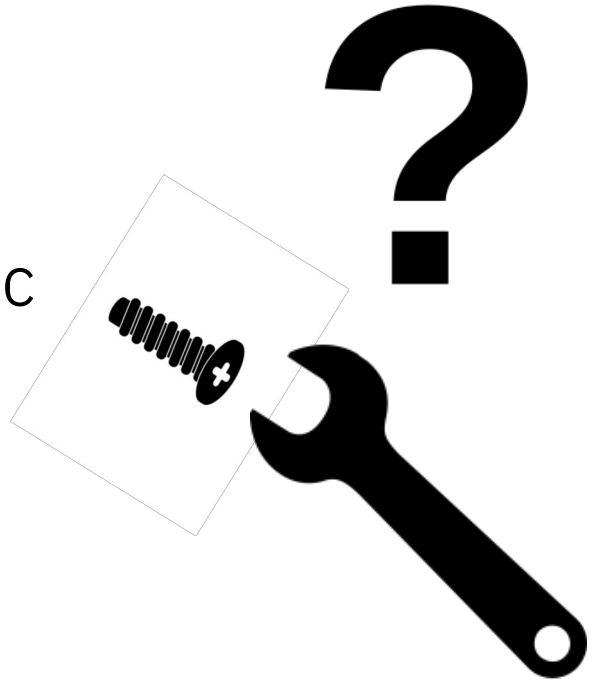
We provide z/VM guests to ISVs (Independent Software Vendors) for them to develop and test software on.

As such, we leverage z/VM features to do things like:

- provide segmented network access to ISVs using VLANs
- run several operating systems such as z/VM, z/OS, and Linux,
- provide access and maintenance to both the z/OS base and products using various minidisk links and RACF rules.
- sort tenants into RACF groups for easier management and separation of access.

The Problem

- The ISV Center runs a lot of z/OS systems, roughly 200-300.
- Today those are built by hand when someone requests one. But as business grows, that's not scalable.
- Very few tools support running z/OS on z/VM. IBM's ICIC only supports Linux, for example.
- Some ISV products do support z/OS, but due to the nature of our business we can't use them.
- We also have a few z/VM and "other" systems we run.
- So what can I do?



SMAPI

System Management Application Programming Interface

The Good:

- Official IBM API for working with z/VM
- Extensible, so you can add any functionality you need
- Available over IUCV or TCPIP and supports TLS on TCPIP
- The functions are all written in uncompiled REXX if you really want to take a peek

The Bad:

- The API is pretty old and requires careful byte ordering instead of more modern API like REST, SOAP, or gRCP
- Kinda slow, even when using LOHCOST
- Not all of DIRMAINT's functions are covered (e.g. setting VLANs)
- Somewhat brittle (or so I've heard)

The Ugly:

- Documentation sometimes leaves out important details
- API formatting is highly inconsistent

SMAPI

What does a SMAPI function look like?

```
                                [0, 0, 0, 74]
                                [0, 0, 0, 26]
[81, 117, 101, 114, 121, 95, 65, 80, 73, 95,
   70, 117, 110, 99, 116, 105, 111, 110, 97,
            108, 95, 76, 101, 118, 101, 108]
                                [0, 0, 0, 7]
            [67, 108, 97, 121, 116, 111, 110]
                                [0, 0, 0, 18]
[77, 121, 83, 101, 99, 114, 101, 116, 80, 97,
       115, 115, 80, 104, 114, 97, 115, 101]
                                [0, 0, 0, 7]
            [67, 108, 97, 121, 116, 111, 110]

                             [36, 219, 10, 145]

                                [0, 0, 0, 12]
                             [36, 219, 10, 145]
                                [0, 0, 0, 0]
                                [0, 0, 2, 14]
```

```
                    74
                    26
Query_API_Functional_Level
                     7
                 Clayton
                    18
          MySecretPassPhrase
                     7
                 Clayton

          Random number

                    12
          Random number
                     0
                   740
```

Query_API_Functional_Level

Input Parameters:

- *input_length*
- *function_name_length*
- *function_name*
- *authenticated_userid_length*
- *authenticated_userid*
- *password_length*
- *password*
- *target_identifier_length*
- *target_identifier*

Response 1 – Immediate Request Verification:

- *request_id*

Response 2 – Output Parameters:

- *output_length*
- *request_id*
- *return_code*
- *reason_code*

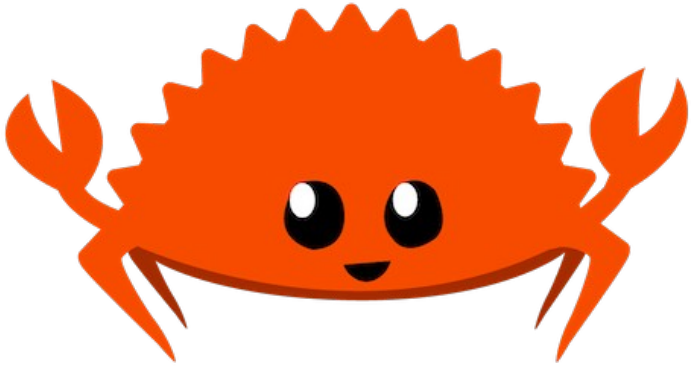
Tools that use SMAPI

smcli – Command line utility for interfacing SMAPI
- Maintained by the OMP
- Requires IUCV connection to z/VM
- Not extensible

Feilong – Open source project to provide REST APIs to z/VM and as a framework to build tools
- Relies on **smcli** and therefor inherits many of it's limitations
- Heavily focused on running Linux workload, no real support for other OSes

ICIC – IBM Cloud Infrastructure Center
- Relies on **Feilong** to communicate with z/VM
- Further limits the functionality exposed by **Feilong** and **smcli**

The Rust Language



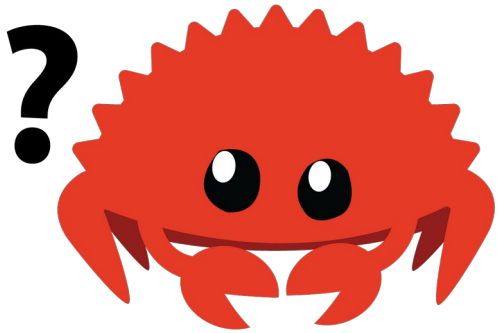
- Systems programming language like C, C++, Go, and Zig
- A focus on memory safety, performance, and “correctness”
- Strongly typed, sometimes to a fault
- Not strictly object oriented (no polymorphism, classes, etc) but shares a lot of concepts
- Backwards compatibility goals
- Built in build system and tools with **cargo**, **clippy**, **rustfmt**, and **rust-analyzer**
- Interoperability with other languages using C as a common API
- Large ecosystem of packages
- Suitable for many tasks, such as writing a CLI, GUI, website, or system automation
- Powerful macro system for meta-programming

The Rust Language

Correctness? What does that mean?

- Data is modeled using Rust's expressive type system
- When code could return different results, the Enum type is used with all possible outcomes
- The programmer then needs to handle each possible outcome before the code will even compile

```
// This function returns a "Result"
fn could_error() → Result<GoodData, Error> {
    …
}
```

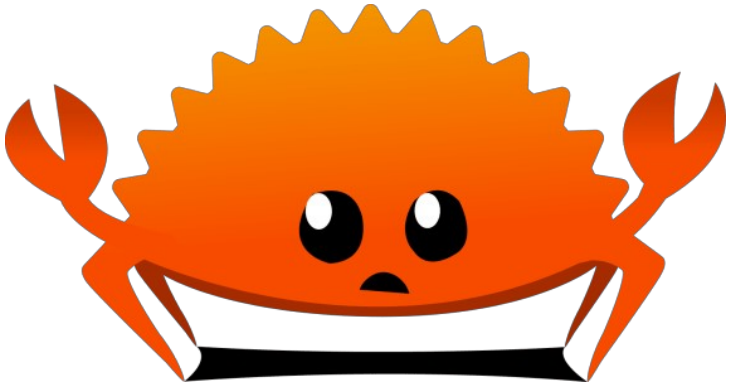



```
// The result has to be handled in code
let data = could_error();
match data {
    Ok(good_data) => {
        … do something with the data
    },
    Err(e) => {
        … do something with the error instead
    }
}
```

The Rust Language

There are *some* downsides

- Not as “mature” or “established” as a language like C, COBOL, or Java.
- Currently no stable ABI for dynamic Rust libraries (and may never be)
- Steep learning curve, especially for features like async, macros, and lifetimes
- “Rewrite it in Rust” commonly ruffles feathers, especially when there’s no need
- Community of Rustaceans can be a tad dramatic at times
- No compiler support for z/VM



When to Reinvent the Wheel

- Current solutions do not work for us
- It's better to leverage existing technology (SMAPI) than to start something completely homegrown
- If I make something reusable and share it, I could get help in supporting it

Goals

- 100% coverage of SMAPI's existing functionality.
  - Connect over IUCV, or TCPIP optionally with TLS
  - Implement all functions and make sure all of their options work
- Make the SMAPI calls easier to consume as a user
  - Auto generate derived value like the lengths of fields
  - No need to specify optional fields
  - Try to add guardrails to prevent incorrect usage
- Easy to implement custom SMAPI functions
- Allow for use both synchronously and asynchronously

My Solution

So far...

- Map SMAPI functions to Rust data types
- Automatically derive the functions for building the types, converting to a stream of bytes, decoding the response, etc.
- Have an object representing a connection that can send the request and receive the response
- Provide this as a crate (a Rust library) so that others could potentially benefit

My Solution

So what does ***this*** look like?

Query_API_Functional_Level

```
Int4   - input_length
Int4   - function_name_length
String - function_name
Int4   - authenticated_userid_length
String - authenticated_userid
Int4   - password_length
String - password
Int4   - target_identifier_length
String - target_identifier
```



```
Pub struct Query_API_Functional_Level {
    input_len: u32,
    fn_name_len: u32,
    fn_name: String,
    userid_len: u32,
    userid: String,
    password_len: u32,
    password: String,
    target_len: u32,
    target: String
}
```

Not a bad start, but I think we can do better!

My Solution

That sure is a lot of “len”s…

- Every single function has the total length encoded first without exception. We can cut it.
- Every single function has the function name length encoded, we can cut it too.
- Strings *usually* encode their length, but there are exceptions. So how do we know when to add the length and when not to?
- And the function name is included twice, both as the name of the struct and as a field. Can’t we do something about that?

```
pub struct Query_API_Functional_Level {
    input_len: u32,
    fn_name_len: u32,
    fn_name: String,
    userid_len: u32,
    userid: String,
    password_len: u32,
    password: String,
    target_len: u32,
    target: String
}
```

My Solution

Shared functionality with Traits

- In order to convert this to a stream of bytes that SMAPI can use, we need to write a method like the following.
- We convert each of the fields into bytes first.
- We create a temporary buffer and store each fields length as bytes, and then the actual bytes.
- Finally, we use a second buffer so we can prefix the total length of the request.
- By using Rust Traits, we can treat all of the functions as interchangeable “Trait Objects”, allowing for sharing code paths for sending.
- But having to hand code all of these is still pretty bad.

```
impl SmapiEncode for Query_API_Functional_Level {
    fn to_smapi_bytes(&self) → Vec<u8> {
        let fn_bytes = “Query_API_Functional_Level”
            .to_smapi_bytes();
        let user_bytes = self.userid.to_smapi_bytes();
        let pass_bytes = self.password.to_smapi_bytes();
        let target_bytes = self.target.to_smapi_bytes();
        let mut temp = Vec::new();
        temp.extend(fn_bytes.len().to_smapi_bytes());
        temp.extend(fn_bytes);
        temp.extend(user_bytes.len().to_smapi_bytes());
        temp.extend(user_bytes);
        temp.extend(pass_bytes.len().to_smapi_bytes());
        temp.extend(pass_bytes);
        temp.extend(target_bytes.len().to_smapi_bytes());
        temp.extend(target_bytes);
        let mut output = Vec::new();
        output.extend(temp.len().to_smapi_bytes());
        output.extend(temp);
        output
    }
}
```

My Solution

Macros are Magical

- Rust has macros that can be sorted broadly into two types. Declarative macros and “function-like” macros.
- Both are a form of meta-programming; code that runs at compile time to transform your existing code.
- In this example, “derive(SmapiReq)” is a macro that automatically creates the code for our SmapiEncode Trait (in addition to some other functionality).
- The “size(Prefixed)” bits are called attributes, and are used by the macro to know when to put a length field in the final outcome.
- The function name can be determined by the struct name.
- We can also define the response in a similar way.
- Finally, by exposing these macros, it’s incredibly easy for a user to define their own custom functions.

```
#[derive(SmapiReq)]
pub struct Query_API_Functional_Level_Req {
    #[size(Prefixed)]
    userid: String,
    #[size(Prefixed)]
    password: String,
    #[size(Prefixed)]
    target: String
}

#[derive(SmapiResp)
pub struct Query_API_Functional_Level_Resp {
    request_id: u32,
    return_code: u32,
    reason_code: u32,
}
```

My Solution

Other Data Types

Other data types like arrays and structures can be cleanly mapped to Rust data types as well.

Image_Query_DM

Response 2 – Output Parameters:

- *output_length*
- *request_id*
- *return_code*
- *reason_code*
- *image_record_array_length*
- *image_record_array* (1)
  - *image_record_structure* (2)
    - *image_record_length*
    - *image_record*

```
#[derive(SmapiResp)]
pub struct Image_Query_DM_Resp {
    request_id: u32,
    return_code: u32,
    reason_code: u32,
    #[size(Prefixed)]
    image_records: Vec<ImageRecord>,
}

#[derive(SmapiPart)]
pub struct ImageRecord {
    #[size(Prefixed)]
    record: String,
}
```

My Solution

Other Data Types

Data types that only have a few valid options can also be encoded as Enums, to provide a kind of guard rail (the user can only select valid input).

Image_Delete_DM

Input Parameters:

input_length
function_name_length
function_name
authenticated_userid_length
authenticated_userid
password_length
password
target_identifier_length
target_identifier
data_security_erase

data_security_erase
(int1) Indicates whether to erase data from the disk(s) being released, as follows:

0
Unspecified (use installation default)

1
Do not erase (override installation default)

2
Erase (override installation default)

```
#[derive(SmapiResp)]
pub struct Image_Delete_DM_Req {
    #[size(Prefixed)]
    userid: String,
    #[size(Prefixed)]
    password: String,
    #[size(Prefixed)]
    target: String,
    secure_erase: SecureErase,
}

pub enum SecureErase {
    Unspecified,
    NoErase,
    Erase,
}
```

Using the code

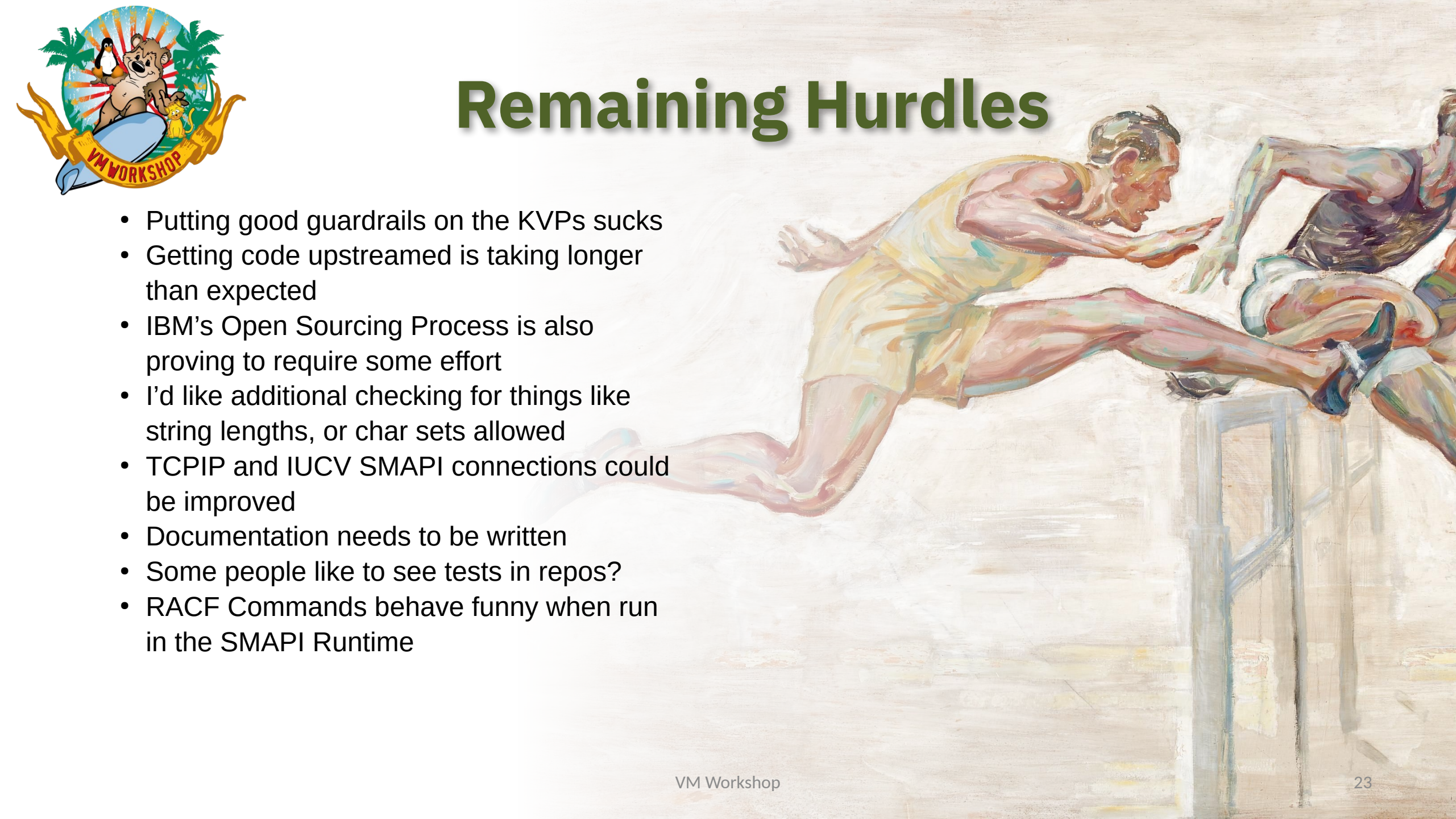
What does using it look like?

```
use ibm_z_tools::zvm::smapi::*;

fn main() {
    // Create a connection to SMAPI, this could also be over IUCV.
    let smcon = BlockingSmapiClient::new_tcpip_secure("myzvmsystem.com", 44445);

    // Create a Smapi request for the function Image_Query_DM.
    let req = Image_Query_DM_Req::builder() // Start with a builder struct
        .userid("Clayton".into())  // Set the username
        .password("MyPass".into()) // Set the password
        .target("Clayton".into())  // Set the target
        .build()                   // Build (and validate) the struct
        .unwrap();                 // Validation can fail, we'd need to handle any errors here

    let resp = smcon.send(req)     // Send the request through the client we created
        .unwrap();                 // Sending could fail, so we'd need to handle errors here

    println!("{:#?}", resp);
}
```

Live Demo?

How can this possibly go wrong?

Remaining Hurdles



- Putting good guardrails on the KVPs sucks
- Getting code upstreamed is taking longer than expected
- IBM's Open Sourcing Process is also proving to require some effort
- I'd like additional checking for things like string lengths, or char sets allowed
- TCPIP and IUCV SMAPI connections could be improved
- Documentation needs to be written
- Some people like to see tests in repos?
- RACF Commands behave funny when run in the SMAPI Runtime

Back to Goals

How are we looking on those?

- 100% coverage of SMAPI's existing functionality
  - Connect over IUCV, or TCPIP optionally with TLS – Could be improved, but works
  - Implement all functions and make sure all of their options work – Getting there, still work to do
- Make the SMAPI calls easier to consume as a user
  - Auto generate derived value like the lengths of fields – Done with the macro
  - No need to specify optional fields – Done using the builder pattern
  - Try to add guardrails to prevent incorrect usage – Could be improved but partially done
- Easy to implement custom SMAPI functions – Macro allows user to easily create custom functions
- Allow for use both synchronously and asynchronously – Basic Async support is there

Feedback Time

Questions?

Comments?

Could you see yourself or your organization using this tool once it's available?

Are there any particular features you'd like to see?

Want to email me about it? clay.slaughter@ibm.com

Clay's Collection of Gnarly SMAPI Functions!

Only has output if there's an error, otherwise just has the initial bytes.

Event_Subscribe

Input Parameters:

input_length
function_name_length
function_name
authenticated_userid_length
authenticated_userid
password_length
password
target_identifier_length
target_identifier
match_key_length
match_key

Response 1 – Immediate Request Verification:

request_id

Response 2 – Output Parameters:

output_length (error only)
request_id (error only)
return_code (error only)
reason_code (error only)

See Usage Note "1" on page 112.

Clay's Collection of Gnarly SMAPI Functions!

Has no return other than the initial bytes.

It still has return and reason codes listed so I guess it's very close to Event_Subscribe but you can't tell from the doc.

Event_Unsubscribe

Input Parameters:

input_length
function_name_length
function_name
authenticated_userid_length
authenticated_userid
password_length
password
target_identifier_length
target_identifier

Response 1 – Immediate Request Verification:

request_id

Response 2 – Output Parameters:

None. See Usage Note "1" on page 116.

Clay's Collection of Gnarly SMAPI Functions!

Image_Volume_Add, in case it's too small to read.

In addition to just being a massive function, about half of the fields are now deprecated. They still have to be entered, so it's on my list to make a "Deprecated" option in the macro to auto enter a default value.

Several other functions have deprecated fields.

Image_Volume_Add

Input Parameters:

- *input_length*
- *function_name_length*
- *function_name*
- *authenticated_userid_length*
- *authenticated_userid*
- *password_length*
- *password*
- *target_identifier_length*
- *target_identifier*
- *image_device_number_length*
- *image_device_number*
- *image_vol_id_length*
- *image_vol_id*
- *system_config_name_length*
- *system_config_name*
- *system_config_type_length*
- *system_config_type*
- *parm_disk_owner_length*
- *parm_disk_owner*
- *parm_disk_number_length*
- *parm_disk_number*
- *parm_disk_password_length*
- *parm_disk_password*
- *alt_system_config_name_length*
- *alt_system_config_name*
- *alt_system_config_type_length*
- *alt_system_config_type*
- *alt_parm_disk_owner_length*
- *alt_parm_disk_owner*
- *alt_parm_disk_number_length*
- *alt_parm_disk_number*
- *alt_parm_disk_password_length*
- *alt_parm_disk_password*

Response 1 – Immediate Request Verification:

- *request_id*

Response 2 – Output Parameters:

- *output_length*
- *request_id*
- *return_code*
- *reason_code*

Clay's Collection of Gnarly SMAPI Functions!

Most of the time, when using key value pairs, you don't need to specify a length, since the end can be found by looking for 0x00 bytes. But some functions still require the prefix, so I couldn't set a default, and have to go function by function.

Image_Volume_Space_Define_Extended_DM

Input Parameters:

- *input_length*
- *function_name_length*
- *function_name*
- *authenticated_userid_length*
- *authenticated_userid*
- *password_length*
- *password*
- *target_identifier_length*
- *target_identifier*
- *image_volume_space_define_names_length*
- `function_type=`*value*
- `region_name=`*value*
- `image_vol_id=`*value*
- `start_cylinder=`*value*
- `size=`*value*
- `group_name=`*value*
- `device_type=`*value*
- `alloc_method=`*value*

Response 1 – Immediate Request Verification:

- *request_id*

Response 2 – Output Parameters:

- *output_length*
- *request_id*
- *return_code*
- *reason_code*

Clay's Collection of Gnarly SMAPI Functions!

Any of the KVP functions that use numbers to define meaning (I'd like to use enums but the KVP part makes this hard), and any of the KVP functions that vary which are valid based upon other data. Both of these make it really hard to have sane guardrails that don't take more work to use than they save by helping.

`function_type=`*value*

(string,1,char10) One of the following numeric values, determining which additional parameters are required:

1

Define region as specified. Additional parameters required for this function:

- `image_vol_id=`*value*
- `region_name=`*value*
- `start_cylinder=`*value*
- `size=`*value*

2

Define region as specified and add to group. Additional parameters required for this function:

- `image_vol_id=`*value*
- `region_name=`*value*
- `start_cylinder=`*value*
- `size=`*value*
- `group_name=`*value*

3

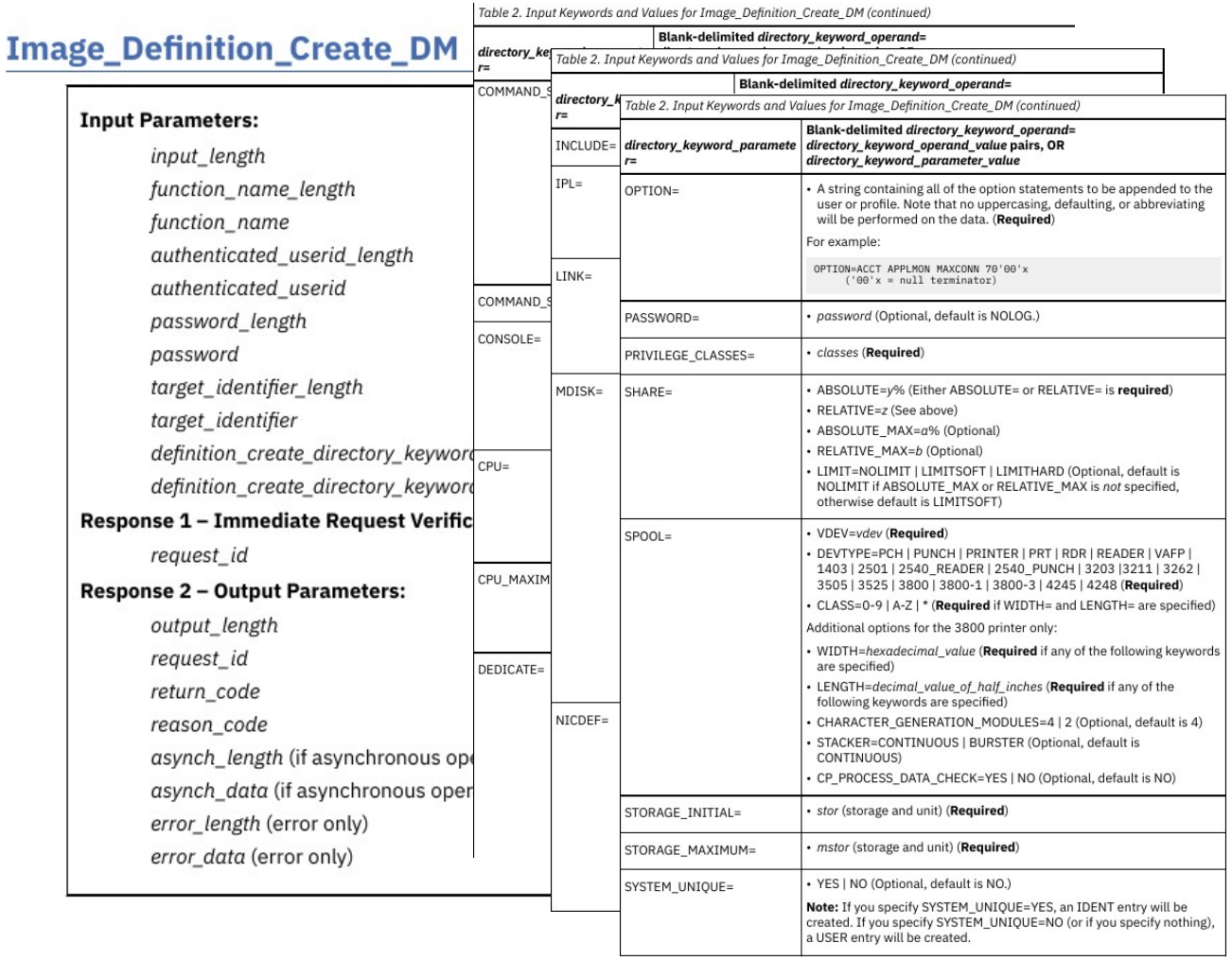
Define region as full volume. Additional parameters required for this function:

- `image_vol_id=`*value*
- `region_name=`*value*

Clay's Collection of Gnarly SMAPI Functions!

And finally this particular function. I think I've spent as much if not more time on trying to make this one easy to use, as the rest of the SMAPI inventory. For now, I'm letting users freehand the KVPs, but eventually I'll add easy ways to define valid options.

The reason it's gnarly are the total number of KVPs.



Image_Definition_Create_DM

Input Parameters:

input_length
function_name_length
function_name
authenticated_userid_length
authenticated_userid
password_length
password
target_identifier_length
target_identifier
definition_create_directory_keywor
definition_create_directory_keywor

Response 1 – Immediate Request Verific

request_id

Response 2 – Output Parameters:

output_length
request_id
return_code
reason_code
asynch_length (if asynchronous op
asynch_data (if asynchronous oper
error_length (error only)
error_data (error only)

Table 2. Input Keywords and Values for Image_Definition_Create_DM (continued)

directory_keyword_parameter=	Blank-delimited *directory_keyword_operand=* *directory_keyword_operand_value* pairs, OR *directory_keyword_parameter_value*
COMMAND_S	
COMMAND_S	
CONSOLE=	
CPU=	
CPU_MAXIM	
DEDICATE=	
INCLUDE=	
IPL=	
LINK=	
MDISK=	
NICDEF=	

Table 2. Input Keywords and Values for Image_Definition_Create_DM (continued)

directory_keyword_parameter=	Blank-delimited *directory_keyword_operand=* *directory_keyword_operand_value* pairs, OR *directory_keyword_parameter_value*
OPTION=	• A string containing all of the option statements to be appended to the user or profile. Note that no uppercasing, defaulting, or abbreviating will be performed on the data. (**Required**) For example: `OPTION=ACCT APPLMON MAXCONN 70'00'x ('00'x = null terminator)`
PASSWORD=	• *password* (Optional, default is NOLOG.)
PRIVILEGE_CLASSES=	• *classes* (**Required**)
SHARE=	• ABSOLUTE=*y*% (Either ABSOLUTE= or RELATIVE= is **required**) • RELATIVE=*z* (See above) • ABSOLUTE_MAX=*a*% (Optional) • RELATIVE_MAX=*b* (Optional) • LIMIT=NOLIMIT | LIMITSOFT | LIMITHARD (Optional, default is NOLIMIT if ABSOLUTE_MAX or RELATIVE_MAX is *not* specified, otherwise default is LIMITSOFT)
SPOOL=	• VDEV=*vdev* (**Required**) • DEVTYPE=PCH | PUNCH | PRINTER | PRT | RDR | READER | VAFP | 1403 | 2501 | 2540_READER | 2540_PUNCH | 3203 |3211 | 3262 | 3505 | 3525 | 3800 | 3800-1 | 3800-3 | 4245 | 4248 (**Required**) • CLASS=0-9 | A-Z | * (**Required** if WIDTH= and LENGTH= are specified) Additional options for the 3800 printer only: • WIDTH=*hexadecimal_value* (**Required** if any of the following keywords are specified) • LENGTH=*decimal_value_of_half_inches* (**Required** if any of the following keywords are specified) • CHARACTER_GENERATION_MODULES=4 | 2 (Optional, default is 4) • STACKER=CONTINUOUS | BURSTER (Optional, default is CONTINUOUS) • CP_PROCESS_DATA_CHECK=YES | NO (Optional, default is NO)
STORAGE_INITIAL=	• *stor* (storage and unit) (**Required**)
STORAGE_MAXIMUM=	• *mstor* (storage and unit) (**Required**)
SYSTEM_UNIQUE=	• YES | NO (Optional, default is NO.) **Note:** If you specify SYSTEM_UNIQUE=YES, an IDENT entry will be created. If you specify SYSTEM_UNIQUE=NO (or if you specify nothing), a USER entry will be created.

Thank you