# 138.https://stackoverflow.com/questions/72709581/need-help-on-building-a-function-that-creates-a-new-edition-nft-from-a-master-ed

**T:**Need help on building a function that creates a new edition nft from a master edition nft

**Q:**[image link to my code]  
  
https://i.stack.imgur.com/UWtpj.png  
  
I am currently writing a Solana Metaplex NFT program in Rust/Anchor-Solana, specifically working on writing a logic to create a edition NFT from a master edition NFT.  
  
While invoking mpl\_token\_metadata::instruction::mint\_new\_edition\_from\_master\_edition\_via\_token instruction, I found out that it requires both metadata and metadata\_mint field as arguments. Whereas, on the metaplex documentation, it seems the instruction only requires the Master record metadata account.  
  
WARN: THIS PARAGRAPH CONTAINS TAG: [CODE]   
  
Question:  
  
Which account key, or value, should I put in to each of those fields(metadata and metadata\_mint) and why?  
  
WARN: THIS PARAGRAPH CONTAINS TAG: [CODE]   
  
Code:  
  
pub fn create\_new\_edition\_nft( ctx: Context<CreateNewEdition>, edition: u64, ) -> Result<()> { let edition\_infos = vec![ ctx.accounts.token\_program.to\_account\_info(), ctx.accounts.new\_metadata.to\_account\_info(), ctx.accounts.new\_edition.to\_account\_info(), ctx.accounts.master\_edition.to\_account\_info(), ctx.accounts.new\_mint.to\_account\_info(), ctx.accounts.new\_mint\_authority.to\_account\_info(), ctx.accounts.payer.to\_account\_info(), ctx.accounts.token\_account\_owner.to\_account\_info(), ctx.accounts.token\_account.to\_account\_info(), ctx.accounts.new\_metadata\_update\_authority.to\_account\_info(), ctx.accounts.metadata.to\_account\_info(), ctx.accounts.system\_program.to\_account\_info(), ctx.accounts.rent.to\_account\_info(), ]; msg!("Edition Account Infos Assigned"); invoke(&mint\_new\_edition\_from\_master\_edition\_via\_token( ctx.accounts.token\_program.key(),ctx.accounts.new\_metadata.key(),ctx.accounts.new\_edition.key(), ctx.accounts.master\_edition.key(), ctx.accounts.new\_mint.key(),ctx.accounts.new\_mint\_authority.key(), ctx.accounts.payer.key(), ctx.accounts.token\_account\_owner.key(), ctx.accounts.token\_account.key(), ctx.accounts.new\_metadata\_update\_authority.key(), ctx.accounts.metadata.key(), ctx.accounts.metadata.key(), edition ), edition\_infos.as\_slice())?; msg!("A New Edition Nft Minted !!!"); Ok(()) }#[derive(Accounts)]pub struct CreateNewEdition<'info> { /// CHECK: This is not dangerous because we don't read or write from this account #[account(mut)] pub new\_metadata: UncheckedAccount<'info>, /// CHECK: This is not dangerous because we don't read or write from this account #[account(mut)] pub new\_edition: UncheckedAccount<'info>, /// CHECK: This is not dangerous because we don't read or write from this account #[account(mut)] pub master\_edition: UncheckedAccount<'info>, /// CHECK: This is not dangerous because we don't read or write from this account #[account(mut)] pub new\_mint: UncheckedAccount<'info>, /// CHECK: This is not dangerous because we don't read or write from this account #[account(mut)] pub edition\_mark\_pda: UncheckedAccount<'info>, #[account(mut)] pub new\_mint\_authority: Signer<'info>, #[account(mut)] pub payer: AccountInfo<'info>, // /// CHECK: This is not dangerous because we don't read or write from this account #[account(mut)] pub token\_account\_owner: UncheckedAccount<'info>, // /// CHECK: This is not dangerous because we don't read or write from this account #[account(mut)] pub token\_account: UncheckedAccount<'info>, /// CHECK: This is not dangerous because we don't read or write from this account #[account(mut)] pub new\_metadata\_update\_authority: UncheckedAccount<'info>, /// CHECK: This is not dangerous because we don't read or write from this account #[account(mut)] pub metadata: UncheckedAccount<'info>, // #[account(mut)] pub token\_program: Program<'info, Token>, pub system\_program: Program<'info, System>, /// CHECK: This is not dangerous because we don't read or write from this account pub rent: AccountInfo<'info>,}  
  
WARN: THIS PARAGRAPH CONTAINS TAG: [CODE]   
  
Documentation Reference:  
  
Documentation for printing a new edition from a master edition:  
  
https://docs.metaplex.com/programs/token-metadata/instructions#print-a-new-edition-from-a-master-edition

**C1:**Only your comments have the field metadata\_mint, which field or argument do you mean?

**C2:**Here is the screenshot of it. You can see the field metadata\_mint from this pic. i.stack.imgur.com/UWtpj.png

1 **Answer**

**A1:**The documentation on metaplex you are talking about is the documentation of the solana instructions but not the rust api. They are different. If you take a look at the rust code, you will see the differences.  
  
The accounts described in the documentation match the accounts provided by the code:  
  
let accounts = vec![ AccountMeta::new(new\_metadata, false), AccountMeta::new(new\_edition, false), AccountMeta::new(master\_edition, false), AccountMeta::new(new\_mint, false), AccountMeta::new(edition\_mark\_pda, false), AccountMeta::new\_readonly(new\_mint\_authority, true), AccountMeta::new(payer, true), AccountMeta::new\_readonly(token\_account\_owner, true), AccountMeta::new\_readonly(token\_account, false), AccountMeta::new\_readonly(new\_metadata\_update\_authority, false), AccountMeta::new\_readonly(metadata, false), AccountMeta::new\_readonly(spl\_token::id(), false), AccountMeta::new\_readonly(solana\_program::system\_program::id(), false), AccountMeta::new\_readonly(sysvar::rent::id(), false),];  
  
WARN: THIS PARAGRAPH CONTAINS TAG: [CODE]   
  
The variable metadata\_mint you are talking about is used to calculate the address of the edition\_mark\_pda address on chain:  
  
WARN: THIS PARAGRAPH CONTAINS TAG: [CODE]   
  
 let (edition\_mark\_pda, \_) = Pubkey::find\_program\_address( &[ PREFIX.as\_bytes(), program\_id.as\_ref(), metadata\_mint.as\_ref(), EDITION.as\_bytes(), as\_string.as\_bytes(), ], &program\_id, );  
  
WARN: THIS PARAGRAPH CONTAINS TAG: [CODE]   
  
When you compare this with the documentation, you will see what should be provided:  
  
Edition pda to mark creation - will be checked for pre-existence. (pdaof ['metadata', program id, master metadata mint id, 'edition',edition\_number]) where edition\_number is NOT the edition number youpass in args but actually edition\_number =floor(edition/EDITION\_MARKER\_BIT\_SIZE).  
  
metadata := Master record metadata account  
  
WARN: THIS PARAGRAPH CONTAINS TAG: [CODE]   
  
metadata\_mint := master metadata mint id  
  
WARN: THIS PARAGRAPH CONTAINS TAG: [CODE]

**C1:**Thanks for answering my question! Btw, I am still a bit confused... I get that metadata mint id is used to calculate an Edition Mark PDA, but still don't get what master metadata mint id is referring to. Is it the metadata account of the master edition account? or just the mint account of the master edition account? or else

**C2:**Oh I think I get it. It's just a original mint account id. Then, within my rust function above, how can I retrieve that mint account id? I mean I don't see the original mint account among the list of accounts required for the instruction, so should I derive it from somewhere or can I access that value from some object? plz let me know @AlexN

**C3:**You can take a look at the documentation you provided it shows the relation between the metadata\_mint and the metadata. From the metadata\_mint you can compute the metadata but not the other way around. docs.metaplex.com/programs/token-metadata/… + docs.metaplex.com/programs/token-metadata/…

**C4:**Yeah I get that part, but amongst the list of accounts described on the documentation there is no field for the original mint account while the mint\_new\_edition\_from\_master\_edition\_via\_token instruction requires it. Should I just simply add the original mint account in the list or simply pass that as a argument or sth? @AlexN