Job Manager User Guide

Libero SoC v2025.1

Introduction (Ask a Question)

The Job Manager tool is a part of the Secured Production Programming Solution (SPPS) ecosystem that consists of the Libero®, Job Manager, and FlashPro Express applications as well as User HSM servers (U-HSM) and Manufacturer HSM servers (M-HSM).

This user guide describes how to use the Job Manager to organize manufacturing flow as described in the Secure Production Programming Solution (SPPS) User Guide.

The Job Manager is primarily intended for use by the Operation Engineer (OE) who is responsible for the manufacturing process organization and security.

The current version of the Job Manager is based on the Tcl interface running in command-line mode. The Job Manager supports the creation of:

- Programming Jobs for the regular production programming flow (non-HSM manufacturing flow in this document)
- Programming Jobs for the secured production programming flow (HSM-based manufacturing flow in this document)
- Bitstream files in various formats (STAPL, DAT, SPI, and so on) for use by third-party programming tools (non-HSM flow)

Programming Jobs created by the Job Manager can be programmed into the device using FlashPro Express (see the FlashPro Express User Guide) or In House Programming (IHP).

Programming bit stream generation supported by the Job Manager is based on design information imported from Libero. eNVM/sNVM data and security settings can be modified in the Job Manager project, which allows update of areas such as M3 firmware image and enforcement of security policies.

The Job Manager allows generation of the programming bitstreams outside the Libero flow, eliminating the need for the OE to handle the design using Libero, obtain design level Libero licenses, and address design migration to newer versions of Libero.

Table of Contents

1. Manufacturing Flows (Ask a Question)

The following sections discuss the different types of manufacturing flows.

1.1. HSM-based Manufacturing Flow (Ask a Question)

The HSM-based manufacturing process uses device-supported security protocols. For more information, refer to the Secure Production Programming Solution (SPPS) User Guide.

The U-HSM allows the user to generate and use various encryption and pass keys. Cryptographical operations requiring those keys are executed inside the security boundaries of the HSM Module:

Operation support for the HSM-based Manufacturing Flow is as follows:

- Key import, generation, and use under protection of the U-HSM and the M-HSM
- Initial secure key injection into the device using the Authorization Code protocol
- Secure data transmission between the U-HSM and the M-HSM
- Overbuild protection
- Device authentication
- U-HSM verification of cryptographically sealed certificate of conformance (CoC) from the design programmed into the device
- U-HSM verification programming job end certificates
- Initiator and Upgrade programming job types
- eNVM and sNVM client update
- eNVM client selection
- Security overwrite
- Secure debug with SmartDebug tool

1.1.1. HSM Server Requirements (Ask a Question)

In the HSM-based Manufacturing Flow, the Job Manager is configured to work with the U-HSM to generate the HSM Programming Job, and HSM Programming Jobs require the M-HSM for job execution. For more information about HSM servers, refer to the Secure Production Programming Solution (SPPS) User Guide. For information about installation, refer to the User HSM Installation and Setup User Guide for the U-HSM and the Manufacturer HSM Installation and Setup User Guide for the M-HSM.

1.2. Non-HSM Manufacturing Flow (Ask a Question)

Non-HSM Manufacturing Flow support by the Job Manager allows bitstream file and programming job generation outside of the Libero tool. The Job Manager supports the following operations for non-HSM Manufacturing Flow:

- Initial key loading via KLK-protected bitstreams
- Generate UEK1/UEK2/UEK3 encrypted update bitstreams
- eNVM and sNVM client Update
- eNVM client selection
- Key value overwrite
- Security overwrite

2. HSM Parameter Configuration (Ask a Question)

To use Job Manager in the HSM flow, U-HSM parameters must be set. These parameters are stored in the user-level DEF file and automatically loaded for any new or existing Job Manager project.

U-HSM configuration data specifies:

- IP address of the U-HSM server
- U-HSM UUID assigned by Microchip
- U-HSM Master Key UUID
- Default location of the keyset repository (see the Libero SoC Design Flow User Guide)
- M-HSM UUID assigned by Microchip

The `set_hsm_params` Tcl command is used to configure HSM. See Tcl Interface for details.

3. Keyset File (Ask a Question)

The keyset file is used in the HSM flow only. It contains U-HSM generated keys encrypted with the Master Key of the U-HSM. New keyset files can be generated randomly by the HSM, derived from the existing keyset files, or created from the imported plain text values.

Keyset files are stored in the keyset repository, and can be shared among different Job Manager projects. The repository is configured through HSM parameters (refer to the FlashPro Express User Guide).

Keyset files contain the following keys:

- Ticket Key that encrypts all keys in the keyset file.
  - Ticket Key is protected by the HSM Master Key.
  - Keyset file contains HSM Master Key UUID to help identify the origin of the file.
- Encryption keys: UEK1, UEK2, UEK3
- Pass keys: UPK1, UPK2, DPK
- Base keys for deriving per-device key values for UEK1/UEK2/UEK3/UPK1/UPK2/DPK

For more information about SPPS key management, refer to the Secure Production Programming Solution (SPPS) User Guide.

A keyset file is generated outside the Job Manager project and is associated upon creation of a new Programming Data entry (refer to the User HSM Installation and Setup User Guide).

The `create_keyset` Tcl command is used to manage keyset files. See Tcl Interface for more information and keyset generation scenarios.

4. Programming Data (Ask a Question)

Programming Data entry is created from the design information imported from Libero in a Job Data Container (JDC) file (see the Libero SoC Design Flow User Guide for details about JDC Export for a Libero project). Programming Data contains all information required for bitstream generation in the HSM flow and non-HSM flow. A bitstream that is initialized for non-HSM flow can be exported directly from Programming Data into a bitstream file in a format selected by the user.

The following design data modifications can be done by the user within Programming Data:

- One or more eNVM or sNVM clients can be updated with an image loaded from external data files.
- Design security can be overwritten from the external Security Policy Manager (SPM) file from Libero.
- In non-HSM flow, plain text values of the encryption and pass keys can be changed. In the HSM flow, all key values are used from the HSM-protected keyset file.

A Job Manager project can have one or many Programming Data entries to support programming of multiple Microchip devices on the same board.

The following sections provide information about the creation and modification of data in Programming Data entry.

4.1. Create Programming Data from JDC File (Ask a Question)

New Programming Data is created using the `new_prog_data` Tcl command. It must be created within an existing or new Job Manager project.

When creating a new Programming Data entry, design data is copied from the external JDC file within the current project. After this step, the external JDC file is no longer used.

In the HSM flow, all keys are generated and controlled by the U-HSM. Therefore, in this flow, a new Programming Data must be associated with a keyset file created as shown in Keyset File. All cryptographic operations involving protected keys are executed inside the HSM module. For example, a Master bitstream is generated using the U-HSM and programmed via the Authorization Code protocol using the M-HSM.

4.2. eNVM and sNVM Update (Ask a Question)

This is an optional step that allows the user to modify one or more eNVM or sNVM clients found in the design loaded into the Programing Data. Refer to the Secure Production Programming Solution (SPPS) User Guide for the use model definition.

To add a client update, the design in the Programming Data entry must have an eNVM component that already contains a target client. The size of the update data must be equal to or smaller than the client size. See the `set_envm_update` and `set_snvm_update` Tcl commands in Tcl Interface for details.

4.3. Key Overwrite (Non-HSM Flow) (Ask a Question)

This feature applies to the non-HSM flow and allows the user to modify plain text key values for the keys coming from Libero as a part of the design security settings. New key values can be set for UEK1/UEK2/ UEK3/UPK1/UPK2/DPK using the `set_key` Tcl command. Key overwrites can be reverted using the `remove_key` Tcl command.

4.4. Security Overwrite (Ask a Question)

Security overwrite ignores security settings imported into the Programming Data entry from Libero (via the JDC file) and uses settings imported from the SPM file on disk. After import, the external SPM file is no longer used by the Job Manager.

Security overwrite is available in HSM flow and non-HSM flow. In both flows, security policy is used from the overwrite, while key values follow these rules:

HSM Flow:

- Key values are always used from the keyset file.

Non-HSM Flow:

- Key values are used from the security settings of the security overwrite.
- For keys that were overwritten using the `set_key` command (see Key Overwrite (Non-HSM Flow)), the key overwrite value is used.

For details, see the `set_security_overwrite` and `remove_security_overwrite` Tcl commands in Tcl Interface.

4.5. Bitstream Initialization (Ask a Question)

Bitstream initialization allows the user to setup bit stream generation parameters for use during:

- HSM job export
- Non-HSM job export
- Export of a programming bit stream in non-HSM flow

The bit stream can be initialized for use in the HSM programming flow and non-HSM programming flow. For use model details about HSM and non-HSM bitstreams, refer to the Secure Production Programming Solution (SPPS) User Guide.

Programming Data can have one or more bit stream entries that can be used by the same or different programing jobs.

Bitstream initialization is done with the `init_bitstream` Tcl command.

4.5.1. HSM Flow (Ask a Question)

The following sections describe how to initialize and use programming bitstreams for various situations in the HSM flow. Because all keys in the HSM flow are protected by the HSM, the U-HSM must generate bitstreams for all cases described below.

Initiater Bitstream

The Initiater bit stream in HSM is designed to program initial security and all other user-selected device features in an untrusted environment. Secure key loading is achieved using the device-supported Authorization Code protocol. Refer to the Secure Production Programming Solution (SPPS) User Guide for more information.

The Initiater bit stream can be generated to program project or per-device UEK1/UPK1/UEK2/UPK2/UEK3/DPK values.

Project keys are inserted into the bit stream from the keyset file upon bit stream generation.

Per-device keys are generated and infused into the programming bit stream by the M-HSM during device programming. Per-device key value is derived from respective base keys in the keyset file and device device serial number (DSN). Per-device protocol and key types are specified with the `init_bitstream Tcl` command parameters.

The Initiater bit stream programs security settings imported into Programming Data from Libero or according to SPM overwrite, if any.

Important: Due to security policy, after the initial key loading, programming actions such as ERASE and VERIFY require HSM support to unlock device security.

WARNING Security settings programmed into the device can only be changed with the ERASE action. The ERASE action does not erase content of the eNVM or sNVM. eNVM and sNVM memory is fully accessible after security settings have been erased.

UEK1/UEK2/UEK3 Update Bitstream

This type of bit stream is used for reprogramming Fabric and/or eNVM/sNVM depending on device type. The security component cannot be reprogrammed with this file type. This bit stream can be used if the device already has security programmed.

UEK1/UEK2/UEK3 Project Keys, No Security Lock

In this case, all devices in the project have the same UEK1, UEK2, or UEK3 values, and target device feature programming is allowed without FlashLock/UPK1 match. The Job Manager can generate a stand-alone programming file or a non-HSM programming job that does not require the M-HSM during programming.

UEK1/UEK2/UEK3 Project Keys, Security Locked

If the target device feature programming is locked, the M-HSM must perform a secured unlock of the device, because the plain text value of the lock key cannot be used in an untrusted environment. This type of bit stream can be used in an HSM programming job. UPK1 unlock is performed securely via the OTPK protocol. For more information, refer to the Secure Production Programming Solution (SPPS) User Guide.

UEK1/UEK2/UEK3 Per-Device Keys, No Security Locks, DSN is Known

If UEK1, UEK2, or UEK3 are per-device keys, target features are not locked, and DSN for the target device(s) is known, the user has an option to generate a device-specific programming bit stream that does not require the M-HSM during production. The bit stream can be exported as a stand-alone bit stream file or a non-HSM programming job. In either case, DSN must be provided in the `export_bitstream` Tcl command during bit stream file generation or in the `add_microsemi_device` Tcl command that adds target devices to the chain inside the programming job.

UEK1/UEK2/UEK3 Per-Device, All Other Cases

For all other cases related to per-device UEK1/UEK2/UEK3, the M-HSM and HSM programming job must be used. If target device features are locked by per-device UPK1, UPK1 unlock is performed securely via the OTPK protocol. For more information, refer to the Secure Production Programming Solution (SPPS) User Guide.

4.5.2. Non-HSM Flow (Ask a Question)

In non-HSM flow, the keyset file is not used. All key values are used from the Libero design security setting. There are two mechanisms to overwrite the user-defined design security keys (UPK1, UPK2, UEK1, UEK2, UEK3, and DPK):

1. Security Overwrite—The security setting and key values set supersede the original Libero settings.
2. Key Overwrite—The key values set supersede both the Libero and Security Overwrite setting.

The following sections describe non-HSM bitstream types that can be generated for production programming in a trusted environment.

Trusted Facility Bitstream

This bitstream type can program Fabric, eNVM, and non-authenticated plain text sNVM clients. The entire bitstream is encrypted with the KLK encryption key.

Master Bitstream

Similar to Trusted Facility, but also programs security and supports all types of sNVM clients. After custom security is programmed, all Microchip factory default key modes, including KLK, DFK, KFP, and KFPE key modes as well as User ECC keys (KUP and KUPE), become disabled.

Note: Per security policy, programming of UEK1 or UEK2 will program the UPK1 or UPK2 passkeys, respectively, and lock the security segment. As a result, the ERASE and VERIFY actions in generated bitstream files or programming jobs will contain plain text UPK1/UPK2 values. This is required to unlock security segments for the programming actions when using the non-HSM flow (the HSM-based flow uses encrypted one-time passcodes).

UEK1/UEK2/UEK3 Update Bitstream

This bitstream type can reprogram Fabric and/or eNVM/sNVM device features. If the target device programming is protected by FlashLock/UPK1, plain text values of UPK1 are included in the exported bitstream file/programming job. when using the non-HSM flow.

4.5.3. Export Programming Bitstream File (Ask a Question)

Export of the programming bitstream file is available in non-HSM flow only. Export is handled by the `export_bitstream_file` Tcl command and can be performed in all supported programming file types.

Specifying the optional DSN parameter is applicable only for situations explained in the Non-HSM Flow section. The exported bitstream file is created in a user-specified location.

Important: Job files designed for SmartFusion® 2, IGLOO® 2, and PolarFire® devices, from the latest versions of Libero, are compatible with HSM Server v12.6.

5. Programming Job (Ask a Question)

A Programming Job is a set of data used by programming systems for device programming in HSM and non-HSM flows.

The current SPPS ecosystem supports FlashPro Express and IHP job types.

FlashPro Express can program HSM and non-HSM jobs (refer to the FlashPro Express User Guide), and IHP supports the HSM job type only.

A Programming Job contains the following data:

- Job type (FlashPro Express or IHP)
- Job origin
- Bitstream(s) for various programming actions (PROGRAM, ERASE, and VERIFY)
- Hardware setup information – for FlashPro Express job type
  - Type of hardware interface (JTAG in this version of the Job Manager)
  - Configuration
- Job device(s) – includes basic device information
- Data for HSM flow only
  - M-HSM UUID
  - U-HSM UUID
    - Encrypted Job Tickets authorizing programming actions and overbuild protection under control of the HSM
    - Encrypted keys and security protocol data required by HSM protocols

5.1. HSM Programming or Debug Job (Ask a Question)

A programming or debug job can be set up for execution in an untrusted environment. In this case, the M-HSM protects cryptographic key material and other sensitive data required by device security protocols during the manufacturing processes.

HSM data is added to the Programming or Debug Job with HSM Task(s). See HSM Tasks for details. A Programming Data entry is created in the Job Manager project using the `new_prog_job` Tcl command.

5.2. Creating a FlashPro Express or SmartDebug Job (Ask a Question)

After creating a FlashPro Express or SmartDebug job, the user specifies the type of hardware setup:

JTAG Chain – supported in the current release of the Job Manager

5.2.1. Configuring a JTAG Chain (Ask a Question)

The following device types can be added to a JTAG chain:

- Microchip device targeted for programming
- Microchip bypass device not targeted for programming
  - Non-Microchip bypass device

Each device has a user-defined name that is unique within the JTAG chain. Microchip devices can be programmed using a bit stream generated by a Programming Data entry or by using existing programming bit stream files (STAPL) loaded from disk.

Adding a Microchip Device for Programming by Generated Bitstream

If a Microchip device is programmed by a bit stream generated from Programming Data, the bit stream must be specified in the Programming Data and bit stream name parameters in the `add_microsemi_prog_device` Tcl command (see Tcl Interface for details).

The "DSN" parameter is used in the HSM flow to create a device-specific update programming file, see UEK1/UEK2/UEK3 Update Bitstream. Actual bit stream is generated while exporting the programming job, see HSM Task Export.

Adding a Microchip Device for Programming by Existing Bitstream File

A Microchip device can be programmed using the existing bit stream file generated outside the Job Manager.

Important: If an external bit stream file has been loaded in the device, it cannot be programmed using HSM, and tickets cannot be created.

Use the `add_microsemi_prog_device` Tcl command pointing to the target bit stream file on disk specifying the path to the file with the "bitstream_file" parameter.

Adding a Microchip Bypass Device

A Microchip bypass device can be added by specifying the device name or pointing to the device programming file.

Refer to the `add_microsemi_bypass_device` Tcl command for more information.

Adding a Non-Microchip Bypass Device

A non-Microchip device can be added to the JTAG chain with the `add_non_microsemi_bypass_device` Tcl command.

JTAG bypass parameters can be specified either by pointing to the BSDL file accepted by the command or by explicit parameter specification. Refer to the `add_non_microsemi_bypass_device` Tcl command for more information.

5.2.2. Export of Non-HSM Programming Job (Ask a Question)

A non-HSM Programming Job is exported from the Job Manager with the `export_prog_job` Tcl command. All bitstreams generated from Programming Data entries are created during command execution.

Important: A Programming Job that has one or more HSM tasks is considered to be HSM type and cannot be exported using the `export_prog_job` command. For more information, see the HSM Tasks section.

5.2.3. HSM Tasks (Ask a Question)

The HSM task in the HSM flow allows flexibility in organizing the manufacturing process. It is possible to utilize multiple Contract Manufacturers simultaneously, or the entire manufacturing volume can be split onto smaller chunks for overbuild protection. For example, after creating a Programming Job, the OE can create and export an HSM Task for each manufacturer in production.

HSM tasks add HSM data to the Programming Job. For each HSM task, the user creates job tickets and specifies programming actions for each ticket. Overbuild protection and other protocol-specific information is specified during ticket creation.

For more information about the HSM use model and flow description, refer to the Secure Production Programming Solution (SPPS) User Guide.

Job Tickets

The HSM Task Ticket (Job Ticket in this document) is used to enforce security policies on the manufacturing side and encrypt sensitive information used by device security protocols.

A Job Ticket is created per device in the Programming Job. Each device can have one or more ticket. The Job Ticket is created per the user-selected programming action.

A new Job Ticket is created with the `new_hsmtask_ticket` Tcl command. The `max_device` parameter is used to limit the number of devices a programming action can be executed on.

Job Request

A Job Request is exported from the Job Manager Project after creation of all tickets within the HSM Task. The Job Request is then sent to and processed by FlashPro Express or IHP using its M-HSM.

A Job Request is created with the `hsmtask_m_request` Tcl command.

Job Reply

A Job Reply returns ticket generation information created by the FlashPro Express/IHP. This information is cryptographically bound to the physical M-HSM/U-HSM module that processed the Job Request. After performing this handshake protocol, the HSM Job exported from this HSM Task can only be used with that particular module. This prevents HSM Task replication on the manufacturing side.

A Job Reply is generated by FlashPro Express or IHP and can be imported into the requesting U-HSM Task with the `hsmtask_m_reply` Tcl command.

HSM Task Export

An HSM Task (HSM Job in this document) can be exported with the `export_hsmtask` Tcl command. This command executes the part of export done during the non-HSM job export and adds HSM-specific information to the job export container. This data includes job tickets, encryption keys, protocol data, and other HSM-specific information. The HSM job can only be exported after importing the Job Reply.

Job Status

Job Status can be generated by FlashPro Express or IHP during job execution or after ending the job. A Job Status file is generated and sent to the customer.

An HSM Programing Job being executed on the manufacturing side can be ended when all target devices are programmed, or the job can be terminated at any time.

The Job Manager uses Job Status to:

- Validate job end status, which is cryptographically protected proof that the job has ended and can no longer execute programming actions controlled by its tickets
- Display the number of devices that can be handled by each ticket
- Ensure that the correct bit stream is programmed into each device by validating the CoCs.

6. Export SPI Directory (Ask a Question)

The Job Manager can be used to create the SPI directory used by SmartFusion® 2/IGLOO® 2 devices during auto-update and programming recovery. The SPI directory contains information about golden and update programming bitstreams placed by the user into SPI Flash.

The SPI directory is created with the `export_spi_directory` Tcl command. It does not require an existing Job Manager project.

The version number for the golden and update bitstreams can be entered manually or by providing previously generated SPI files. The user also needs to provide addresses for both images in the final Flash memory.

For more information about using the SPI Flash directory, refer to the Libero SoC Design Flow User Guide.

7. Key Rotation (Ask a Question)

Key rotation flow enables the user to securely update user keys (UPK1, UEK1, UPK2, UEK2, DPK and UEK3) in SmartFusion 2 and IGLOO 2 devices.

The key rotation flow is as follows:

1. The Libero SoC tool is used to create the design and export the JDC file.
2. Job Manager uses the JDC file to generate the master and key rotation update bitstreams.
3. Program devices with initial master bitstream that does not lock user key segments so it can update user keys in future.
4. Program key rotation bitstreams in sequence, which updates the user keys.

7.1. Master Bitstream Job Flow (Ask a Question)

Users can generate the Master Bitstream Job for Hardware Security Modules (HSM) or Non-HSM flow. The parameter `enable_key_rotation` in `init_bitstream` Tcl command is used to allow updating user keys in future.

Note:
When the key rotation is enabled, the master bit stream generated does not lock User Lock-Bit, User Key1, and User Key2 segments to allow user keys to be updated in the future.

When key rotation is enabled, Job Manager checks for the following:

- If Design version is set in JDC file.
- Verify that no permanent security setting is selected.
- If more than one keysets are being programmed.

Note:
The Key Rotation Flow is not supported for `UNIQUE_KEY` protocol.

7.2. Update Bitstream Job Flow (Ask a Question)

Using Job Manager you can generate a stand-alone bit stream file (STAPL/SPI/DAT), which will update the user keys that are already programmed on device using the Master Bitstream Flow. Since multiple Key Sets need to be updated, there will be multiple key rotation bit stream files (one for each key set). The multiple key rotation bit stream files need to be programmed in a specific order. The order in which they need to be programmed is embedded as part of the file name.

Users can specify the new key values, which are used to replace old keys programmed in the device using the `set_new_keys_for_rotation` Tcl command. Depending on whether HSM is being used, the user specifies the key values:

- If it is an HSM flow, then the user must create a new keyset file with new keys values and use the keyset file as input to the Tcl command.
- If it is a non-HSM flow, then the user must specify new keys values, which must be updated in this Tcl command. Any key that is not specified will retain its original value.

User can use the `init_bitstream` Tcl command to initialize bit stream that allows users to specify the key rotation parameters.

7.3. HSM Flow (Ask a Question)

If the Master Bitstream Flow was run using HSM, then in the Update Bitstream File Flow, user must create a new key set using the `create_keyset` Tcl command and set the new keyset file as input in the `set_new_keys_for_rotation` Tcl command. User then specifies the bit stream type as KEY_ROTATION in the `init_bitstream` Tcl command and then exports the file using the

export_bitstream_file Tcl command. This generates multiple bit stream files, one for each user key set.

```
set_new_keys_for_rotation -data_name "design data name" \
                          [-keyset_file "keyset file name"]

init_bitstream -data_name "design data name" \
               -bitstream_type "bitstream name" \
               -bitstream_type "KEY_ROTATION" \
               [-auto_inc_design_version] \
               [-first_keymode "UEK1 | UEK2 | UEK3"]

export_bitstream_file -data_name "design data name" \
                      -bitstream_name "bitstream name" \
                      -formats "[STAPL | SPI | DAT]" \
                      -export_path "export_file"
```

7.4. Non-HSM Flow (Ask a Question)

If the Master Bitstream Flow was run without HSM, then in the update bitstream file flow user specifies new values for keys via TCL command. Any key that is not specified will retain the original value (from Master Bitstream Flow). Details about each key is logged while running set_new_keys_for_rotation TCL command.

```
set_new_keys_for_rotation -data_name "design data name" \
                                  [-upk1 "security key value (UPK1)"]
\
                                  [-uek1 "security key value (UEK1)"] \
                                  [-upk2 "security key value (UPK2)"] \
                                  [-uek2 "security key value (UEK2)"] \
                                  [-dpk "security key value (DPK)"] \
                                  [-uek3 "security key value (UEK3)"] \

 init_bitstream -data_name "design data name" \
                -bitstream_type "bitstream name" \
                -bitstream_type "KEY_ROTATION" \
                [-auto_inc_design_version] \
                [-first_keymode "UEK1 | UEK2 | UEK3"]

export_bitstream_file -data_name "design data name" \
                      -bitstream_name "bitstream name" \
                      -formats "[SPI | DAT]" \
                      -export_path "export_file"
```

Example 7-1. Details About Key Log On Running set_new_keys_for_rotation TCL Command

```
Info: UEK1 will not be updated during key rotation as no new key value is
specified.
Info: UEK2 will not be updated during key rotation as it is absent in JDC.
Info: UEK3 will not change during key rotation as the old and new values are
same.
Info: UPK1 will not change during key rotation as the old and new values are
same.
Info: UPK2 will not be updated during key rotation as it is absent in JDC.
Info: DPK will be updated during key rotation.
```

Note:
Key rotation files for each user key set are always exported (UKS1, UKS2, and UEK3) even if the key value is same. For example, if user does not specify new key value for UPK1 and UEK1, then original key values are used to generate the key rotation bitstream, so that it amounts to reprogramming, same key value.

7.5. Key Rotation Bitstream Files (Ask a Question)

When export_bitstream_file Tcl command is run for KEY_ROTATION bit stream type, multiple files will be generated, one file for each user key set used in JDC.

The key rotation bit stream files are named in the following format:

`<user_specified_filename>_rotate_{1|2|3}_uks{1|2|3}`

Where:

- `<user_specified_filename>` is the file name specified in the `export_path` of `export_bitstream_file` command.
- Number after `rotate_` is the file order in which it must be programmed.
- Number after `uks` is the user key set that is being modified by this key rotation file.

Example 7-2. Example to Explain the Key Rotation Bitstream File Format

If exported file name is `myUpdateProgFile_rotate_1_uks3.spi`, then `myUpdateProgFile` is the same as user specified in the `export_bitstream_file`. `rotate_1` denotes it is the first file that needs to be programmed, `uks3` denotes that the SPI file will update User Key Set 3 (UEK3).

If the JDC design has all the three keysets being programmed and the key rotation bitstream files are being generated with name `myUpdateProgFile` then one set of possible key rotation bitstream files can be: `myUpdateProgFile_rotate_1_uks3.spi`, `myUpdateProgFile_rotate_2_uks1.spi`, and `myUpdateProgFile_rotate_3_uks2.spi`.

If the JDC file has Auto-Update enabled, then design version is automatically incremented for each key rotation bitstream file and info messages log this during bitstream generation.

Example 7-3. Example of Message Log Showing Successful Generation Of Key Rotation Bitstream File

```
Info: Successfully generated key rotation bitstream file
'myprogfile2_rotate_1_uks3.spi'. This will update the design version to 1435.
Info: Successfully generated key rotation bitstream file
'myprogfile2_rotate_2_uks1.spi'. This will update the design version to 1436.
Info: Successfully generated key rotation bitstream file
'myprogfile2_rotate_3_uks2.spi'. This will update the design version to 1437.
```

7.6. Export SPI Directories (Ask a Question)

User must export multiple SPI directory files, one for each key rotation bitstream file and use it in the Auto-Update flow using the `export_spi_directory` TCL command.

8. Tcl Interface (Ask a Question)

The following sections discuss the Tcl interface.

8.1. Application (Ask a Question)

```
set_hsm_params -hsm_server_name <hsm_server> -u_hsm_uuid <u_uuid> - u_master_hsm_uuid
<u_master_uuid > -hsm_key_set_dir <keyset_dir> - m_hsm_uuid <m_uuid>
```

- `hsm_server` - Name or IP address of HSM server machine.
- `u_uuid` - User HSM UUID.
- `u_master_uuid` - User HSM Master UUID.
- `keyset_dir` - Keyset repository location: a directory in which the keyset files will be created or used.
- `m_uuid` - Manufacturer HSM UUID. This command saves the HSM parameters for the Job Manager application. This remains in effect until its overridden using this same command.

`get_software_info [-version]`

- `version` - Get the software version info.

This command prints the Job Manager software information.

8.2. Keyset Management (Ask a Question)

```
create_keyset -file <output_file_name> [-source_file
<source_file_name>] [-kip <token_key>]
[-upk1 <token_key>] [-upk1_base <base_key>]
[-uek1 <token_key>] [-uek1_base <base_key>]
[-upk2 <token_key>] [-upk2_base <base_key>]
[-uek2 <token_key>] [-uek2_base <base_key>] [-dpk <token_key>]
[-dpk_base <base_key>]
[-uek3 <token_key>] [-uek3_base <base_key>]
```

- `output_file_name` - Name of the new keyset file. Name only, no path. The file is created in the keyset repository defined by the Job Manager application settings.
- `source_file_name` - Optional source keyset file name for key import. The ticket key in this file must be encrypted with the same U-HSM Master Key.
- `token_key` - Optional value of the token keys that will be imported into the new keyset. This parameter takes precedence over the keys in the source file, if specified. Token key values can only be specified in plain text format.
- `base_key` - Optional value of the base keys (that is, keys are used to derive device-specific token keys) that will be imported into the new keyset. This parameter takes precedence over the keys in the source file, if specified. Base key values can only be specified in plain text.

This command creates a new keyset file for the HSM flow tasks. It can create keys for the following scenarios:

- Create a new keyset file.
  - All keys are randomly generated by the U-HSM.
- Create a new keyset file with key import.
  - Ticket key is randomly generated.
  - User-specified keys are imported and protected by the ticket key.
  - The rest of the keys are randomly generated and protected by the ticket key.
- Create a modified copy of the existing keyset file.

  - New keyset file receives copies of the keys from the source file including ticket key.
- Ticket is protected by the same U-HSM Master Key.
  - User-specified keys will be imported into the new keyset file in place of the keys in the source keyset file. All imported keys are protected by the ticket key from the source file.

Notes:

1. Because the same keyset file can be shared between different Job Manager projects, `create_keyset` always creates a new file. It cannot delete, rename, or overwrite existing keyset files. All such operations should be handled by the user manually.
2. Keys specified in the keyset file always take effect regardless of whether `set_security_overwrite` command is run.
3. UEK3 is only available for M2S060, M2GL060, M2S090, M2GL090, M2S150, and M2GL150 devices.

8.3. Project Management (Ask a Question)

```
new_project -location <path> -name <file_name>
```

- `path` - Top level project directory.
- `file_name` - Project file name.

This command creates a new Job Manager project. The project directory can be moved to any other location on disk as project internally uses only relative paths. If the project already exists, command will exit with error.

```
open_project -project <path>
```

- `path` - Path to the Job Manager project file (.jprj).

This command opens an existing Job Manager project.

```
close_project [-save <TRUE |FALSE>]
```

This command closes the Job Manager project with or without saving it. If project was modified, then - save option must be specified.

8.4. Programming Data (Ask a Question)

8.4.1. Design Import (Ask a Question)

```
new_prog_data    -data_name <name>
                 -import_file <path>
                 [-keyset_file < keyset_name>]
```

- `name` - Programming data entry name.
- `path` - Libero design data file (JDC).
- `keyset_name` - Name of the keyset file. File name only. The file will be in the keyset directory as specified in the Application settings.

This command creates a new Programming Data entry from JDC (design data exported from Libero project). If keyset parameter is specified, HSM handles all the key management.

8.4.2. Security Modifications (Ask a Question)

```
set_key -data_name <name>
[-upk1 <upk1_value>] [-uek1 <uek1_value>] [-upk2 <upk2_value>] [-uek2
<uek2_value>] [-dpk <dpk_value>] [-uek3 <uek3_value>]
```

- `name` - Name of the Programming Data.

- uek*_value - New value for the selected encryption key.
- upk*_value - New value for the selected pass key.
- dpk_value - New value for the DPK.

This command overwrites key values imported from JDC with the ones specified in the command arguments.

Notes:
- This command is applicable to non-HSM flow only.
- Keys specified by this command always take affect irrespective of whether set_security_overwrite command is run.
- UEK3 is only available for M2S060, M2GL060, M2S090, M2GL090, M2S150, and M2GL150 devices.

```
remove_key    -data_name <name>
              -key_names <ALL UPK1 UEK1 UPK2 UEK2 DPK> UEK3
```

- name - Name of the Programming Data.
- key_names - Names of the keys whose values to be reverted (can specify multiple values).

This command reverts the value of the specified encryption and/or pass key to the value imported from JDC. This command supports action opposite to set_key.

Notes:
- This command is applicable to non-HSM flow only.
- UEK3 is only available for M2S060, M2GL060, M2S090, M2GL090, M2S150, and M2GL150 devices.

```
set_security_overwrite -data_name <name> -file <spm_file_path>
```

- name - Name of the Programming Data entry.
- spm_file_path - File path of the new SPM file.

This command overwrites design security settings imported from Libero through the new SPM file. This command is applicable to HSM and non-HSM flows. How the security policies and keys are overridden is explained below:

- Security policies: Always overwritten from the new SPM file (HSM and non-HSM flows)
- Security Keys (UPK1, UEK1, UPK2, UEK2, UEK3, and DPK)
  - Non-HSM flow:
    - If set_key is run before or after this command, then all keys specified in set_key are used. The keys that are not specified are taken from the new SPM file provided as argument.
    - If set_key is not run, then all security keys are taken from the new SPM file.
  - HSM flow:
    - Security keys are always taken from the keyset file.

Note: UEK3 is only available for M2S060, M2GL060, M2S090, M2GL090, M2S150, and M2GL150 devices.

```
remove_security_overwrite -data_name <name>
```

- name - Name of the Programming Data entry.

This command removes existing security overwrite settings. This command supports action opposite to set_security_overwrite.

Note:
This command will fail if `set_security_overwrite` is not run previously.

```
set_new_keys_for_rotation -data_name <name> \
                          [-keyset_file <keyset_name>] \
                          [-upk1 <upk1_value>] \
                          [-uek1 <uek1_value>] \
                          [-upk2 <upk2_value>] \
                          [-uek2 <uek1_value>] \
                          [-dpk <dpk_value>] \
                          [-uek3 <uek3_value>]
```

- `name` - Name of the Programming Data.
- `keyset_name` - Name of the keyset file containing new key values.
- `uek*_value` - New value for the selected encryption key.
- `upk*_value` - New value for the selected pass key.
- `dpk_value` - New value for the DPK.

This command is used to specify the new key values that user wants to replace with an already programmed secured device as part of Key Rotation flow. For more information, see HSM Flow and Non-HSM Flow.

Notes:

- `keyset_file` parameter is only applicable to the HSM flow.
- upk1, uek1, upk2, uek2, dpk, and uek3 parameters are only applicable to non-HSM flow.
- UEK3 is only available for M2S060, M2GL060, M2S090, M2GL090, M2S150, and M2GL150 devices.

8.4.3. eNVM Client Modifications (Ask a Question)

```
set_envm_update   -data_name <data>
                  -client_name <client>
                  -file < file_path>
                  [-overwrite <YES | NO>]
```

- `data` - Programming data entry name.
- `client` - Name of the target eNVM client.
- `file_path` - File path of the client update data file.

This command creates a new client update entry in the specified Programming Data. If an update already exists for that client, the command will fail unless the optional `-overwrite` parameter is set to "yes". Data file size must not exceed the client size defined by the Libero project and imported via the JDC file. This command allows using all eNVM data file formats that Libero supports. For more information, see Libero SoC Design Flow User Guide. This command is available for the PolarFire® SoC device family.

```
remove_envm_update   -data_name< data >
                     -client_name < client >
                     -all <YES| NO>
```

- `data` - Programming data entry name.
- `client` - Name of the target eNVM client.
- `all` - Removes all clients. Default is "YES"

This command removes an entry that `set_envm_update` created. All client updates can be removed using "all" key set to "YES". A removed client update results in the client being reverted to the original eNVM data received in the JDC.

8.4.4. sNVM Client Modifications (Ask a Question)

```
set_snvm_update   -data_name <data>
                  -client_name <client>
                  -file < file_path>
                  [-overwrite <YES | NO>]
```

- `data` - Programming data entry name.
- `client` - Name of the target sNVM client.
- `file_pathc` - File path to the client update data file.

This command creates a new client update entry in the specified Programming Data. If an update already exists for that client, the command will fail unless the optional `-overwrite` parameter is set to "YES". Data file size must not exceed the client size defined by the Libero project and imported via the JDC file. This command allows using all sNVM data file formats that Libero supports. For more information, see Libero SoC Design Flow User Guide.

The command is available for the PolarFire and PolarFire SoC device family.

8.4.5. Bitstream Management (Ask a Question)

```
init_bitstream   -data_name "design data name" \
        -bitstream_name "bitstream name" \
        -bitstream_type "TRUSTED_FACILITY | MASTER | UEK1 | UEK2 | UEK3 | KEY_ROTATION |
DEBUG" \
        [-features "[ALL | FABRIC | ENVM | SNVM | SECURITY]+"] \
        [-use_protocol "AUTH_CODE | UNIQUE_KEY"] \
        [-unique_key_types "[UPK1 | UEK1 | UPK2 | UEK2 | UEK3 | DPK]+"] \
        [-envm_clients "[eNVM client selection]+"] \
        [-snvm_clients "[sNVM client selection]+"] \
        [-generate_coc "TRUE | FALSE"] \
        [-auth_keymode "DFK | KFPE | KFP | KUP | KUPE"] \
        [-enable_key_rotation "TRUE | FALSE"] \
        [-auto_inc_design_version] \
        [-first_keymode "UEK1 | UEK2 | UEK3"] \
        [-enable_passkey_export "TRUE | FALSE"]
```

- `data_name` - Name of the Programming Data entry.
- `bitstream_name` - Name of the bit stream being added (without path or extension).
- `bitstream_type` - Specifies bit stream type. The resulting bit stream will be generated according to the security policy specified in the Programming Data Security settings. Only one of the following bit stream types can be selected at a time:
  - TRUSTED_FACILITY - For the Non-HSM flow only. Programs selected bit stream components (Fabric, eNVM, or non-authenticated plain text sNVM clients) using KLK key mode.
  - Initiator - Programs security and any other selected components.
    - For the Non-HSM flow: Uses the KLK key mode to program security and other components. If security is programmed with UEK1/UEK2, then bit stream will also include plaintext UPK1/UPK2 respectively to enable erase and verify actions.
    - For the HSM flow: Uses DFK, KFP, KFPE, KUP, and KUPE key modes to program security and other components. In this case, all keys are encrypted by HSM.
      - UEK1 - Programs (updates) selected bit stream component(s) (Fabric/eNVM/sNVM) using UEK1 key mode. If the target component is update-protected, UPK1 will be used to unlock target component.
        - For the Non-HSM flow, generated bit stream uses plain text UPK1 value unlocking security.
        - For the HSM flow, the M-HSM performs security unlock using the encrypted value of UPK1.
      - UEK2 - Similar to UEK1, but uses UEK2 and UPK2 respectively.

  - UEK3 - Similar to UEK1, but uses UEK3. Supported for M2S060, M2GL060, M2S090, M2GL090, M2S150, and M2GL150 devices.
- KEY_ROTATION - Specifies that bit stream will be used for updating user keys, see Key Rotation section.
- DEBUG - Initializes bit stream that can be used for debugging with tools like SmartDebug that require HSM support to unlock device debug features using One Time Passkey protocol protected by user encryption keys. This type of bit stream requires specification of the protocol (`ONE_TIME_PASSCODE`) in the current version and `auth_keymode` that can be UEK1 or UEK2 depending on security settings.

- `features` - Any combination of the features or ALL (default option) SECURITY is selected for programming.

  Important: Security is always programmed in the Initiator bit stream.

  - FABRIC - Selected for programming.
  - eNVM - Selected for programming. Supported for SmartFusion 2, IGLOO 2, and PolarFire SoC devices.
  - sNVM - Selected for programming. Supported for PolarFire and PolarFire SoC devices.
  - ALL - All features available in Design Data to be selected (default).

  Important: For PolarFire devices, Fabric and sNVM must be programmed all at once. Separate programming of sNVM disables Fabric.

- `use_protocol` - Allows selecting one of the following HSM security protocols:
  - AUTH_CODE - Securely sends Encryption Key (KIP) to the device using Authorization code. Use this option when all the user keys are project keys. This protocol supports INITIATOR bit stream flow and UEK1/UEK2/UEK3 update bit streams flow.
  - UNIQUE_KEY - Allows programming unique per-device keys in the Initiator flow. For the UEK1/UEK2/UEK3 update flow, this option can be used to generate per-device bit stream files (if DSN is known at the time of job or bit stream file generation) or to use per-device keys during programming jobs using the M-HSM.
- `hsm_protocol_parameters` -
  `-unique_key_types <UEK1 UEK2 UPK1 UPK2 DPK UEK3>`
  Parameter required for UNIQUE_KEY protocol. It specifies which of the user keys are per-device keys. Each per-device key is derived from a base key and Device Serial Number (DSN) during programming, which makes it device-specific.

  Important: UEK3 is supported for M2S060, M2GL060, M2S090, M2GL090, M2S150, and M2GL150 devices only.

- `generate_coc` - Parameter to setup the generation and export of Certificate of Conformance. This command allows the user to setup parameters for bit stream generation. Actual generation occurs upon job or bit stream file export. Bitstream generation is based on design information in the Programming Data entry, including all modifications such as eNVM update, security overwrites, and so on.

- auth_keymode - Specifies what key mode to use for Authorization component in INITIATOR bit stream HSM flow. This parameter is required for M2S060, M2GL060, M2S090, M2GL090, M2S150, M2GL150, and all PolarFire and PolarFire SoC devices and optional for the rest of SmartFusion 2 and IGLOO 2 devices. The following table shows the valid and default value of this parameter.
- enable_passkey_export - By default, the programming or the job file does not contain plain text UPK values. Use –enable_passkey_export to include plain text UPK values in the STAPL or the JOB file. Set to TRUE in the non-HSM flow to allow erase/verify operations, if security is programmed.

Important: By default, UPKs are not staged to prevent security risks.

Device	Valid Values
PolarFire® and PolarFire SoC devices M2S060, M2GL060, M2S090, M2GL090, M2S150, M2GL150	KFP, KFPE, KUP, KUPE
Other SmartFusion® 2 and IGLOO® 2 devices	DFK

- enable_key_rotation - Used to generate an initiator file, which allows updating user keys using Key Rotation flow. The UNIQUE_KEY protocol is not supported.
- auto_inc_design_version - Flag to automatically increase design version in each of the exported key rotation bit stream file. This parameter is set by default if Auto-Update flow is enabled in JDC.
- first_keymode - Denotes the key mode used to encrypt the first key rotation bit stream file (*_rotate_1_uks*). If user is aware that particular User Key Set is not compromised, then they can use it to securely start the key rotation flow.

The following is an example of bit stream settings for the SmartDebug tool.

```
init_bitstream -data_name {MyProgData} -bitstream_name {MyDebugBitstream}
-bitstream_type {DEBUG} \
-features {ALL} -use_protocol {ONE_TIME_PASSCODE} -auth_keymode {UEK1}
```

```
export_bitstream_file   -data_name <name>
                        -bitstream_name <bitstream_name>
                        -formats <format_selection>
                        -export_path <path > [-dsn <dsn_value>]
```

- name - Name of the programming data.
- bitstream_name - Name of the bit stream entry to be exported.
- format_selection - Any combination of the format types. File will be created for each selected type:
  - STAPL file (non-HSM and HSM update flow)
  - SPI file (non-HSM and HSM update flow)
  - DAT file (non-HSM and HSM update flow)
- path - Complete file path without any extension. Extensions are determined based on format parameters above.
- dsn - Device Serial Number (DSN) of the device for which to generate per-device UEK1/2/3-bit stream files.

This command generates and exports specified format bit stream file(s). The bit stream file is generated based on the information provided in the Programming Data entry.

8.5. Programming Job (Ask a Question)

```
new_prog_job -job_name <name>
             -job_type <job_type | SDebug>
             [-setup < hw_setup_type>]
```

- `name` - Name of the programming job entry.
- `job_type` - Type of job. One of the following:
  - IHP: Job for IHP flow.
  - FPExpress: Job for FPExpress flow.
  - SDebug: Create Debug Job (DDC file) for SmartDebug tool.
- `hw_setup_type` - Type of the hardware setup.
  - JTAG_CHAIN: JTAG chain (default parameter)

This command creates a new programming job. The programming job can be for IHP flow or for FPExpress. The job is created for a specific hardware setup type. The default setup type is JTAG chain. Job name must be unique among other job names. The job can have one or more devices and optional HSM Tasks for the HSM flow.

```
add_microsemi_prog_device   -job_name <job>
                              -device_name <device>
                              -device_hw_location <location>
                              [-data_name <data>]
                              [-bitstream_name <bitstream>]
                              [-dsn <dsn_value>]
                              [-bitstream_file <bitstream file>]
```

- `job` - Name of the programming job to add device.
- `device` - User name of the device. Must be unique within the Job.
- `location` - Hardware location of the device in the setup: Chain index for JTAG chain.
- `data` - Name of the Programming Data containing bit stream file.
- `bit stream` - Bitstream name in the Programming Data.
- `dsn` - Parameter to set the DSN of the device. This is used for generating per-device UEK1/2/3 bit stream files.
- `bitstream_file` - Path of STAPL file to program this device. This is useful for adding device in chain.

This command manually adds a Microchip device targeted for programming by a bit stream generated from the specific design (Programming Data) or STAPL file. Device name must be unique among other devices inside specified jobs.

Note: You can add any valid Microchip device using a stand-alone STAPL file. However, only SmartFusion 2 and IGLOO 2 are supported when using the Programming Data option.

```
add_microsemi_bypass_device   -job_name <job>
                                -device_name <device>
                                -device_hw_location <location>
                                [-device_type <die_name>]
                                [-bitstream_file <bitstream_file>]
```

- `job` - Name of the programming job to add device.
- `device` - User name of the device. Must be unique within the Job.
- `location` - Hardware location of the device in the setup: Chain index for JTAG chain.
- `die_name` - Name of Microchip device (for example, M2S010, M2GL090TS, and so on).
- `bitstream_file` - STAPL file path for this device.

This command adds a Microchip bypass device by either specifying the die name or by specifying a STAPL file.

```
add_non_microsemi_bypass_device   -job_name <job>
                                    -device_name <device>
                                    -device_hw_location <location> [-ir <IR_len>]
                                    [- tck <tck>]
                                    [-file <bsdl_file>]
```

- `job` - Name of the programming job to add device.
- `device` - User name of the device. Must be unique within the Job.
- `location` - Hardware location of the device in the setup: Chain index for JTAG chain.
- `ir` - IR length.
- `tck` - Max TCK frequency (in MHz).
- `bsdl_file` - BSDL file path for targeted non-Microchip device.

This command adds a non-Microchip bypass device by either specifying IR length and TCK frequency or by specifying BSDL file.

```
export_prog_job -job_name <job>
                -location <path>
                -name <file>
```

- `job` - Name of the programming job being exported.
- `path` - Path to the directory in which the job file will be exported.
- `name` - File name of the job file to be exported.

This command exports a non-HSM job for FlashPro Express or IHP.

Note: An HSM job is exported using the `export_hsmtask` command.

```
import_job_status -job_status_file <path>
```

- `path` - Path to the job status container generated by FlashPro Express.

HSM flow only: Imports and validates job status received from FlashPro Express or IHP. Status can be generated in the process of job execution to provide current job status, or as a result of job end, in which case, it includes cryptographically protected proof of job removal from the HSM.

8.5.1. HSM Task (Ask a Question)

```
add_hsmtask_to_job   -job_name <job>
                       -hsmtask_name <task>
                       [-m_request_type {INTERNAL|EXTERNAL}]
```

- `job` - Name of the programming (or debug) job for which task is created.
- `task` - Name of the HSM Task. Must be unique within the job.
- `m_request_type` - Specifies how the M-HSM request is executed.
  - INTERNAL - This mode can only be specified if the same physical User HSM is used to generate and execute the programming job. In this case, the Job Manager will internally execute the request.
  - EXTERNAL - Default mode requires user to export the request, process it using FlashPro Express, and then import it back into the Job Manager project.

This command creates a new HSM Task for the specified programming job. The HSM task can contain one or more job tickets. This HSM task is then used to send a job request to the M-HSM and the job response received is imported into the HSM Task, which then enables HSM task export.

```
new_hsmtask_ticket   -job_name <job>
                      -hsmtask_name <task>
                      -ticket_name <ticket>
                      -device <device>
                      -actions <programming action | DEBUG>
                      -max_device <max>
```

- `job` - Name of the programming job which adds a new ticket.
- `task` - Name of the task within the Job.
- `ticket` - Name of the new ticket. Must be unique within the task.
- `device` - Name of the target device in the Job for the new ticket.
- `action` - Programming action for the ticket.
  - DEBUG: SmartDebug action for the ticket.
- `max` - Overbuild protection: max devices to use this ticket. Can be 'unlimited'.

This command creates a new job ticket for the HSM Task. The HSM task can have one or more job tickets for each device, but each job ticket is created per programming action. The overbuild protection parameter `max_device` is applicable to the protocols that are capable of controlling the number of authorized devices, such as the Authorization Code and Unique Key protocols.

```
hsmtask_m_request   -job_name <job>
                     -hsmtask_name <task>
                     -request_file <path>
```

- `job` - Name of the programming job that contains target HSM Task.
- `task` - Name of the HSM task for which CM request is being generated.
- `path` - Full file name of the request container.

This command creates a job request that is required to perform a handshake protocol with the M-HSM (FlashPro Express or IHP). Once FlashPro Express has processed the request, it generates and exports a Job Reply that must be imported into the HSM Task on the Job Manager side. This handshake protocol guarantees one time use of the HSM Task on the FlashPro Express or IHP side (M-HSM), thus preventing job replication. This command is only applicable if the HSM task was created with the request set to EXTERNAL value (default). See `add_hsm_task` for details.

```
hsmtask_m_reply   -job_name <job>
                   -hsmtask_name <task>
                   -reply_file <path>
```

- `job` - Name of the programming job that owns target HSM Task.
- `task` - Name of the task in the job.
- `path` - Full file name to the container with Job Response.

Import job reply by FlashPro Express or IHP (M-HSM). This command is only applicable if the HSM task was created with the request set to EXTERNAL value (default). See `add_hsm_task` for details. For more information, see the `process_job_request` FlashPro Express command documented in the Tcl Commands Reference Guide.

```
export_hsmtask   -job_name <job>
                  -hsmtask_name <task>
                  -location <path>
                  -name <file_name>
```

- `job` - Name of the programming job that contains task being exported.
- `task` - Name of the task in the job.
- `path` - Location of the export container.
- `file_name` - File name for the export container.

This command exports the HSM job for further execution by FlashPro Express or IHP. This can only be executed after importing job reply.

8.5.2. SPI Directory (Ask a Question)

```
export_spi_directory   -golden_ver <value or SPI file>
                           -golden_addr <hex value>
                           -update_ver <value or SPI file>
                           -update_addr <hex value>
                           -file <file_name>
```

- `golden_ver <value or SPI file>` - Specifies Golden SPI Image design version. There are two ways to specify the value:
  - Decimal value less than 65536 (exclusive)
  - SPI file from which the design version is read.
- `-golden_addr <hex value>` - Specifies Golden SPI Image address where hex is 32-bit hexadecimal value with prefix 0x/0X.
- `-update_ver <value or SPI file>` - Specifies Update SPI Image design version. There are two ways to specify the value:
  - Decimal value less than 65536 (exclusive)
  - SPI file from which the design version is read.
- `-update_addr <hex value>` - Specifies Update SPI Image address where hex is a 32-bit hexadecimal value with prefix 0x/0X.
- `-file <file>` - Mandatory argument; specifies the file export location.

Supported Families

- SmartFusion 2
- IGLOO 2

Both golden* options go together. The same is true for both update* options; the file argument is required:

```
export_spi_directory \
-golden_ver
{D:\flashpro_files\m2s025_jb_spi_dir\designer\a1_MSS\export\a1_MSS.spi} \
-golden_addr {0xa} \
-file {D:\flashpro_files\jobmgr_project12\dev.spidir}

export_spi_directory \
-update_ver {456} \
-update_addr {0xdef} \
-file {D:\flashpro_files\jobmgr_project12\dev.spidir}

export_spi_directory \
-golden_ver {123} \
-golden_addr {0xabc} \
-update_ver {456} \
-update_addr {0xdef} \
-file {D:\flashpro_files\jobmgr_project12\dev.spidir}
```

9. Referenced Documents (Ask a Question)

This user guide references the following documents:

- Secure Production Programming Solution (SPPS) User Guide
- Libero SoC Design Flow User Guide
- FlashPro Express User Guide
- User HSM Installation and Setup User Guide
- Manufacturer HSM Installation and Setup User Guide

10. Revision History (Ask a Question)

The revision history describes the changes that were implemented in the document. The changes are listed by revision, starting with the most current publication.

Revision	Date	Description
N	05/2025	This document is released with Libero SoC Design Suite v2025.1 without changes from v2024.2.
M	11/2024	Added a note in the Export Programming Bitstream File section.
L	08/2024	This document is released with Libero SoC Design Suite v2024.2 without changes from v2024.1.
K	02/2024	This document is released with Libero SoC Design Suite v2024.1 without changes from v2023.2.
J	08/2023	This document is released with Libero SoC Design Suite v2023.2 without changes from v2023.1.
H	04/2023	This document is released with Libero SoC Design Suite v2023.1 without changes from v2022.3.
G	12/2022	The following changes are made in this revision: • Updated the HSM-based Manufacturing Flow section. • Updated the HSM Programming or Debug Job section. • Updated the Creating a FlashPro Express or SmartDebug Job section. • Updated the Update Bitstream Job Flow section. • Updated the Bitstream Management section. • Updated the Programming Job section. • Updated the HSM Task section.
F	08/2022	The following changes are made in this revision: • Updated the Design Import section. • Updated the Security Modifications section. • Updated the eNVM Client Modifications section. • Updated the sNVM Client Modifications section. • Updated the Bitstream Management section.
E	04/2022	This document is released with Libero SoC Design Suite v2022.1 without changes from v2021.3.
D	12/2021	This document is released with Libero SoC Design Suite v2021.3 without changes from v2021.2.
C	08/2021	Key Rotation: Added a new section related to key rotation. Security Modifications: Added a new Tcl command `set_new_keys_for_rotation`. Bitstream Management: Added KEY_ROTATION bit stream type for `init_bitstream` Tcl command.
B	04/2021	Editorial updates only. No technical content updates.
A	11/2020	Document converted to Microchip template. Initial revision.

Microchip FPGA Support

Microchip FPGA products group backs its products with various support services, including Customer Service, Customer Technical Support Center, a website, and worldwide sales offices. Customers are suggested to visit Microchip online resources prior to contacting support as it is very likely that their queries have been already answered.

Contact Technical Support Center through the website at www.microchip.com/support. Mention the FPGA Device Part number, select appropriate case category, and upload design files while creating a technical support case.

Contact Customer Service for non-technical product support, such as product pricing, product upgrades, update information, order status, and authorization.

- From North America, call **800.262.1060**
- From the rest of the world, call **650.318.4460**
- Fax, from anywhere in the world, **650.318.8044**

Microchip Information

Trademarks

The “Microchip” name and logo, the “M” logo, and other names, logos, and brands are registered and unregistered trademarks of Microchip Technology Incorporated or its affiliates and/or subsidiaries in the United States and/or other countries (“Microchip Trademarks”). Information regarding Microchip Trademarks can be found at https://www.microchip.com/en-us/about/legal-information/microchip-trademarks.

ISBN: 979-8-3371-1094-3

Legal Notice

This publication and the information herein may be used only with Microchip products, including to design, test, and integrate Microchip products with your application. Use of this information in any other manner violates these terms. Information regarding device applications is provided only for your convenience and may be superseded by updates. It is your responsibility to ensure that your application meets with your specifications. Contact your local Microchip sales office for additional support or, obtain additional support at www.microchip.com/en-us/support/design-help/client-support-services.

THIS INFORMATION IS PROVIDED BY MICROCHIP “AS IS”. MICROCHIP MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND WHETHER EXPRESS OR IMPLIED, WRITTEN OR ORAL, STATUTORY OR OTHERWISE, RELATED TO THE INFORMATION INCLUDING BUT NOT LIMITED TO ANY IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A PARTICULAR PURPOSE, OR WARRANTIES RELATED TO ITS CONDITION, QUALITY, OR PERFORMANCE.

IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INDIRECT, SPECIAL, PUNITIVE, INCIDENTAL, OR CONSEQUENTIAL LOSS, DAMAGE, COST, OR EXPENSE OF ANY KIND WHATSOEVER RELATED TO THE INFORMATION OR ITS USE, HOWEVER CAUSED, EVEN IF MICROCHIP HAS BEEN ADVISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE. TO THE FULLEST EXTENT ALLOWED BY LAW, MICROCHIP’S TOTAL LIABILITY ON ALL CLAIMS IN ANY WAY RELATED TO THE INFORMATION OR ITS USE WILL NOT EXCEED THE AMOUNT OF FEES, IF ANY, THAT YOU HAVE PAID DIRECTLY TO MICROCHIP FOR THE INFORMATION.

Use of Microchip devices in life support and/or safety applications is entirely at the buyer’s risk, and the buyer agrees to defend, indemnify and hold harmless Microchip from any and all damages, claims, suits, or expenses resulting from such use. No licenses are conveyed, implicitly or otherwise, under any Microchip intellectual property rights unless otherwise stated.

Microchip Devices Code Protection Feature

Note the following details of the code protection feature on Microchip products:

- Microchip products meet the specifications contained in their particular Microchip Data Sheet.
- Microchip believes that its family of products is secure when used in the intended manner, within operating specifications, and under normal conditions.
- Microchip values and aggressively protects its intellectual property rights. Attempts to breach the code protection features of Microchip products are strictly prohibited and may violate the Digital Millennium Copyright Act.
- Neither Microchip nor any other semiconductor manufacturer can guarantee the security of its code. Code protection does not mean that we are guaranteeing the product is “unbreakable”. Code protection is constantly evolving. Microchip is committed to continuously improving the code protection features of our products.