SIC Marking Laser Connector

Laser marking systems driven through the CCI ASCII interface

- Name: **SIC**
- Version: 1.0.1.1
- Protocol: CCI (ASCII command/reply)
- Interface: TCP/IP
- Runtime: .NET Standard 2.0 (Multiplatform)
- Configuration:
  - Devices / Channels / Nodes / Points

Overview
Channel Configuration
Node Configuration
Point Configuration
Selection Points
Global Values
Entity Data
Bit Points
Derived Points
Commands
Laser Card Telnet Interface
Status Points
Adding a New Laser
Notes and Limitations
Driver Revision History

Overview

The SIC connector controls laser marking systems through its **CCI** ASCII interface. Every exchange is a single text command terminated by a line feed, answered by a single line: `1` for accepted, `0` for rejected, or the value itself for a read.

The connector covers the full marking workflow — loading and saving jobs, pen and laser parameters, entity geometry, text and barcode content, counters and date/time objects, hatching and wobble, and firing the mark — so a marking station can be built entirely from device points, with no scripting required to reach the laser.

A rejected command is an *application-level* answer, not a transport fault: an unknown entity name, a job path that does not exist, or a barcode setting that is invalid for the current symbology all return `0` while the connection stays healthy. Those surface as Bad quality on the point.

Communication Driver Information	
Driver name	SIC
Assembly Name	T.ProtocolDriver.SIC
Assembly Version	1.0.1.1
Multiplatform	True
Devices supported	Laser marking systems controlled with the CCI ASCII interface enabled
Manufacturer	SIC Marking
Protocol	CCI (ASCII command/reply)
Interface	TCP/IP

Channel Configuration

Simultaneous Requests

Set the channel to a **single simultaneous request** — both `SimultaneousRequestGlobal` and `SimultaneousRequestEachNode` set to `1`. This is a requirement, not a tuning preference.

The laser software serves one client on one port. If the channel is allowed to run requests in parallel, the platform creates additional channel instances and each opens its own connection — most of which fail to connect. The selection state is also held per instance, so an entity chosen on one instance is invisible to the others and entity-scoped points silently act on the wrong scope. Sequences that must stay together, such as selecting a pen and then reading a pen value, can interleave between instances and return a correct-looking value from the wrong pen.

TCP/IP Settings

The CCI interface listens on TCP port **350** by default. The connection is opened once and reused — the connector does not reconnect per command.

Node Configuration

Station Configuration

The station string is semicolon-separated: `IP;Port`

- **IP**: address of the machine running the CCI interface. Use `127.0.0.1` when it runs on the same machine as the runtime.
- **Port**: TCP port of the interface. Default `350`. Additional lasers run on different ports.

For example, `127.0.0.1;350` for a local laser, or `127.0.0.1;351` for a second one.

A node that reads the laser card through the `FLASH` family points at the card itself, on its Telnet port: for example `192.168.1.2;23`. Give that node a channel of its own, so a card that is unplugged or rebooting never delays the polls of the laser software.

The pen is **not** a station field. It is a point — see Required Points below.

Point Configuration

Address Format

Every address is exactly two tokens: **`Family:Item`**

- **Family**: selects both the wire call and the value type — see Address Families below.
- **Item**: a friendly name, or a raw numeric id.

The **target entity and the target pen are not part of the address**. They come from two required points whose values the connector applies to every entity-scoped and pen-scoped operation. One address therefore serves every entity in the job, and switching the target is a single write rather than a reconfiguration.

Required Points

Every node must configure these two. Without them the node acts on the whole job and on pen 1 for its entire lifetime.

Address	Access	Purpose
`SEL:Entity`	Read /Write	Name of the entity every entity-scoped point acts on. Write `All` (or an empty string) to target the whole job. Read it back to confirm what the connector adopted.
`SEL:Pen`	Read /Write	Pen every pen-scoped point acts on. **`0` selects AUTO**, which uses the pen the job itself assigned to the selected entity. `1`–`255` pins a specific pen.

Two further selections are required only on nodes that use flash storage. They follow the same rule: the slot and the file path are not part of a flash address, so the commands resolve through these points.

Address	Access	Required for
`SEL:FlashSlot`	Read /Write	All four flash commands. Defaults to slot `1`, so a node that never writes this point still resolves to a real slot. Write a negative value to disarm the commands — every one of them refuses a negative slot.
`SEL:FlashPath`	Read /Write	`CMD:FlashStoreFile` and `CMD:FlashSaveToFile` only. The two commands that move a job between the laser software and the card ignore it. Defaults to `C:\SIC_MARKING\Slot1.unf`, naming the same slot as the default above. The path is passed through unchanged — no folder is created and no extension is appended.

Write `SEL:Entity` *before* the points that depend on it, and not in the same scan. In a typical screen this happens naturally: the entity selector writes on change, and the action buttons write later.

Address Families

Family	Scope	Type	Access	Description
`SEL`	Connector	—	R / R+W	Selection state. Answered by the connector rather than by the laser.
`DV`	Global	Double	R+W	Global floating-point value.
`LV`	Global	Long	R+W	Global integer value.
`SV`	Global	String	R+W	Global string value.
`ED`	Entity	Double	R+W	Floating-point data on the selected entity.
`EL`	Entity	Long	R+W	Integer data on the selected entity.
`ES`	Entity	String	R+W	String data on the selected entity.
`BIT`	Varies	Boolean	R / R+W	One checkbox inside a flag word. The connector reads the word, changes the single bit and writes it back, so the neighbouring settings survive.

CALC	Varies	Varies	R / R+W	A value the connector derives from one or more reads. A derivation that cannot be completed reports Bad quality rather than a stale number.
CMD	—	—	W	An action. The point's value carries the argument.
STS	—	Integer	R	Connector status.
FLASH	Laser card	String	R	Data the laser card reports over its own Telnet port. Only on a node pointed at the card, never on the node of the laser software. See Flash Job List.

The same number means different things in different families — identifier 9 is one value as a global integer, another as global string, another as entity data. The family is what disambiguates it, which is why every address carries one.

Raw Identifier Addresses

On DV, LV, SV, ED, EL and ES, the Item may be a raw numeric identifier instead of a name — for example ED:71. This reaches identifiers that have no friendly name yet.

Unit conversion is keyed to the *name*, never to the identifier. DV:Frequency converts kHz to Hz; DV:6 does not. A raw address is passed through exactly as written.

Selection Points

Address	Type	Access	Description
SEL:Entity	String	R+W	**Required.** Target entity name; All means the whole job.
SEL:Pen	Integer	R+W	**Required.** Target pen; 0 = AUTO, 1–255 = fixed.
SEL:EntityList	String	R	Top-level named entities in the job, semicolon-separated and prefixed with All. Feeds an entity selector. Entities nested inside a group are not listed.
SEL:EntityType	String	R	Type of the selected entity — see Entity Types below. Empty when the whole job is selected.
SEL:FlashSlot	Integer	R+W	Target slot for every flash command. Defaults to 1; a negative value disarms them.
SEL:FlashPath	String	R+W	PC-side file path for the two file-mode flash commands. Defaults to C:\SIC_MARKING\Slot1.unf.
SEL:SaveAsPath	String	R+W	Destination for CMD:SaveJobAs. Defaults to C:\SIC_MARKING\LASER\jobfiles\NewJob.sjf.
SEL:EntityValues	String	R	What every named top-level entity in the job marks, as Name=Value pairs separated by semicolons, for example Serial=ABC123;Date=2026-10-16. Entities with no content, such as logos, groups and shapes, are left out. Each read costs about two exchanges per entity and is not cached, so put it on a slow scan or read it on demand. Read it when the mark starts: counters advance after marking, so a later read reports the next value instead of the one on the part.
SEL:ImportPath	String	R+W	Source file for CMD:ImportReplace: the full path on the machine running the laser software. The file type comes from the extension: raster png, jpg, bmp, gif, tif, tga, pcx, or vector svg, dxf, plt, ai. Defaults to C:\SIC_MARKING\Image.dxf. An empty value makes the command refuse.

Entity Types

SEL:EntityType reports the object class, which is what decides *which editing screens apply*.

Value	Object
ScWinTextChars2D	Text
ScBarCode12Chars2D	Barcode, any symbology
ScSerialNumber2D	Counter or date/time
ScRectangle2D, ScEllipse2D, ScPolyLine2D, ScSpiral2D	Geometry
ScEntities2D	Group
ScMotionControl, ScExecutable	Control objects
(empty)	Whole job selected, or the name is not in the job

Two distinctions the type does **not** make. It does not separate DataMatrix from QR from 1D — read ES:BarcodeType for that. And a counter and a date/time object are the same class; BIT:IsDateTime is what tells them apart. Treat the list as non-exhaustive and compare case-insensitively.

Global Values

DV — Global Double

Items marked **pen** act on `SEL:Pen`.

Item	Unit	Scope	Description
`LaserPower`	W	pen	Laser power of the selected pen.
`MarkSpeed`	mm/s	pen	Marking speed.
`JumpSpeed`	mm/s	pen	Jump speed between marked segments.
`Frequency`	kHz	pen	Pulse frequency. The interface works in Hz; the connector converts.
`MarkDelay`	µs	pen	Delay after a marked segment.
`JumpDelay`	µs	pen	Delay after a jump.
`LaserOnDelay`	µs	pen	Delay before the beam switches on.
`LaserOffDelay`	µs	pen	Delay before the beam switches off.
`PolygonDelay`	µs	pen	Delay at a corner between segments.
`WobbleFrequency`	Hz	pen	Wobble frequency.
`WobbleAmplitude`	mm	pen	Wobble amplitude.
`OverridePower`	%	global	A global multiplier applied on top of the pen power. Not the pen power itself.
`OpticOffsetX`	mm	optic	Horizontal offset of the marking field.
`OpticOffsetY`	mm	optic	Vertical offset of the marking field.
`OpticGainX`	factor	optic	Horizontal scale correction. Dimensionless: `1.0` marks at nominal size.
`OpticGainY`	factor	optic	Vertical scale correction.
`OpticRotation`	degrees	optic	Rotation of the whole marking field, absolute.
`FieldMinX` `FieldMinY` `FieldMaxX` `FieldMaxY`	mm	optic	**Read only.** Edges of the optic's physical marking field, fixed by the lens and the correction file. The scanner cannot deflect past them. The interface sets field size only through a combined call, so writing these has no effect.
`WorkAreaMinX` `WorkAreaMinY` `WorkAreaMaxX` `WorkAreaMaxY`	mm	optic	Edges of the usable work area — a software rectangle inside the optic field, normally set smaller to stay clear of a fixture or of the distorted outer field. Writable.

The two limits are different things and a position can be inside one and outside the other, so check both when a job will not go where it was told. Position is the *centre* of the mark, so the usable range on an axis is the work-area edge minus half the mark size.

Optic values are machine settings, not job contents. They apply to every job this head marks, they leave every entity untouched, and `CMD:SaveJob` does **not** persist them.

LV — Global Long

Item	Description
`PointerMode`	Mode of the guide pointer.
`PenEnableFlags`	Raw pen enable flag word. Pen-scoped. Use `BIT:WobbleEnable` to change it.
`TransformAction`	`0` stores the transform baseline, `1` restores it, `2` restores with update.
`TotalEntityNum`	Entity count, all levels, including entities nested in groups.
`ToplevelEntityNum`	Entity count, top level only — a group counts once.
`ExecutionStatus`	Stop and abort flags: `0x1` external stop, `0x2` stopped from the laser software, `0x4` waiting for a trigger. `0x0` means nothing was stopped. Reset at each new mark.

`MarkResult`	Outcome of the *last* mark: `0` failure, `1` success, `2` external stop, `3` mark-dialog stop, `4` trigger-window stop, `5` stopped by the interface, `6` stepper controller, `7` input, `8` emergency stop.

Neither of these reports whether a mark is in progress — they say what happened, not what is happening. Use `STS:IsMarking` for the live state.

SV — Global String

Item	Description
`JobFileName`	Read-only. Path of the job currently open in the laser software. Read it after a save to confirm which file the software is pointing at.
`JobImagePath`	Writing a path captures an image of the job. `CMD:SaveJobImage` is the reliable form — a full-resolution capture needs the window visible, which that command handles.

Entity Data

All of these act on the entity named by `SEL:Entity`.

ED — Entity Double

Item	Unit	Description
`FontSize`	mm	Character height. **Millimetres, not points.** Ignored while a height limit is active.
`TextLengthLimit`	mm	Width the text is scaled to fit. `CALC:EntityWidth` is the preferred form.
`TextHeightLimit`	mm	Height the text is scaled to fit. `CALC:EntityHeight` is the preferred form.
`TextGap`	mm	Gap between a barcode and its human-readable text.
`HatchDistance`	mm	Distance between hatch lines. Valid range `0.0001`–`1000`.
`HatchAngle`	degrees	Hatch line angle. The interface works in radians; the connector converts.
`LineReduction`	%	Line reduction applied to barcode modules.
`CellSizeX`	ratio	Horizontal cell fill ratio, `0`–`1`. Honoured only while `BIT:CellMode` is on.
`CellSizeY`	ratio	Vertical cell fill ratio.
`QuietZoneX`	ratio or mm	Horizontal quiet zone. A scale factor by default; millimetres once `BIT:Invert` has set the absolute flag.
`QuietZoneY`	ratio or mm	Vertical quiet zone. Follows X unless the absolute flag is set.
`RotationRelative`	degrees	Rotates the entity *by* this amount. Accumulates on repeated writes — use `CALC:EntityRotation` for an absolute angle.
`RotationAbsolute`	degrees	Absolute angle. Not implemented on every build; `CALC:EntityRotation` handles the fallback.

EL — Entity Long

Item	Description
`HatchStyle`	`0` off, `1` wavy, `2` horizontal LR, `3` horizontal RL, `4` rotational, `5` wavy with jumps, `6` zigzag. Prefer `CMD:ApplyHatch`.
`Passes`	Mark loop count.
`TextCharFlags`	Raw text checkbox word. Use the `BIT:TextLimit*` points to change it.
`EntityPen`	Pen the job assigned to this entity, `1`–`255`. This is what `SEL:Pen = 0` reads.
`HumanReadable`	Human-readable text on or off.
`BarcodeEcc`	QR error correction: `0` = L, `1` = M, `2` = Q, `3` = H. DataMatrix is fixed and ignores it.
`BarcodeFlags`	General barcode flag word.
`BarcodeLimitFlags`	`0x1` limit length, `0x2` limit height, `0x4` keep aspect.

`DmSymbolMode`	DataMatrix symbol-mode word.
`DmSymbolSize`	Raw zero-based symbol-size index. `CALC:SymbolSize` is the readable form.
`QrCodeExVersion`	QR version — the QR counterpart of the DataMatrix symbol size.
`SerialStart`	Counter start value. Takes effect only after a reset.
`SerialStep`	Counter increment.
`SerialCurrent`	Raw current counter value. See the caveat on `CALC:SerialCurrent`.
`SerialBeatCount`	Advance the counter only every N marks.
`SerialResetCount`	Auto-reset after N. `0` never resets.
`SerialModeFlags`	Raw mode word: text `0x1`, barcode `0x2`, file `0x4`, date/time `0x8`, custom format `0x10`.

ES — Entity String

Item	Description
`Content`	The string a text object marks, or the payload a barcode encodes. Reads back as well as writes. On a counter or date/time object this is regenerated from the format at the next mark — change the format instead.
`BarcodeType`	Symbology, e.g. `DataMatrixEx`, `QR Code Ex`, `Code-128`, `EAN-13`, `Code-39`. Set it *first* — it decides which other barcode settings are valid.
`BarcodeFormat`	Format descriptor, e.g. `2,2,2,2`.
`SerialFormat`	Counter format string, e.g. `LOT-%06.0f-A`. The conversion must be `f`; an integer specifier is rejected. Shared with the date/time pattern.
`SerialAsciiFile`	Path to a serial source file. The file must already exist. Requires `BIT:SerialFromFile`.

Bit Points

Each of these is one checkbox inside a flag word that also carries unrelated settings. The connector reads the word, changes the single bit and writes it back, so nothing else is disturbed.

Item	Access	Description
`WobbleEnable`	R+W	Wobble on or off for the selected pen. The wobble *shape* is configured per pen in the laser software and is not reachable over the interface.
`Invert`	R+W	Barcode invert. Also switches the quiet zone to absolute millimetres and releases the vertical axis, because the two settings travel together.
`CellMode`	R+W	Cell mode, required for flash operation. Set it *before* the cell size.
`DmAutoSize`	R+W	DataMatrix automatic symbol size.
`DmRectangle`	R+W	DataMatrix rectangular sizes. Switches the symbol-size list to the rectangular set.
`TextLimitLength`	R+W	Force the text to a fixed width.
`TextLimitHeight`	R+W	Force the text to a fixed height.
`TextLimitAspect`	R+W	Keep the aspect ratio while forcing a limit.
`SerialCustomFormat`	R+W	Use the custom format string. This is the only route to zero-padding — turning it off disables prefix, suffix and padding together.
`SerialFromFile`	R+W	Take the serial from a file. Mutually exclusive with the custom format.
`IsDateTime`	**R**	Whether the selected object is a date/time rather than a counter. Read-only: it reports what the object *is*. This is the only reliable way to tell the two apart.

Derived Points

These present a value the interface does not expose directly. Each costs several exchanges, so they belong on a moderate scan rate rather than the fastest one.

Item	Type	Access	Description
`MarkWidth`	Real	R+W	Width of the whole mark in mm. Reading returns the job's extent scaled by the optic gain; writing derives the gain from the wanted size, so no entity is modified and read-then-write changes nothing.
`MarkHeight`	Real	R+W	Height of the whole mark in mm, by the same mechanism.
`EntityCenterX`	Real	R	Horizontal centre of the selected entity in mm, derived from its bounding box.
`EntityCenterY`	Real	R	Vertical centre in mm. Write both axes together with `CMD:ApplyEntityPosition`.
`EntityWidth`	Real	R+W	Entity width in mm. Reading prefers the configured size limit and falls back to the bounding box when there is none. On text and barcodes, writing sets the limit and its checkbox together. An entity without size limits, such as an imported logo, a group or a bitmap, is resized through its outline instead: only this axis changes and the centre stays in place, so the artwork can be distorted.
`EntityHeight`	Real	R+W	Entity height in mm, by the same rule. A non-zero height overrides `ED:FontSize`.
`EntityRotation`	Real	R+W	Entity angle in degrees, **absolute**, counterclockwise positive. Writing reads the current angle and sends the difference, so writing the same value twice does not rotate twice.
`SerialPrefix`	String	R	Text before the counter digits, extracted from the format string.
`SerialSuffix`	String	R	Text after the counter digits.
`SerialMinDigits`	Integer	R	Zero-padding width. Prefix, suffix and padding have no identifiers of their own — they exist only inside the format string. Write all three with `CMD:ApplyCounter`.
`SerialCurrent`	Integer	R	Current counter value, recovered from the rendered content. The direct value has been observed reporting `0` for a counter that was not at zero, and a read carries no status to distinguish the two.
`DateTimeFormat`	String	R+W	Date/time pattern in readable form, e.g. `DD/MM/YYYY` — the connector converts to and from the internal representation. Refuses a counter object in both directions, because they share the same format field.
`SymbolSize`	String	R+W	DataMatrix symbol size as text, e.g. `20x20` or `Auto`. Square sizes only; with rectangular sizes selected the read reports Bad quality rather than a wrong size.
`HatchOnOff`	Integer	R	Whether hatching is active. Turn it on and off with `CMD:ApplyHatch`.

Commands

A command point is written, not polled. Where a command takes an argument, it is the point's own value.

Marking and Job Control

Item	Value	Description
`Mark`	—	Marks the whole job. Returns immediately without blocking; poll `STS:IsMarking` for completion.
`MarkSelected`	—	Marks only the selected entity. Refused when the whole job is selected.
`StopMarking`	—	Aborts the current mark.
`LoadJob`	File path	Loads a job. Marking and the pointer are stopped first, then a clean transform baseline is stored.
`SaveJob`	—	Saves over the job currently open, carrying both entity and pen changes.

`SaveJobAs`	—	Saves the job to the path held in `SEL:SaveAsPath`, carrying both entity and pen changes. The destination is not written into the command, so a plain trigger saves to the same path as often as the operator presses it. Refused when `SEL:SaveAsPath` is empty.
`SaveJobImage`	File path	Captures an image of the job. Reports the result of the capture itself.
`RenameEntity`	New name	Renames the selected entity and re-points the selection at the new name.
`ImportReplace`	Any non-zero value	Replaces the artwork of the selected entity with the file in `SEL:ImportPath`, keeping its name, its place in the job and, for vector files, its pen. The new artwork is then fitted inside the box of the old one: as large as fits, proportions kept, centred. The entity must already exist and be of the matching kind, a bitmap placeholder for raster files or a vector one for vector files, so create it once by importing a logo by hand in the laser software. Refused when the whole job is selected.
`PointerStart`	—	Starts the guide pointer.
`PointerStop`	—	Stops the guide pointer.
`ShowApp`	—	Shows the laser software window.
`HideApp`	—	Hides the window. The interface keeps working.
`TransformStore`	—	Stores the current transform of every entity so it can be restored later.
`TransformRestore`	—	Restores the stored transforms.

Applied Settings

These write several values in a fixed order. They exist as single commands precisely *because* the order matters — separate points give no ordering guarantee.

Item	Value format	Example	Description
`ApplyMarkSize`	`width,height`	`50,63.5`	Whole-mark size in mm, both axes in one exchange so the mark is never left scaled on one axis only.
`ApplyEntityPosition`	`x,y`	`12.5,-4`	Absolute centre in mm. With one entity selected it moves that entity; with the whole job selected it translates the job as a single block.
`ApplyHatch`	`on,distance,angle`	`1,0.05,45`	Distance and angle are written before the style. Enabling hatching while the stored distance is invalid produces a dialog on the equipment, so the range is checked before anything is sent.
`ApplyTextLimits`	`width,height`	`20,6`	Size limits in mm, written before the checkboxes that activate them. `0` means no limit on that axis.
`ApplyCounter`	`start, minDigits, prefix, suffix`	`1,6,LOT-,-A`	Sets the counter and resets it so the new values take effect. The reset is job-wide. Refuses a date/time object.
`ApplyBarcode`	`key=value; key=value`	`sym=DataMatrixEx;cell=0.8; invert=1;size=20x20; content=ABC123;rot=90`	Applies barcode settings in the required order. Keys not present are left alone. See below.

ApplyBarcode keys

Recognised keys, listed in the order they are applied: `sym`, `cellmode`, `invert`, `cell`, `qz`, `lr`, `size`, `ecc`, `content`, `hr`, `gap`, `font`, `w`, `h`, `rot`.

The order is not arbitrary. The symbology decides which other settings are valid, so it goes first. The cell mode must precede the cell size. The content goes late so the symbol is encoded against the final constraints. The rotation goes last, because any step that changes the symbol rebuilds its geometry and would discard an angle set earlier. The quiet zone is written only when `invert=1`, since it has no effect otherwise.

Counters

Item	Description
`ResetSerial`	Restarts every counter in the job, not only the selected one.
`IncSerial`	Increments every counter in the job.
`DecSerial`	Decrements every counter in the job.

Flash Memory

Job storage on the controller card, for standalone operation. All four commands take their slot from `SEL:FlashSlot` and, where a file is involved, their path from `SEL:FlashPath` — the values are **not** written into the command point.

Item	Direction	Description
`FlashStoreCurrent`	job slot	Loaded job into a flash slot. The job must already be saved on disk, so pair it with `CMD:SaveJob`.
`FlashLoadToSICAdvanced`	slot job	Flash slot back into the laser software. Replaces the loaded job; card settings are not restored.
`FlashStoreFile`	file slot	A file on disk into a flash slot.
`FlashSaveToFile`	slot file	A flash slot out to a file on disk.

Because they carry no value, these are plain triggers: write any non-zero value to fire. A timer tag with a short delay-off is the natural shape — the press fires the command and the automatic reset to `0` does not fire it again. This is also what lets the operator repeat the same slot: a command that carried the slot in its own value could not be re-fired, because the second press would write the value the tag already held.

Flash access is slow relative to every other command, and the connector raises the channel timeout for the duration.

The CCI does not list which job is stored in each slot. The laser card reports that over its own Telnet interface; see **Laser Card Telnet Interface** below.

Laser Card Telnet Interface

The connector reaches the laser through the CCI of the laser software, on the port in the station string (`350` by default). The laser card also has its own ASCII command interface, reached over Telnet on the card's network port. Some information is answered only there. The main example is the flash memory contents: the card's `ST` (statistics) command lists every job stored in flash, with its slot ID, job name and size.

The Telnet interface is a separate connection, to the card's own IP address on port 23, and it uses a different command set: the card answers its own short commands and rejects CCI commands as unknown. The connector reads the flash list through the `FLASH:Jobs` point; see Flash Job List below.

Telnet mode is **disabled** on the card by default. While it is disabled, the card still answers ping on its network port, but every command times out. It is enabled once, over USB, and stays enabled.

Enable Telnet Mode

Connect the laser card to the PC with a USB cable.

1. Close the laser software.
2. In the SIC Laser Advanced installation folder, open the `System` folder and create a desktop shortcut to the **USC Server** executable.
3. In the shortcut's **Properties**, add `/v` at the end of **Target**, separated by a space, and click **Apply**. Without `/v`, USC Server runs in the background with no window.
4. Start USC Server from the shortcut and scroll up in its log. It reports that Telnet mode is disabled.
5. Click **Flash**, then **Configuration** at the top right. Check **Telnet** and click **OK** on each dialog until they close.
6. Click **Reconnect** and wait about 10 seconds. The log now reports that Telnet mode is enabled, and the line at the top shows the card's dongle number and IP address, `192.168.1.2` by default.
7. Close USC Server.

Connect over the Network

USB takes priority. While a USB cable is plugged in, the card sends all communication to it and ignores the network port, so the card must boot without USB.

1. Unplug the USB cable.
2. Connect a network cable to the Ethernet port on the laser card itself, next to its USB port. An EtherNet/IP or other fieldbus module in the controller has its own port and IP address and is not this interface.
3. Reboot the laser card with the **UC reboot** button at the top right of the controller's HMI screen. The reboot takes 10 to 15 seconds. If the card does not answer afterwards, power-cycle the controller and reset the E-stop circuit.
4. Give the PC's Ethernet adapter a static address on the card's subnet. For a card at `192.168.1.2`, this configuration is known to work: IP address `192.168.1.5` (any host number other than 2), gateway `192.168.1.250`, preferred DNS `192.168.1.250`. When working on the PC remotely, keep the remote session on a different adapter, or the change cuts it off.
5. Check the link with `ping 192.168.1.2`.

The laser software reaches the card over the network as well, so the CCI side keeps working after the USB cable is removed.

Send Commands

- Open a TCP connection to the card's IP address on port 23. Any TCP client or terminal works for testing.
- End every command with a carriage return and a line feed (`CR LF`).
- Every reply starts with a code and a colon. `0:` means the command was accepted and executed. Codes `1:` to `20:` are errors, for example `3:` for an unknown command. The full list is in the SCAPS documentation for the USC card.
- `ST` returns the card statistics: the software version and the job stored in each flash slot, with its ID, name and size.

The card runs a subset of what the laser software can do. It has far less memory, and functions such as the mark preview are available only through the laser software. The SCAPS documentation lists what flash mode supports.

Flash Job List

`FLASH:Jobs` lists every job stored on the card's flash as `id=name` pairs separated by semicolons, lowest slot first, for example `10=retract.unf; 11=mark rev.unf`. Empty slots are left out, and an empty flash reads as an empty string. Mapped to a DataTable tag, the point arrives as a table instead, with columns `Slot` and `Job` and one row per stored job, ready to bind a DataGrid to.

Setting	Value
Node	The card's IP and port 23, for example `192.168.1.2;23`. The node carries `FLASH` points only and needs none of the required selection points.
Channel	A channel of its own, separate from the laser software's, with a single simultaneous request.
Scan	5 s or slower. One poll costs one exchange per slot up to the highest job ID.

The list is not cached, so a slot overwritten in place shows its new name on the next poll. A sweep that fails or comes back inconsistent fails the whole point instead of showing a list with gaps.

Status Points

Item	Description
`Connected`	Reads `1` while the interface answers. A lost connection appears as Bad *quality* on the point, not as a `0` — gate logic on the quality, not on the value.
`IsMarking`	Reads `1` while the laser is marking and `0` when it is idle. No configuration — the point calls the interface's own marking query. `CMD:Mark` returns immediately rather than blocking, so this is the completion signal to poll. Pair it with `LV:MarkResult` when the outcome matters as well as the state.

Adding a New Laser

A multi-laser solution keeps one tag, one channel and one set of points per laser. The displays bind to a reference tag that points at the laser selected on screen, so they need no change when a laser is added. The only exception is a display that lists every laser in its own row. The steps below add a third laser to a solution that already has `Laser1` and `Laser2`; later lasers follow the same pattern.

Laser Software and Card

The laser software serves one client on one port, so each laser needs its own CCI port. Two lasers on the same PC cannot both use 350: give the new one a port that the other lasers do not use. If the solution also reads the laser card over Telnet, the new card needs its own IP address on the card subnet, with Telnet mode enabled as described in Enable Telnet Mode.

Tag

Create a tag named `Laser3` with the same type as the existing lasers. Keep the `Laser` plus number naming: the laser selection and the shift counter reset build the tag name from the number. The type already carries every member that the points and scripts use. A value that differs from laser to laser, such as the type of the selected entity, belongs in that type. If it lives in a global tag bound to two lasers, each laser overwrites the value of the other.

Channel and Nodes

Create one channel per laser, and a second one for its card when the solution reads the card over Telnet. Each channel runs on its own thread, so a laser or a card that is switched off or slow to answer does not delay the others.

Object	Name	Configuration
Channel	`SIC_L3`	Protocol SIC, interface TCP/IP. Copy the settings and timeouts of an existing laser channel, including a single simultaneous request (see Simultaneous Requests).

Node	`SIC_L3`	Channel `SIC_L3`. Station: IP and CCI port of the new laser, for example `127.0.0.1;354`.
Channel	`SIC_L3_Flash`	Only when the solution reads the card over Telnet. Protocol SIC, interface TCP/IP, a single simultaneous request.
Node	`SIC_L3_Card`	Channel `SIC_L3_Flash`. Station: card IP and port 23, for example `192.168.1.4;23`.

Points

Every laser uses the same point list. Copy all points of an existing laser and change only the tag name and the node. Address, data type and access type stay the same.

Column	Existing laser	New laser
TagName	`Laser2.`	`Laser3.`
Node	`SIC_L2`	`SIC_L3`
Node	`SIC_L2_Card`	`SIC_L3_Card`

The quickest way is to filter the points grid by the nodes of the existing laser, copy the rows to a spreadsheet, replace the names and paste the rows back. When you finish, both lasers have the same number of points. Access types are shared by all lasers. An access type that reads on a trigger reads the points of every laser when it fires. The `Operator.IsMarking` point uses an access type that polls always, not only while a display shows it, so a laser that is not on screen is still watched; the copy keeps it.

Scripts

Each laser has its own mark logging task. The rest of the logic is shared, and one setting tells the scripts how many lasers there are.

Item	Change
Mark logging task	Copy an existing one, for example `tMarkLaser2`, and name the copy `tMarkLaser3`. Set its trigger to `Tag.Laser3.Operator.IsMarking` and replace every `Laser2` in its code with `Laser3`.
`Global.LaserCount`	Set it to 3. The laser selection in the header display and the shift counter reset both follow it, so neither script needs editing.

A logging task watches its own laser only, so every laser is logged whichever laser the screen shows, and two lasers can mark at the same time. The shared class (`cProductionLog`) and the startup script need no change.

Station PLC

When the solution monitors each station through a PLC, with one status tag per station, create the status tag of the new station with the same type. Copy its PLC points with the addresses of the new station, then copy the alarm items and expressions that use the existing station tag and replace the tag name.

Verification

Check	Expected result
Start the runtime	The `SIC_L3` channel, and `SIC_L3_Flash` when used, connect without communication errors.
Select `Laser3` on screen	The displays show the job and the entities of the new laser.
Mark a part on `Laser3` while another laser is selected on screen	A production record with `Laser3` is added, and the mark counter of `Laser3` goes up by 1.
Read the flash job list on `Laser3` (Telnet only)	The list shows the jobs stored on the new laser card.

Notes and Limitations

One point, one exchange

The interface is strictly one command and one reply, so points are never grouped into blocks. A `DV`, `LV`, `SV`, `ED`, `EL` or `ES` point is a single exchange; a `BIT` or `CALC` point is several; `SEL:EntityList` costs one exchange per top-level entity the first time it is built, and two per poll after that — the connector caches the list and rebuilds it only when the entity count changes or when a command it issued replaced the job. Set scan rates accordingly: the derived points and the enumerations are the expensive ones and belong on a slow scan or on demand.

Commands act on a non-zero value

Every `CMD` point ignores a value of `0` or an empty string. That is what makes a timer tag a safe trigger: a delay-off timer writes `1` on the press and `0` when it expires, and only the press fires the command.

Points with nothing to report

A point whose value does not apply to what is currently selected reports **Bad quality** rather than a communication error. Reading a text-only property while a barcode is selected, or a per-entity flag while the whole job is selected, is an ordinary state on a screen whose points span entity types — it changes as soon as the operator picks a different object. Gate screen logic on quality, and keep the error counters meaningful for real transport faults.

Writes are not treated this way. A write that cannot be performed fails, so an operator action is never silently discarded.

Counters and date objects generate their own content

A counter and a date/time object are the same class of entity, and neither stores the text it marks — it rebuilds it at every mark from the counter and the format string, or from the next line of a file. Writing `ES:Content` on one of these appears to work and is then discarded by the next mark.

Drive them through the field that actually produces the text: `ES:SerialFormat` for a counter, `CALC:DateTimeFormat` for a date object. `SEL:EntityType` and `BIT:IsDateTime` together tell a screen which of the three fields to offer. `ES:Content` remains useful on all of them as a read-only preview of what will be marked.

The guide pointer follows whole-mark geometry

Writing `DV:OpticOffsetX`, `DV:OpticOffsetY`, `DV:OpticRotation`, `CALC:MarkWidth` or `CALC:MarkHeight` restarts the guide pointer, so its outline redraws in the new position without the operator switching it off and on. The pointer is started unconditionally: adjusting one of these while the pointer is off will switch it on. The per-entity geometry points do not do this.

Marking requires the pointer to be stopped first, and `CMD:Mark` and `CMD:MarkSelected` handle that themselves.

Selection ordering

Write `SEL:Entity` before the points that depend on it. Points are sorted so the selection is sent ahead of the rest within a batch, but that is a mitigation and not a guarantee: nothing forces both writes into the same batch. Driving the selection from an operator action, and the operations from a separate one, avoids the question entirely.

Changes live in memory until saved

Everything written to entity and pen values changes only the job held in memory. `CMD:SaveJob` writes it to disk. The optic values are the exception in the other direction — they are machine settings and `CMD:SaveJob` does not persist them.

Read-only by nature

Some values can be written but never read back, so the connector derives the readable form instead: barcode width and height, the counter's prefix and suffix, and the symbol size are all reported through `CALC` points rather than the raw identifiers.

Not covered

The wobble shape, the date/time offset value, and the interface language are configured in the laser software and have no command in this interface.

Driver Revision History

SIC Revision History	
Version	**Notes**
1.0.1.1	New Save As command that writes the job to a configurable path and can be repeated to the same file; job file name readable; entity list now covers top-level entities only and is much faster, without changing SAMLight's own settings; new points to read the values every entity marks, replace a placeholder's artwork with an image or vector file, and list the jobs stored on the USC card; width and height now resize logos, bitmaps and shapes; flash commands now default to slot 1 instead of refusing until a slot is written.
1.0.1.0	Marking boundaries (`FieldMin/Max`, `WorkAreaMin/Max`) and marking diagnostics (`ExecutionStatus`, `MarkResult`) exposed as points. `STS:IsMarking` answers the interface's own marking query and no longer needs configuring; the `IsMarkingSource` channel option is removed. Flash commands take their slot and path from `SEL:FlashSlot` and `SEL:FlashPath` and are plain triggers. Marking content is written through the interface's text-change call so the mark reflects it. Points with nothing to report use Bad quality instead of a communication error. Whole-mark geometry writes redraw the guide pointer, and marking stops it first. The entity list is cached. `CMD:ImportReplace` and `SEL:ImportPath` replace the artwork of an existing entity and fit it to the old box. `CALC:EntityWidth` and `CALC:EntityHeight` also resize entities without size limits, such as imported logos. `SEL:EntityValues` reads what every entity marks. The `FLASH` family reads the card's flash job list over its Telnet port, as text or as a DataTable.

1.0.0.0	Initial release. Job load and save, pen and laser parameters, optic transform, entity geometry, text and barcode content, counters and date/time objects, hatching and wobble, flash job storage. Entity and pen are selected through the `SEL:Entity` and `SEL:Pen` points.

In this section...