1-Master Channel

Master Channel

The Master Channel configuration defines a "Scan Group" to a collection of field units that may be over one of the available serial or network circuits. The Master Channel defines how each field unit is scanned over the network, and is independent of which protocols are being used on individual field units.

Attributes	Function
Object Type	Master Channel
Parent(s)	System Clients Master Channels
Instance	0 to 15 (typically 15 is reserved for Internal Channels

The Master Channel must have at least one child Circuit object defined (Async or Network).

Properties	Values
Name	Enter the Master Channel name. *This is the name which appears in the user diagnostics menu, and is also see in an HCP diagnostic window if used with Elecsys HCP.*
Channel Type	*In some configurations this may be listed as "Direct Master", which includes a few operational differences noted below. The main differences between the Channel Types are listed below:* ***Direct Master*** • *Global Scan Period* • *One failed poll changes RTU comms status.* ***Direct Master Flex Scan*** • *Scan Period configured for each scan.* • *All "effective" polls must fail before RTU comms status fails.*
Auto Start	Select the automatic polling method for the channel. *Automatic polling types supported are:* • ***Yes*** *– polling started automatically upon power-up* • ***No*** *– polling started manually through the MMI* • ***Link Based Poll*** *– polling is started only after a P/R connection has been made from an HCP.*
Response Timeout	Enter the response timeout in milliseconds. *Time in milliseconds to wait for a poll response before declaring the message failed.* *For Network Circuits, if a scan fails because the socket is broken or interface is unavailable, then the Master Channel protocol will wait a period of time (Response Timeout * 2) to try the next IP address in the Circuit.*
Broadcast Delay	Enter the broadcast delay in milliseconds. When a host computer sends a command to a field unit via the Master Channel, some field units do not want to be polled again for a certain amount of time to allow processing the command. This option allows normal polling to be delayed temporarily. *Delay in milliseconds after a command is sent to the field device before normal polling resumes. Normally this can be left to the default of 0.*
Interpoll delay	Enter the interpoll delay in milliseconds. Use this to add a delay between each poll sent by the channel to any field unit. *Time in milliseconds to wait between each poll.*

Scan Effective Limit	The Scan Effective Limit is the time (in seconds) defining which scans in the Scan Table are considered "effective" – meaning, polls which affect the status of the Field Unit if there are poll failures. Scan Table entries which have a Scan Period greater than the Scan Effective Limit do not mark the Field Unit offline when the scan fails. *For instance, if the Scan Effective Limit is configured for 30 seconds, then any scans defined with Scan Period <= 30 will be used to mark the Field unit online or offline. Scans with Scan Period greater than 30 will not mark the Field Unit offline even if they fail. The Scan Effective Limit only applies to the "Direct Master Flex Scan Table" version of the channel object.* A Scan Effective Limit of 0 disables this feature, thus all polls will affect the Field Unit status.
Network Recovery	Enter the network recovery period in seconds. *Time period to wait after an RTU fails, before attempting to re-establish communications with that RTU. This will take the device off scan, allowing other devices on the channel to be polled more frequently and not waste as much time retrying a failed device.*
Scan Table	Click the **Edit Table** button to define the order and selection of polls to be sent to all field units on this channel, independent of protocol. Field Unit configurations (Modbus, etc.) define the protocol-specific nature of the individual polls that are sent. **Scan Table details**: **Unit Address** - This is the Field Unit Address as configured in each field unit on this Channel. (To force the Scan Table to ignore the Scan Period, enter a Scan Table row with the Unit Address of -1.) **Poll Record** - This is the row number in the Poll Record in the Field Unit definition. The first row in a Poll Table is referenced as record 1. Only those polls which are to be polled continuously need to be listed in this Scan Table. **Scan Period** - Enter the scan period in seconds. The Scan Period is the amount of time to use for scheduling each scan (global for all scans in the Direct Master, or configured per scan row in the Direct Master Flex Scan Table). For the Direct Master, the channel will restart the scan table sequence after the Scan Period has expired. If the total time for a given channel exceeds the scan period, the next scan shall be scheduled immediately. For the Direct Master Flex Scan Table, each poll is scheduled based on its own Scan Period. If the total time required for scans at any point is greater than allowed by the Scan Periods, the scans will operate as fast as possible. Setting a Scan Period to a negative number will disable a scan. However, the first entry in the Scan Table for each Unit Address should not be disabled, or it may not correctly set the Alive/Dead status of the unit. **Comment -** Optional column, allowing a descriptive comment to be entered for each row in the table. The Comment field is unused in the configuration.

Async Circuit

An Async Circuit is a serial communications path to one or more field units from a common Master Channel, using an Async serial port. The Async Circuit allows for redundant serial ports to a common set of field units, such as a Primary and Secondary radio or modem communication path.

You should generally use a single AsyncCircuit object for a single physical serial port, and include multiple FieldUnit objects under it if they are multidropped on the same serial line. (One exception to this is when mixing serial communications with a DF1 PLC and other devices, since the DF1 has a customized circuit definition.)

Attributes	Function
Object Type	AsyncCircuit
Parent(s)	System Clients Master Channels Master Channel
Instance	Must be between 0 and 16.

The Async Circuit should have at least one Field Unit child object defined under it.

Properties	Values

Primary Port	Select the primary physical communication port for this circuit. The selected port must be defined as an object under Networks, where its Async port properties (baud rate, etc.) are also defined (see the section Async Port). *The same port (e.g. COM1) may also be shared with certain other tasks, such as Terminal Server, and may be used with Virtual Ports.*

Network Circuit

A Network Circuit is an IP network communications path to one or more field units from a common Master Channel. The Network Circuit is used when the field unit is connected via a network, such as TCP/IP, PPP, or SLIP.

Because of the fact that the Network Circuit includes the IP address of the end device, you will generally need to use multiple Network Circuit objects under a channel, one per device. (An exception would be a bridged device that uses a single IP address but represents multiple protocol FieldUnit devices.)

Each Network Circuit represents a TCP socket connection to a device, which is made when the Master Channel initiates a poll to the device. Each socket (one or more, if configured) is kept open independently according to the Failover Delay parameter (time to live). This avoids having to open and close sockets repeatedly to the device, as long as the scan interval is less than the Failover Delay and polls are successful.

Attributes	Function
Object Type	NetCircuit
Parent(s)	System Clients Master Channels Master Channel
Instance	Must be consecutive, starting from 0 (unique among other types of circuits). The Network Circuit should have at least one Field Unit child object defined under it.

Properties	Values
Circuit Type	Select the circuit type as 'Network Circuit'
Failover Delay	Time to Live for network socket (in seconds). Make sure this is large enough to allow for the time it takes to poll on this circuit, taking into account timeouts and interpoll delays of all scans on the channel.
Master Network Port	Must be consecutive, starting from 0 (unique among other types of circuits). The Network Circuit should have at least one Field Unit child object defined under it.
Connect Table	Click the **Edit Table** button to edit the IP address or list of IP addresses to connect for the Field Unit(s) on this circuit. Entering multiple IP addresses will allow failover connections when one connection fails (all connections made to the same Master Network Port on different IP addresses). **Destination Address** Enter the IP address to connect. *The IP address must be in the same IP network or reachable via the Default Gateway or Route Table configuration.* **Interface** Enter the network interface over which to connect to this Destination Address. The network interface must match the Interface name in the ACE object (such as "Ether1") rather than the Linux interface name (such as "eth0").

DF1 RS-232 Async Circuit

The DF1 RS-232 Async Circuit is a special serial communications path to one or more Allen Bradley DF1 field units from a common Master Channel. Use this circuit instead of the generic Async Circuit when configuring a DF1 field unit under a Master Channel.

See the Protocol_DF1-CSP-Master protocol documentation for information on configuring the DF1 RS-232 Async Circuit and FieldUnit.

HART Circuit

The HART Circuit object is a special serial communications path for one or more HART devices from a common master channel. Use this circuit instead of the generic Async Circuit when configuring a HART device under a Master Channel.

See the Protocol_HART-Master protocol documentation for information on configuring the HART Circuit and FieldUnit.

NMEA (GPS) Field Unit

The NMEA Field Unit object contains unique information for a special internal Field Unit that reads location information from an Elecsys cellular modem.

Attributes	Function
Object Type	FieldUnitModbus32, FieldUnitModbusTCP32
Parent(s)	System Clients Master Channels Master Channel Circuit
Instance	Must be unique under a channel

The NMEA Field Unit must have an RTDB child object defined under it.

Properties	Values
Unit Name	Enter the field unit name. *Unit name is displayed in diagnostic menus and in an HCP diagnostic screen.*
Unit Address	Enter the actual field unit address which is configured in the device being polled. *Valid Modbus addresses 1 to 255.*

FieldUnit - Modbus Master (and others)

See the Elecsys documentation on various FieldUnit protocols for information on configuring the FieldUnit, including protocol-specific Poll Table, such as:

– Protocol_Modbus-Master

– Modbus SOS (Specific Outstation) poll modifications

– Protocol_DF1-CSP-Master

– Protocol_HART-Master

RTDB – RealTime DataBase

An RTDB (Real Time DataBase) defines the size of the virtual database reserved for the Field Unit. All FieldUnit objects require a child RTDB in order to function properly, which is defined using a numeric register address format (typically, using Modbus-like addresses).

Attributes	Function
Object Type	RtdbMod
Parent(s)	System Clients Master Channels Master Channel Circuit Field Unit
Instance	Must be 0.

The RTDB object supports several additional optional child objects (see the sections Deadband , Pre-Initialized RTDB, Tag Names, Data Blocking, and Timestamp).

Properties	Values

Database Definition	Click the **Edit Table** button to edit the details of the RTDB definition. **Point Count** – Enter the number of data points of this type to be allocated space in the database. **Field Format** – Select the point data format: ***Boolean*** – Boolean ***UINT8*** – Unsigned 8-bit integer (0 to 255) ***SINT16*** – Signed 16-bit integer (-32,768 to 32,767) ***UINT16*** – Unsigned 16-bit integer (0 to 65,535) ***SINT32*** – Signed 32-bit long integer ***UINT32*** – Unsigned 32-bit long integer ***REAL32*** – IEEE floating point (32-bit) ***STRING32*** – Each field contains up to 32 ASCII characters ***STRING256*** – Each field contains up to 256 ASCII characters ***EVENT*** – Timestamped event data obtained from field device. The following field formats are the same as the above but do not generate an RBE flag when the data changes, even if the Field Unit is set to Produce RBEs=Yes. ***No-Rbe Boolean*** – Boolean ***No-Rbe UINT8*** – Unsigned 8-bit integer (0 to 255) ***No-Rbe SINT16*** – Signed 16-bit integer (-32,768 to 32,767) ***No-Rbe UINT16*** – Unsigned 16-bit integer (0 to 65,535) ***No-Rbe SINT32*** – Signed 32-bit long integer ***No-Rbe UINT32*** – Unsigned 32-bit long integer ***No-Rbe REAL32*** – IEEE floating point (32-bit) ***No-Rbe STRING32*** – Each field contains up to 32 ASCII characters ***No-Rbe STRING256*** – Each field contains up to 256 ASCII characters **Data Address** – Enter the address of the starting register within the RTDB for the Field Format and Count defined on this row. The RTDB fields must be defined so they are non-overlapping, and there need to be enough data points defined to hold all of the data returned in the Poll Table entries defined for this FieldUnit. All RTDB database fields (except String types) may hold 32-bit data items, regardless of the data type or address. The RTDB typically uses registers defined in the range of Modbus addresses, although this is not a strict requirement. However, if the RTDB is connected to a Modbus Slave Channel, it does require Modbus addressing to work properly as a slave (see the Modbus Slave Channel documentation). **Comment** - Optional column, allowing a descriptive comment to be entered for each row in the table. The Comment field is unused in the configuration.

Deadband

A Deadband object defines deadbands for the data fields configured within a Real-time Database (RTDB). This is only used to reduce the communications traffic on an RBE (Report by Exception) connection. If no RBE connection is configured, an RTDB does not require a Deadband object.

The way the deadband works is that when a poll occurs and data is received from a Field Unit, if there is a Deadband defined for any of the points included in the poll, the current value in the RTDB is checked first. If the new values are not changed from the existing RTDB values by an amount greater than the deadband, the values are discarded and not stored in the RTDB.

Attributes	Function
Object Type	LinuxDeadband
Parent(s)	System Clients Master Channels Master Channel Circuit Field Unit RTDB
Instance	Must be 0

Properties	Values

Deadband	Click the **Edit Table** button and add as many rows as necessary to define the desired deadband values for the points in the RTDB. • **Field(Row)** – Enter the row number of the field in the RTDB, containing the data point. RTDB row numbers start at 1. • **Offset** – Enter the offset into data point address referenced in the field. Offset of 0 refers to the first point number in the RTDB field. • **Count** – Enter the number of data points that the deadband limit will be applied to. • **Deadband** – Enter the deadband value, which is the amount a value in the RTDB can change before it will be flagged as an RBE. *Deadband value is entered as an integer or floating point value, which is handled as an 11-character (max) string. If IEEE floating point format is used, its entry must include a decimal point (such as 11.0).* • **Comment** - *Optional column, allowing a descriptive comment to be entered for each row in the table. The Comment field is unused in the configuration.*

For example, let's say the RTDB is configured with the following fields:

	Point Count	Type	Data address
Field 1:	100	Boolean	1
Field 2:	100	Boolean	10001
Field 3:	100	UINT16	30001
Field 4:	100	UINT32	40001

The first four analogs at address 30,001 are 12-bit analogs that change from 0 to 4095, and we want to deadband them to report as RBE only when their values change more than 5% of their range (205). The 3rd and 4th analogs in the range starting at 40,001 we want to throttle their RBE reports to only change when the values increase or decrease by 100 and 500, respectively. All other points will be allowed to report as RBE with any single change positive or negative in their values. For this example, the Deadband table will be defined as follows:

Field	Offset	Count	Deadband	Explanation
3	0	4	205	Deadband for 30,001-30,004, 5% of its range
4	2	1	100	Deadband for 40,003
4	3	1	500	Deadband for 40,004

Pre-Initialized RTDB

Ordinarily, all RTDB database locations are initialized to zero values upon system startup (or zero-length strings). However, sometimes it may be desired to initialize certain database locations to a non-zero value, before any polling or other data operation occurs. Each RTDB has an optional ACE object that allows one or more registers to be initialized at startup.

Attributes	Function
Object Type	PreInitRtdb
Parent(s)	System Clients Master Channels Master Channel Circuit Field Unit RTDB
Instance	Must be 0

Properties	Values

Init Values	Click the **Edit Table** button to define any pre-initialized RTDB values. • **Data Address** – Enter the starting register number to define default values. This should be a register number that is defined as part of the RTDB. • **Count** – Enter a count of registers, beginning with Data Address, that should be initialized to the same value. • **Init Value** – Enter the value to which the register(s) will be initialized on startup. For Boolean registers, enter a '0' or '1' initial value. For floating point registers, enter the initial value in floating point format. • **Comment** - Optional column, allowing a descriptive comment to be entered for each row in the table. The Comment field is unused in the configuration.

Tag Names

RTDB database locations are configured using numeric address locations. However, the optional Tag Names child object under the RTDB allows one or more numeric address to be associated with an ASCII tag. This may be used for publishing data by tag using MQTT, for internal display using Custom Reports, and they may be used for other purposes.

Attributes	Function
Object Type	TagNames
Parent(s)	System Clients Master Channels Master Channel Circuit Field Unit RTDB
Instance	Must be 0

Properties	Values
Init Values	Click the **Edit Table** button to define any RTDB tags. • **Register Address** – Enter the RTDB register address in the parent RTDB object. • **Tag Name** – Enter an ASCII tag (up to 32 characters). Do not use the following characters: "|" (pipe), "\" (backslash), and "," (comma). Space characters in the tag are converted to underscores. • **Comment** - Optional column, allowing a descriptive comment to be entered for each row in the table. The Comment field is unused in the configuration.

> *When publishing to MQTT, data values are published with corresponding tag name, if configured in ACE, with some exceptions (substitutions) noted below.*
>
> *With MQ-RBE, Sparkplug-B, or MQ-JSON:*
>
> - *If tag name includes a space, the space will be converted to an underscore _ character instead.*
> - *The space-to-underscore conversion also applies to the Ethernet/IP master protocol.*
>
> *With MQ-RBE or Sparkplug-B (not JSON):*
>
> - *If tag name includes a period . it's tag will be published with a forward slash / instead.*
> - *If tag name includes an integer between square brackets for an array (such as [23]), it's tag will be published with the integer surrounded by forward slash and underscore instead (such as /23_).*
> - *In Ignition, the forward slash in a published tag name creates a level in the collapsible tag hierarchy.*

Data Blocking

The Data Blocking object allows groups of RTDB points to be blocked together for exception reporting (RBE) to an HCP. If any one point in the defined Data Block changes, all the points are reported, including the unchanged ones. If no data blocking capability is required, this object is optional.

Although it is normally recommended to store 32-bit data into 32-bit registers, Data Blocking could be used if a configuration requires 32-bit data to be stored in pairs of 16-bit registers. Each pair of registers could be defined in a separate row (count of 2), and if either value changes, the Data Block will force both registers in the pair to be reported together.

Attributes	Function
Object Type	
Parent(s)	System Clients Master Channels Master Channel Circuit Field Unit RTDB
Instance	Must be 0

Properties	Values
Block Definition	Click the **Edit Table** button to add as many rows as necessary to define the Data Block capability. • **Data Address** – Enter the register number of the beginning of each block. This should be a register number that is defined as part of the RTDB. • **Point Count** – Enter the number of data points to group together in each block. Ensure the Point Count is not defined larger than the available points in the RTDB field. Data blocking currently supports a contiguous block of data registers using a range of different data types (Boolean, UINT16, UINT32, REAL32), but cannot be used on larger data types (Strings, 64-bit data). • **Comment** - Optional column, allowing a descriptive comment to be entered for each row in the table. The Comment field is unused in the configuration.
Parent(s)	System Clients Master Channels Master Channel Field Unit RTDB
Instance	Must be 0

> Data Blocking does not work properly if you are using Pre-Initialized registers on the same RTDB. Blocks will be broken up at the boundaries of pre-initialized registers.
>
> Data Blocking does not work if the blocks span discontiguous (non-sequential) register addresses in the RTDB.

Linux Timestamp

A Timestamp object is used to store the time and date at which data is polled by a Master Channel. The timestamp is stored in register(s) within the RTDB, and thus may itself be reported with the RBE packet or polled via a Slave Channel.
Timestamps may be stored in one of two conditions whenever a specified poll occurs:

- "Always" = store timestamp whenever a poll for data occurs, even if nothing is stored in the RTDB because deadband values have not been exceeded.
- "Post-Deadband" = store timestamp only when one or more data points is stored into the database. If Deadbands are configured, data is not stored into the database until the difference between the old value and new value exceeds the configured deadband.

Attributes	Function
Object Type	LinuxTimeStamp
Parent(s)	System Clients Master Channels Master Channel Circuit Field Unit RTDB
Instance	Must be 0

Properties	Values
Reserved	Unused field

Timestamp	Click the **Edit Table** button to add as many rows as necessary to define the Timestamp operation. **Poll Number** – Enter the row number of the poll defined in the unit's Poll Table. Whenever this poll occurs for the defined points (or when any changed data points are stored in the RTDB field), a timestamp is also stored. Poll numbers start at 1. **Stamp Address** – Register address within the RTDB for this Field Unit in which to store the timestamp value for this poll. The Stamp Address should be the first of one or more registers with a UINT16 or UINT32 data type, and must be defined in the RTDB with the correct quantity and type. Make sure that each register or registers occupied by the timestamp are not overwritten by any other data value to avoid conflicting data. **Stamp Format** – Data format to use when storing data into the specified register (UINT16 or UINT32). The Stamp Format should be chosen appropriately to match the Stamp Type (below), and the data type of the RTDB register. UINT24 or UINT32 data types should be stored into a UINT32 RTDB register. • UINT16 • UINT32 • UINT64 (seconds*1000 + mSec from 1969 if Packed, else 1979) **Stamp Type** – Format in which to store the timestamp. • **32 bit centi-seconds (1 reg, Always)** - Store timestamp at the time each poll is initiated, as a 32-bit number as centi-seconds (10's of milliseconds) since the last startup. • **32 bit seconds + mSec (2 reg, Always)** - Store timestamp at the time each poll is initiated, as two 32-bit numbers (seconds since January 1, 1980; and milliseconds). • **YYYY,MM,DD,HH,mm,ss,mSec (7 reg, Always)** - Store timestamp at the time each poll is initiated, as seven 16-bit registers containing year, month (1-12), day (1-31), hour, minute, second, milliseconds. • **32 bit centi-seconds (1 reg, Post-Deadband)** - Store timestamp only on changed RTDB data, as a 32-bit number as centi-seconds (10's of milliseconds) since the last startup. • **32 bit seconds + mSec (2 reg, Post-Deadband)** - Store timestamp only on changed RTDB data, as two 32-bit numbers (seconds since January 1, 1980; and milliseconds). • **YYYY,MM,DD,HH,mm,ss,mSec (7 reg, Post-Deadband)** - Store timestamp only on changed RTDB data, as seven 16-bit registers containing year, month (1-12), day (1-31), hour, minute, second, milliseconds. **Comment** - Optional column, allowing a descriptive comment to be entered for each row in the table. The Comment field is unused in the configuration.