Document #: 12041

Status: Approved

Version 1.9

SunSpec Information Model Specification

SunSpec Alliance Interoperability Specification

ABSTRACT

The SunSpec Alliance Interoperability Specification suite consists of the following documents:

- SunSpec Technology Overview
- SunSpec Information Model Specification
- SunSpec Information Model Reference Spreadsheet
- Collection of SunSpec Device Category Model Specifications
- SunSpec Plant Extract Document
- SunSpec Model Data Exchange

This SunSpec Information Model Specification describes the construction of the information models. It applies to Models of all device categories. It is intended to provide SunSpec users and device implementers the detailed information needed to interoperate with existing SunSpec-compliant devices, or implement new devices.

About the SunSpec Alliance

The SunSpec Alliance is a trade alliance of developers, manufacturers, operators and service providers, together pursuing open information standards for the distributed energy industry. SunSpec standards address most operational aspects of PV, storage and other distributed energy power plants on the smart grid—including residential, commercial, and utility-scale systems—thus reducing cost, promoting innovation, and accelerating industry growth.

Over 70 organizations are members of the SunSpec Alliance, including global leaders from Asia, Europe, and North America. Membership is open to corporations, non-profits, and individuals. For more information about the SunSpec Alliance, or to download SunSpec specifications at no charge, please visit www.sunspec.org.

Change History

1.3: Added change history
Allow device aggregation via multiple common models
Manufacturer ID should be registered with SunSpec
Added M/O/C column to xls map.
Require Manufacturer, Model, and Serial Number to be supplied

1.4 Add Copyright

1.5 Updated Logo
Corrected Assigned ID range for String Combiner and Module
Added Assigned ID ranges for Inverter model, version 1.2, Module model, version 1.1, Inverter Controls and Network Interface.
Added concept of a scale factor as a basic SunSpec type.
Added concept of SunSpec Device type to Common Model
Changed over to new names for points
Added definition for acc64, ipaddr, ipv6addr types
Scale factor clarification wrt unimplemented
Align the document with the new short names
Update Network Configuration, new model ids and some descriptions
Add recommendation on default serial port settings (9600,8,N,1)
Removed model table and replaced with reference to the PICS
Added descriptions for bitfield and enum
John Blair's contribution: document new types, terminology, model structure, best practices, other conventions. Included Control Procedural Requirements

1.6 Call out conformance statements, add new conformance requirements for control
New title to better distinguish "Device Modeling" from "Common Model" overload

1.7 Restructure document

1.8 Updated descriptions. Underlying model definitions are unchanged.

1.9 Further descriptive clarification.

Table of Contents

Introduction 5
Terminology 6
Specification Status Lifecycle 6
SunSpec Certification 6
SunSpec Information Model Definition 7
SunSpec Device Definitions 7
Information Model Structure and Contents 8
Common Model 8
Standard Model 9
Vendor Models 9
End Model 9
Model Structure 9
Canonical Structure 9
Determining the Number of Repeating Instances 10
Cases Derived From the Canonical Structure 11
Zero Length Model 13
Data Points 13
Rules for Implementing Data Points 13
Standard Data Formats 13
16-bit Integer Values 14
32-bit Integer Values 14
64-bit Integer Values 15
128 Bit Integer Values 15
String Values 15
Floating Point Values 16
Scale Factors 16
Defined Units 16
Modbus Register Mappings 16
Base and Alternate Base Register Addresses 17
Best Practices for Model Design 17
Group scale factors 17
Use Instance Scale Factors When Appropriate 18
Use PAD to keep 32 and 64 bit alignment 18

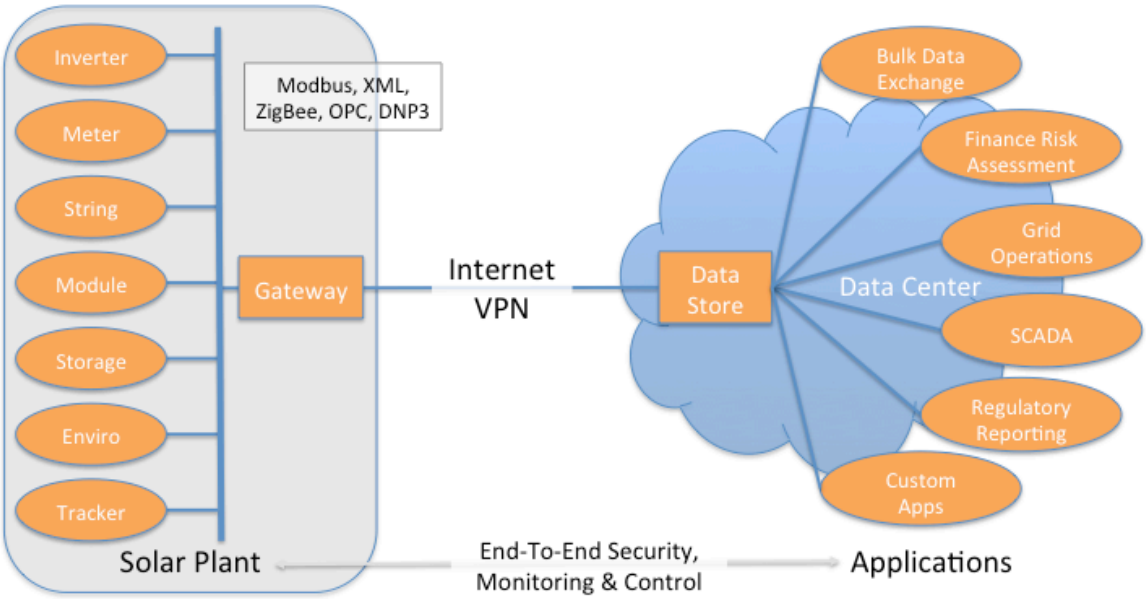
Reuse point IDs from other models 18
Reuse status and events from other models 18
SunSpec Procedural Requirements 18
Status Codes 18
Error Handling 19
Unimplemented Registers 19
Invalid Setting Value 19
Read-Only and Write-Only Registers 19
Incomplete Enumeration 19
Incomplete Operation 19
Organization of Control Read/Write Values 19
Procedures for Multi-Write Operations 21

Introduction

This document offers a detailed description of the construction of SunSpec Information Models. It applies to Models of all device categories.

This information is useful for those wanting to understand or create SunSpec Information Model definitions.

The Information Models can be used to convey device data between any two communicating entities by mapping them to the communications protocol appropriate for the entities. The typical use case is depicted in the following diagram, where the communication between the devices and a gateway is Modbus and the communication between the gateway and data store is an Internet protocol such as HTTP. The Modbus mapping is defined in this document in the Modbus Register Mappings section; the Internet communication typically uses the XML encoding directly; this usage is described in the SunSpec Model Data Exchange document.



Terminology

Device	A physical object that performs a set of functions. Examples are inverters, trackers, and modules.
Device Description	The collection of Information Models that describe a SunSpec device implementation.
Model	A set of data points describing a logical functional block. Information Models are comprised of three elements: the Model ID, the length of the data (in 16-bit registers) and the data.
Block	A collection of data points. The canonical SunSpec Model consists of two blocks, fixed and repeating, each of which is optional.
Instance	A single occurrence of a block that is repeated in a repeating block.
Point	A value encoded using one of the SunSpec data types.
PICS	Protocol Implementation Conformance Statement (PICS). All SunSpec implementations are declared by the vendor in a Protocol Implementation Conformance Statement (PICS). The PICS specifies the details of a specific implementation and is used for verification of conformance to SunSpec standards.

Specification Status Lifecycle

SunSpec Interoperability Specifications follow the lifecycle pattern of

- DRAFT
- TEST
- APPROVED
- SUPERSEDED

TEST specifications may be implemented but resulting implementations are not subject to SunSpec Certification. APPROVED specifications may be implemented and resulting implementations are subject to SunSpec Certification. SUPERCEDED specifications may also be implemented and resulting implementations are subject to SunSpec Certification, but use of SUPERCEDED specifications is discouraged.

Status for each Model is indicated on the Index page of the Comprehensive Data Model and Modbus Map. Be sure to check the status of any model before you implement. Only approved models can be certified by SunSpec.

SunSpec Certification

Devices implementing the Models specified herein can be certified by SunSpec; see http://sunspec.org/sunspec-certified-program.

SunSpec Information Model Definition

SunSpec Information Models are defined using the SunSpec Model Definition XML (SMDX) encoding. Reference the SMDX file for the definitive version of any Model. The SMDX files are also compiled into a spreadsheet called SunSpec Information Model Reference.xlsx for human consumption. There is currently one Excel tab per Model definition. The SMDX file is the definitive version and the spreadsheet is only provided for readability.

See 'Modbus Register Mappings' for how these models are represented in Modbus, and the SunSpec Model Data Exchange specification for how they are represented in XML.

A key feature of the SunSpec approach is flexibility allowing vendors to extend the capabilities of a device category or develop new device categories. SunSpec definition techniques allow for extensibility in the following ways:

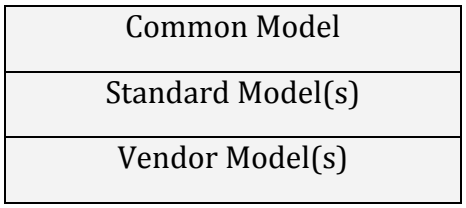
- Definition of new Standard Models. New Models are defined and assigned new model ID's by SunSpec.
- Revisions of Standard Models. Revisions to standard Models are defined and assigned new Model ID's by SunSpec.
- Vendor Models. Vendors may also define a Vendor Model that includes fields and values specific to the vendor. These are assigned a Model id in the 65xxx range by SunSpec.

SunSpec Device Definitions

A Device Definition is a collection of two or more information Models: a Common Model and one or more Standard or Vendor Models.

Each information Model is uniquely defined and contains a well-known identifier and length. This allows a client to browse the contents of a device description and skip information Models with unrecognized identifier (ID) values.

The SunSpec device definition structure is shown below:



Each category of device has a corresponding Information Model detailing the points specific to that category. Currently supported SunSpec Device Categories are

- Inverters
- Meters
- Panels
- Environmental Sensors
- String Combiners
- Trackers
- Energy Storage
- Charge Controllers

Additional device categories will be taken up and defined as directed by the SunSpec Alliance membership.

Information Model Structure and Contents

Common Model

The Common Model is where manufacturer and product-model identification information is specified for a SunSpec device implementation.

The following data elements comprise the Common Model and must be included to uniquely identify a SunSpec device implementation.

- **ID –** A well-known value, "1", that uniquely identifies this as the SunSpec Common Model.
- **L –** A well-known value, "66" or "65". Implementations of length 66 shall contain a pad register as the final register.
- **Mn** – A unique value that identifies the Manufacturer of this device. Manufacturers must register their ID with SunSpec to guarantee uniqueness. SunSpec maintains a database of certified products by Manufacturer. The Mn should be concise and must be constrained to simple alphanumeric text void of spacing.
- **Md** – A manufacturer specific value that identifies the product model of this device. SunSpec maintains a database of certified products by model. The Md should be concise and must be constrained to simple alphanumeric text void of spacing.
- **Opt** - A manufacturer specific value that identifies any model options for this device.
- **Vr** - A manufacturer specific value that identifies the firmware version of this device.
- **SN** - A manufacturer specific value that uniquely identifies this device within the manufacturer name space.
- **DA** - Protocol specific value to address this device instance. For Modbus devices, this is the Modbus device ID.
- **Pad** – SunSpec pad16 register. Padding must be included for Common Models of length 66. The Pad is not included if the length is 65.

Note: SunSpec requires that the result of concatenating the three strings **Mn, Md,** and **SN** return a globally unique string for all of a manufacture's products. This string may be used (for example) by logging and upgrading functions.

Standard Model

Following the Common Model will be zero or more standard Models. The Standard Model is structured as follows:

- **ID –** A well-known value that uniquely identifies the Model. Valid identifiers are assigned by SunSpec.
- **L –** A value that is the length of the Model in 16-bit registers. The length should be even and may contain a rounding pad register. The length does not include the ID or L register. It is the number of registers that follow.
- Model blocks follow as required. If the ID is not recognized by the client, the L value may be used to skip over the Model contents and move to the next Model.

Vendor Models

Vendor Models can be defined by a device vendor to contain points that are only applicable to the vendor's implementation. Vendor Models do not need to go through the SunSpec standard Model review process but must conform to all rules regarding the creation of a SunSpec Model definition. Vendor Models are structured in the same way as standard Models.

A Vendor Model requires an ID assigned by SunSpec.

End Model

The Device Definition is terminated by a Model with ID ((0xFFFF)), followed by a length (L) value of zero.

Model Structure

Every SunSpec Model contains one or more Blocks.

A Block is a logical group of points within a Model. There are two types of Block: fixed and repeating.

A SunSpec Model may only contain a single fixed Block and may contain multiple instances of a repeating Block. The fixed Block contains points that appear only once in the Model. A repeating block allows the Model to contain multiple instances of the same point group. The diagram below is a visual representation of the Model structure.

Canonical Structure

Every Model starts with a fixed 2-register header specifying the ID of the Model and the length of the data portion (in 16 bit registers) following this header.

This structure is shown here:

Header (Model ID + Length) Fixed Length Block (may be zero length)
Repeating Block (may be zero length)

The repeating Block is composed of one or more repeating *instance*s, where an instance is a sequence of points defined in the Model's specification.

Thus the length of the repeating Block is constrained to be a whole number multiple of the length of an individual instance.

This structure is defined as follows:

Component	**Size (16-bit Registers)**	**Comments**
Model ID	1	Assigned value
Model Length	1	Length of data portion (fixed length block + total repeated block length)
Fixed Length Data Block	Data block length	May be zero length
Repeating Block	Number of instances * instance length	May be zero length

All SunSpec data Models must adhere to the SunSpec canonical format. Reading applications, such as data loggers, depend on this structure to properly parse discovered data Models. The structure also allows reading applications to ignore Models it does not understand yet, and continue to parse any Models it does understand.

Determining the Number of Repeating Instances

In all cases the reading application determines the number of repeating instances by using the following formula:

$n = (l - f) / i$

n	number of repeating instances in the repeating block
l	total length of the Model, as provided by the device
f	length of the fixed length block, in registers
i	length of the repeated block, in registers

There may be cases where a Model also indicates the number of instances in the repeating section in a point, usually named N. The reading application must not rely on this value but should instead compute the number of instances based on the total indicated length.

Cases Derived From the Canonical Structure

The specified structure gives rise to three main classes of Models, plus one degenerate case. We will consider each case in turn along with corresponding examples of existing data Models. Length values will be indicated using the *n*, *i*, *f*, *l* notation described above.

Fixed Length Data Models

The number of registers in the fixed length Block is constant. The repeating section is zero length and thus omitted.

Model ID	*id*
Length	*f*
Fixed Length Block	*f* registers

Examples of data Models with this structure are 1: Common Model, 101: Single Phase Inverter Model and 201: Single Phase Meter Model.

Since there is no repeating section the length indicated by the device must match the specified length for a given Model.

Repeating Models

The repeating Model contains no fixed length section.

Model ID	*id*
Length	*n* * *i*
Repeating Block	*n* * *i* registers

Examples of this structure are 302: Irradiance Model, 303: Back of Module Temperature Model and 304: Inclinometer Model.

To determine the number of repeated instances the reading application must know, from the specification, the length of an instance in a given Model. The reading application divides the indicated total Model length by the known instance length to determine the number of instances present.

For example, if the reading application discovers Model 304 with a length of 18. It knows that the length of the repeating instance in Model 304 is 6, so applying the formula from above it determines there are 3 repeated instances.

3 = (18 – 0) / 6

Combined Model with Both a Fixed Length Block and Variable Length Block

A Model may contain both a fixed length section and a repeating section.

Model ID	*id*
Length	*f* + (*n* * *i*)
Fixed Length Block	*n* registers
Repeating Block	*n* * *i* registers

For example, Model 403, the String Combiner, is composed of a fixed length block of 16 registers followed by a repeating block with an instance length of 8 registers. If a device reports the total length is 112 registers then there must be 12 instances in the repeating section.

12 = (112 – 16) / 8

Zero Length Model

The zero-length Model is included for the sake of completeness. The only zero length Model allowed is the SunSpec end marker, with ID 0xFFFF. In the canonical Model, both the fixed length block and the repeated block are length zero.

ID	0xFFFF
Length	0

Data Points

A data point is a machine-readable data value within a block. A Setting is a point that can be changed.

Rules for Implementing Data Points

The following rules apply to implementing data point values:

- A data point must be of one of the SunSpec data types.
- Data point support is designated in a Model as Mandatory or Optional. Mandatory means the attribute is required to be supported. Optional means support for the point is not required.
- Implementations of a given Model must support all Mandatory data points and contain address space for **all** of the data points defined by the Model.
- Implementations must indicate which Optional data points are not supported by returning the appropriate "Not Implemented" value for the data point value.
- Data points may be designated in the Model as having enumerated values, defined bitfields, or range. Implementations should support all of the values and bits.
- Implementations may or may not support all of the enumerated values or bitfields in a register.
- Implementations that implement a setting must indicate the supported setting values (enumerations or bitfields) or valid range (scalar) and default value in the PICS document.
- Implementations must note any data point value limitations in the associated PICS document.
- Data point support may be designated in the Model as Read-Only (R) or Read/Write (RW). Implementations may implement RW data points as R.
- Implementations must note any access exceptions in the associated PICS document.

Standard Data Formats

Implementations are restricted to data points in the following standard data formats.

- int : signed integer value.
- uint: unsigned integer value
- pad: reserved field, used to round a Model to an even number of registers
- acc: accumulated value, used for ever increasing values that may roll over
- enum: enumerated value, used for status and state
- bitfield: a collection of bits, multi-valued alarms or state
- string: a null terminated or fixed length value
- ip: internet protocol formatted network address

SunSpec Models can be conveyed using any underlying communication protocol. The first, and most popular protocol used, is Modbus. A Modbus mapping is therefore included as part of the specification.

16-bit Integer Values

Values are stored in big-endian order per the Modbus specification and consist of a single register. All integer values are documented as signed or unsigned. All signed values are represented using two's-compliment format.

Modbus Register	1															
Byte	0								1							
Bits	15	14	13	12	10	11	9	8	7	6	5	4	3	2	1	0

int16 Range: -32767 … 32767	Not Implemented: 0x8000
uint16 Range: 0 … 65534	Not Implemented: 0xFFFF
acc16 Range: 0 ... 65535	Not Accumulated: 0x0000
enum16 Range: 0 … 65534	Not Implemented: 0xFFFF
bitfield16 Range: 0 … 0x7FFF	Not Implemented: 0xFFFF
pad Range: 0x8000	Always returns 0x8000

NOTE: it is up to the master to detect rollover of accumulated values.

NOTE: if the most significant bit in a bitfield is set, all other bits shall be ignored.

32-bit Integer Values

32-bit integers are stored using two registers in big-endian order

Modbus Register	1		2	
Byte	0	1	2	3
Bits	31 … 24	23 … 16	15 … 8	7 … 0

int32 Range: -2147483647 … 2147483647	Not Implemented: 0x80000000
uint32 Range: 0 … 4294967294	Not Implemented: 0xFFFFFFFF
acc32 Range: 0 ... 4294967295	Not Accumulated: 0x00000000
enum32 Range: 0 … 4294967294	Not Implemented: 0xFFFFFFFF

bitfield32 Range: 0 … 0x7FFFFFFF Not Implemented: 0xFFFFFFFF
ipaddr 32 bit IPv4 address Not Configured: 0x00000000

NOTE: it is up to the master to detect rollover of accumulated values.

NOTE: if the most significant bit in a bitfield is set, all other bits shall be ignored.

64-bit Integer Values

64-bit integers are stored using four registers in big-endian order.

Modbus Register	1		2	
Byte	0	1	2	3
Bits	63 … 56	55 … 48	47 … 40	39 … 32

Modbus Register	3		4	
Byte	4	5	6	7
Bits	31 … 24	23 … 16	15 … 8	7 … 0

int64 Range: -9223372036854775807 … 9223372036854775807

Not Implemented: 0x8000000000000000

acc64 Range: 0 ... 9223372036854775807 Not Accumulated: 0

NOTE: Only positive values in the int64 range are allowed. Accumulator values outside of the defined range shall be considered invalid.

NOTE: The accumulator value shall rollover after the highest positive value in the int64 range (0x7fffffffffffffff). It is up to the reader to detect rollover of accumulated values.

128 Bit Integer Values

128 bit integers are stored using eight registers in big-endian order.

ipv6addr 128 bit IPv6 address Not Configured: 0

String Values

Store variable length string values in a fixed size register range using a NULL (0 value) to terminate or pad the string. For example, up to 16 characters can be stored in 8 contiguous registers as follows

Modbus Register	1		2		3		4		5		6		7		8	
Byte	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Character	E	X	A	M	P	L	E	spc	S	T	R	I	N	G	!	NULL

NOT_IMPLEMENTED value: all registers filled with NULL or 0x0000

Floating Point Values

Floating point values are 32 bits and encoded according to the IEEE 754 floating point standard.

Modbus Register	1															
Byte	0								1							
Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
IEEE 754	sign	Exponent							Fraction							

Modbus Register	2															
Byte	2								3							
Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
IEEE 754	Fraction least															

float32 Range: see IEEE 754 Not Implemented: 0x7FC00000 (NaN)

Scale Factors

As an alternative to floating point format, values are represented by integer values with a signed scale factor applied. The scale factor explicitly shifts the decimal point to the left (negative value) or the right (positive value). Scale factors may be fixed and specified in the documentation of a value, or may have a variable scale factor associated with it. For example, a value "Value" may have an associated value "Value_SF" of type "sunssf" that is a 16 bit two's compliment integer.

sunssf signed range: -10 … 10 Not Implemented: 0x8000

If a value is implemented and has an associated scale factor, the scale factor must also be implemented.

Defined Units

Units are defined as needed by specific Models. Where units are shared across Models, care will be taken to ensure a common definition of those units.

Modbus Register Mappings

All SunSpec device maps begin at one of the well-known base addresses and start with the well-known 32-bit 'SunS' identifier (0x53756e53). This allows for discovery of SunSpec compatible devices. If the base register does not return this value, the alternate base registers are checked. If this test fails, the device is not SunSpec compatible.

Following the 'SunS' identifiers are the Models as defined in the Model Structure section, above.

‘SunS’ (0x53756e53)
Common Model
Standard Model(s)
Vendor Model(s)
End Model

If you read beyond the end of the device, a Modbus exception may or may not be returned according to the implementation. If no exception is returned, then data that comes after the End Model is invalid and should not be used. It is recommended that masters read the common model to determine the contents of the map.

NOTE: This specification only addresses the format of the data. The data can be moved via Modbus/TCP or RTU – or any other protocol that can move Modbus data..

Base and Alternate Base Register Addresses

Device Modbus maps begin at one of three well-known Modbus base addresses.

Preferred Base Register: 40001
Alternate Base Register: 50001
Alternate Base Register: 00001

Base registers are actual register offsets that start at 1 – not a function code and not to be confused with the Modicon convention, which would represent these as 4x40001 and 4x50001.

To read register 40001, use the hexadecimal offset of 0x9C40 (40000) on the wire.

Best Practices for Model Design

We've spent several years developing data models for multiple classes of devices. In addition to the model design described previously, we have come up with the following list of best practices. These are not hard-and-fast rules; in fact, most of them are violated by one or more models. However, going forward we would like all data models to follow these guidelines.

Group scale factors

Scale factors should be grouped together in a block rather than scattered throughout the model. Grouping the scale factors instead of intermixing them with data points reduces the number of registers a Modbus master will need to read.

For example, model 403, the String Combiner Model shows the scale factors grouped in the fixed-length section at the start of the mode. Actually, there are two groups, since InDCA_SF and InDCAhr_SF were added when 401 was deprecated.

Model 101, the single phase inverter model, puts the scale factors in the model close to the data points they modify. While mitigated by the fact the entire model fits into a single Modbus read, this design requires the scale factors to be read to retrieve all the measurements.

Use Instance Scale Factors When Appropriate

Keep in mind that for a device with many subcomponents represented by instance variables, it may not be appropriate to use the same scale factor for aggregated values and instance values. This mistake was made with the first versions of the String Combiner (401 and 402). The same scale factor was used, for example, for aggregated current and individual string current. This meant the dynamic range of the string current value was limited to the range required by the entire combiner. This issue was corrected in models 403 and 404.

Use PAD to keep 32 and 64 bit alignment

As a consideration to low resource devices model designers should align all 32 and 64 bit values on even numbered offsets. A PAD register should be added at the end of the fixed length block and/or at the end of a repeating block to ensure both add up to an even number of registers. Arrange the other points as needed to place the PAD at the end of the block.

Official SunSpec models should not have more than two PAD registers: one for the fixed length block and one for the repeating block. Vendor models may use PAD to mark ranges of registers as reserved for future use.

Reuse point IDs from other models

Use the same ID as existing models to represent the same physical quantity. This will aid implementation and interpretation of the data model.

Reuse status and events from other models

When two models share similar behavior please re-use existing status and event values. Mark unneeded vales as RESERVED. Compare the inverters models (101, 102 and 103), the Smart Panel models (501 and 502) and the Multiple MPPT Inverter Extension (160) for an example.

SunSpec Procedural Requirements

SunSpec compliant devices adhere to the procedural requirements outlined in this section.

Status Codes

Models that support a status code value in the Standard Model must support the following status values.

NORMAL	0x00000000	:	Operating Normally
ERROR	0xFFFFFFFE	:	Generic Failure

Device specific status codes are defined in corresponding device model.

Error Handling

There is a need to handle errors in a consistent way for conformance testing and interoperability. Applications need a known way to detect errors and handle them.

Unimplemented Registers

Unimplemented registers should have the following behavior:

- READ: The value returned is the SunSpec unimplemented value.
- WRITE: The written value is ignored. No exception is generated.

Invalid Setting Value

When a setting is written with an unsupported value for the implementation, the following must occur:

- An exception "3" Illegal Data Value must be returned and processing of the write operation must terminate. Previous registers may have been written but no subsequent registers can be written.
- Such limitations must be documented in the PICS

Read-Only and Write-Only Registers

The following behavior is defined when attempting to write to a read-only register or read from a write-only register.

- WRITE to R: The written value is ignored. No exception is generated.
- READ from W: If the register is supported returns 0 else returns the unsupported value.

Incomplete Enumeration

Unimplemented enumeration values must be marked as unimplemented in the PICS.

Incomplete Operation

Some operations may not take place in time for a Modbus response. If it is desired to return a Modbus Exception "5" ACKNOWLEDGE, the model must support a completion register. Otherwise, no exception is returned.

Organization of Control Read/Write Values

It is desirable to organize related settings values within the Modbus map in a way that minimizes the number of writes necessary to accomplish updating and activating the settings. Control operations that rely on a group of settings must have a register to control the activation (enabling / disabling) of the control.

To facilitate that goal, the following guidelines are recommended:

- Related writable settings field are organized in a contiguous block
- Activation fields are located at the end of the settings block
- Activation fields only operate on the behavior related to that settings block

Procedures for Multi-Write Operations

For operations that require multiple writes (e.g. set operating parameters and then enable), the procedure is recommended. It is not recommended to disable the control to update the settings.

Enable Procedure:

1. All settings are written
2. The activation field is enabled

Change Procedure:

1. Changed settings are written. Changes do NOT take effect, even if the activation field is already enabled, until the activation field is enabled.
2. The activation field is enabled

Disable Procedure:

1. The activation field is disabled