Framework for Tax Preparation

Charles Duan

May 17, 2021

Contents

1 Introduction

This is a system for preparing tax forms. The idea is to translate the relevant computations and forms into modular computer programs that, when assembled together, can compute a complete tax return based on received tax information.

The basic motivation for this system is the separation of form and content. Tax forms are essentially association tables, in which numbered lines are associated with blank spaces to be filled with numbers or other data. The content could be represented as a two-column spreadsheet, but the complexity of

calculations and interactions among multiple forms makes an object-oriented representation in a general-purpose programming language preferable. Thus, the system separates the computations necessary to figure the tax forms first, and then enters the figured numbers into the graphical forms.

Several advantages arise out of automating the filling of tax forms in this way. Automating the computations allows for testing out different elections during the tax computation (e.g., opting for itemized deductions or the standard deduction) without having to recompute the numbers by hand. It allows for programming accuracy checks to find errors in form inputs (e.g., missed lines on W-2 forms). It also saves time across years, since the computations for most tax forms do not change greatly across years. And it enables multiple people's taxes to be computed using the same software. The primary disadvantage is that programming general-case computations is less efficient than computing a single tax return, but it is believed that the aforementioned advantages outweigh the time spent.

Use of the system proceeds in three phases.

The first phase is the computation of form values. The user enters relevant data from received tax forms and other sources into files, and then writes a short script that imports that data and computes the values for tax forms to be prepared. This produces a plain text file containing the line item data to be filled into the tax forms.

The second phase is the identification of form fields. For each tax form to be filled in, the user provides the PDF file for the blank form and also the file of line item data. A program displays a graphical interface where the user can click on the blank spaces where each line item should be entered. The result of this is a second plain text data file that lists the coordinates of each of the form fields.

The third phase is the actual filling of the forms. This now involves simply running a script that does the form filling.

2 Recommended Procedure for Annual Filing

This section provides recommendations on preparing an annual tax filing. This includes updating the software in view of changes to the tax laws and forms, as well as preparing the returns themselves.

1. Prepare the script for running the computations, and enter the data for received forms (W-2s, 1099s, etc.). It is preferable to do this first rather

than updating the tax form computations, because it allows for partially running the program in advance. This has two benefits as you update the computations: It reveals bugs earlier, and it lets you assess with real data whether it is worth implementing a new computation.

At this step, be sure to review the informational forms in the Appendix and ensure that you have entered all relevant information required there.

2. Update the computations. Start with the main form being computed (e.g., Form 1040), and work line by line through the script. I found it preferable to address each form dependency as it came immediately, resulting in a depth-first update. That is, when the 1040 calls for a result from Schedule B, I would pause work on the 1040 and go update Schedule B. The advantage of this is that you can continue running partial tests of the computations. The main disadvantage is that it becomes hard to keep track of which form you are working on; I generally addressed this by leaving comments in the files as I worked on them.

Once all the computations are done and run without errors, move on to phase 2 and mark the form fields. I found it useful to mark both the forms and the worksheets, even though the worksheets do not need to be filed, because it made it easier to review everything for accuracy.

3. Run the script that fills in all the forms (phase 3). Once you have reviewed the draft return and corrected any issues, you can optimize the return (e.g., try different filing statuses, move dependents around, choose standard versus itemized deductions, and so on). Once you are satisfied, you can print, sign, and file your returns.

3 Object Classes and Data Structures

The tax computation system is written in Ruby and is organized around two main classes: the TaxForm and the FormManager.

3.1 The TaxForm API

Generally, each tax form is represented as an instance of a Ruby class for that form, a subclass of TaxForm. A TaxForm has four essential methods:

name The name of this form. The word "Form" should not precede names.

year The tax year for which this form has been updated. This is used to check whether the form has been updated yet. (An out-of-date form will still be executed; it will just produce a warning.)

compute Compute the form. This method should fill in all the lines of the form.

needed? Whether the tax form is ultimately needed for inclusion in the return. This method should be called after compute, and it should inspect the results of the lines of the form to determine if the form needs to be filed.

A form is essentially a hash table that maps line numbers to data values. Line numbers may be any strings so long as they contain no spaces. Values may be atomic values (numbers, strings, dates), or may be arrays of those atomic values. The order of lines is remembered (and can be altered manually).

3.1.1 Referring to Lines in a Form

The TaxForm has several methods with the name "line" for getting and setting lines:

- line[#] returns the value of that line. If the value is an array or is unset, an error is raised.
- line[#, :opt] returns the value of the line, or if the line is unset it returns BlankZero.
- line[#, :all] returns an array for this line's values. It always returns an array, even if the line only has an atomic value.
- line[#, :sum] returns the sum of this line's values.
- line[#, :present] returns true or false depending on whether the line is set.
- line[#] = x will set the line to an atomic value; it will raise an error if given an array. If the line is already set, a warning will be raised.
- line[#, :all] = [x, y, z] will set the line to an array value.

- line[#, :add] = x will force the line to be an array and then will append the given value to that array.
- line[#, :overwrite] = x will set the line to an atomic value, and will not warn if the line was already set.

As a shortcut, the "method_missing" method for TaxForm will infer line numbers from methods named "line_#".

Every TaxForm is associated with a FormManager. Many of the methods of TaxForm are delegated to its FormManager as a matter of convenience.

3.1.2 Line Name Aliases

A line may be given multiple "names," or aliases, separated with a slash. Each portion separated by slashes becomes an alias for the same line. For example, a line named "8b/agi" would have two aliases "8b" and "agi" that could both be used to refer to the same line.

The purpose of aliases is to provide a stable name for key lines in forms, when the numbering may change across years. As a result, the non-numeric alias is preferable for use as a line name, especially when referring to the line from different forms. A warning will be given when the numeric alias (assumed to be the first alias from the line's full name) is used alone in another form.

3.1.3 How Tables Are Stored in Forms

Consider a form that requires input like so:

Line 1a Business name	**1b Income**	**1c Expenses**
Acme Enterprises	5000	400
Foobar Inc.	10000	
Big Corp.	15000	600

This would be stored in a TaxForm as three lines named 1a, 1b, and 1c as follows:

- Line 1a: ["Acme Enterprises", "Foobar Inc.", "Big Corp."]
- Line 1b: [5000, 10000, 15000]

- Line 1c: [400, –, 600]

It may appear transposed from what one ordinarily thinks of as "lines," but it corresponds better to the line numbering that tax forms tend to use. It also makes for somewhat easier processing since the usual situation is that a form calls for summation over a column, which can be easily obtained (e.g., "line['1b', :sum]").

A method "add_table_row" takes a hash mapping line numbers to values for a single row, and updates each line in a manner that reflects a "row" being added to a table of those lines.

3.1.4 Special Line Names

Generally line names in a form may be any text that has no spaces. But several line names will receive special treatment.

Lines ending with an exclamation mark. Such lines are meant for storing metadata or informative data for a form, when there is no space for entering the data. A common use is to create a line for transferring data to another form. Consider, for example, a worksheet that instructs to copy line 7 to a schedule if line 7 is greater than zero, but to copy line 15 otherwise. Instead of having to put logic into the schedule to choose between copying line 7 and line 15, the worksheet can designate a line "fill!" and the schedule can take the value from that line.

Lines ending with the text "...explanation!". This is for explanations that should appear on a continuation sheet. The value should be an array where the first item is the title of the explanation and the subsequent items are the text of the explanation, in troff format.

Line "continuation!". The value of this line is the name of a form found elsewhere in the output; that form will be attached as a continuation sheet in the form of a table. (Currently, only one form can be attached to a given form as a continuation sheet in this manner.)

Lines ending with "*note". These lines indicate a footnote to a line (e.g., line 15*note would be a footnote to line 15). When the form is filled

in, a footnote mark (currently one or more asterisks) will be placed after the value in the specified line and in front of the value of the line's note.

3.2 The TaxForm File Format

As described above, data for tax computation is presented in TaxForm objects. These objects are serialized to plain text files in the following format. A form begins with a line starting with the word "Form", a space, and the name of the form. Lines of the form follow; they must be indented with whitespace, and the line number should be separated from the value with whitespace. For example:

```
Form W-2
     first_name          John
     last_name           Doe
     a                   123-45-6789
     b                   98-7654321
     c                   Acme Widgets Co.
     1                   50000
     2                   15000
```

Further forms may be included in the same file, delineated with the start word "Form" at the beginning of a line.

The line value may be any of the following:

blank zero A single dash, representing a zero value.

number A number, possibly with a decimal point and possibly led with a minus sign.

date Formatted as mm/dd/yyyy or yyyy-mm-dd. This is converted to a Ruby Date object.

array A list of values, each of which is one of the above types. Arrays may be specified in two forms: A comma-separated list surrounded by square brackets, or another way described below.

text string Anything else. To treat a value as a text string (for example, an ID number starting with a zero), place a single quote in front of it.

An array may be constructed by placing each array element on a separate line in the file, replacing the line number with a double quote mark. For example:

```
A   15
"   18
"   20
```

is identical to "`A [15, 18, 20]`".

When forms are read in from a file, they are instantiated as objects of class NamedForm, even if a class specific to the form exists. This helps to distinguish forms read from input from those that were computed. Generally it should be unnecessary to use any of the methods specific to particular forms on forms read from files, since those forms have already been fully computed.

No Form. If there are no forms of a certain type, the directive "No Form [#]" may be included in the import file. This avoids a warning that is produced when the FormManager tries to find a form that is not present. (The purpose of the warning is to avoid accidental omissions of forms.)

Table. If there are many TaxForm objects of the same type (e.g., Charity Gift), you can save some typing by entering them in a tabular format:

```
Table Charity Gift
    amount   cash?   name
    500      yes     Red Cross
    30       no      Salvation Army
    1000     yes     NPR
```

The last of these columns may contain values with spaces in them; the earlier ones may not.

3.3 The FormManager

The FormManager maintains a complete tax return and manages the forms in that return. Its methods mostly deal with adding, computing, and querying forms.

Accessing forms. A FormManager has a method "forms(name)" that returns an array of forms with the given name. The returned object is an Array delegate that has an additional method "lines" for querying all the lines of all the returned forms.

Submanagers. A FormManager for an entity may contain "submanagers," namely references to other FormManager objects for different entities or tax years. This is a convenient way to draw information from a related filing.

Submanagers are added using the add_submanager method, which requires a name for identifying the submanager. Common names are :last_year and :spouse. The add_submanager_from_file method creates a new FormManager, imports data from the file, and adds a new submanager.

The Interviewer. A FormManager maintains an Interviewer that allows for TaxForms to query the user for information during execution. The Interviewer can also store the responses to a file so that the questions need not be asked again.

I'm generally trying to move away from Interviewer questions, instead putting all information into informational TaxForm structures described in the Appendix.

4 Phase One: Computation

This section describes the operation of the first phase, computation of tax form line items.

The usual process is as follows:

1. A FormManager is created and set up with the relevant parameters, submanagers, interviewer, and so on.

2. Input forms are read using the FormManager's "import" method.

3. The FormManager's "compute_form(class)" method is called to compute a new form. The FormManager creates an instance of the given TaxForm class, adds the instance to the manager, executes that instance's "compute" method, and then removes the form if the "needed?" method returns false. The "compute" method will likely make further calls to "compute_form" if further forms are needed.

4. The computed forms are saved to a file using the FormManager's "export" method.

4.1 When Forms Are Computed

Computations of forms are always invoked by either the main program or another form in the course of that other form's computation. This appears to mimic the IRS's expectations that one will start filling Form 1040 line by line, and figure each schedule and other form as the need arises. In particular, many forms must be computed after some portion of the 1040 is already completed, meaning that they must be invoked in the midst of the 1040 computation.

In some cases, notably the computation of nondeductible IRA contributions, the IRS's instructions are problematically nonlinear and even create circular dependencies. It is useful though unfortunate that the rigorous computations of this system expose these issues.

4.2 Push versus Pull Filling

The IRS instructions often use a "push"-style approach in which a worksheet or form is filled to reach a key number, which is to be then entered or included in another form. For example, the last line of Schedule A is to be entered onto the relevant line of Form 1040.

This program instead follows a "pull" convention in which each form is responsible for gathering information from all relevant forms. In the above example, the code for Schedule A would simply compute its last line without using it, and Form 1040 will query the completed Schedule A to find the last-line value to fill. This convention is consistent with the expectation that each form is responsible for performing all computations necessary to fill the form, which includes invoking computations of other necessary forms. The implementer of the 1040 form would know better whether the deduction line

should be filled with the Schedule A value (as opposed to, say, the standard deduction), so it would be problematic for Schedule A to insert the value onto the 1040 prematurely.

A disadvantage of the "pull" approach is that many lines are computed from multiple and possibly unexpected forms. The tax line on the 1040, for example, includes several different taxes, and the 1040 needs to combine all those values, such as the Net Investment Income Tax. A helpful feature that could be implemented in the future would be to allow one form, such as the NIIT form, to add an assertion to an external form to insure that its values are used.

4.3 Preventing Errors

A key problem for programming tax computation is ensuring accuracy. The large amount of input data is a likely source of errors—mistyped numbers or missed forms—and many unusual rules can arise that require special edge-case handling. A few conventions are used to minimize errors.

Assertions. In many cases, certain conditions are so rare that they are not worth implementing. The computation programs will thus raise assertions if they detect conditions that are not implemented. In some cases, detecting the condition will not be possible on the data alone, in which an interview question is asked to determine that the special condition is not present.

Form checking. For input forms, it is assumed that the user may fail to enter certain ones. As a result, when a computation uses an input form that is not present, a warning is given unless the user has explicitly indicated that there are no forms of that type (using the "No Form" instruction described previously). This checking could be improved, because currently there is not much distinction between input forms (1099s, W-2s) and computation forms (Schedule A, Form 8606).

Line presence. If a computation requests a particular line, the line must be present unless the ":opt" option is given as described above, and it must not be an array unless an array was expected.

5 Phase Two: Form Field Marking

The program that operates this phase is "ui/mark.rb". See the options help on running that script for details.

The script is provided with the following inputs:

- The computed TaxForm file(s), which are used to discern the lines for each form.
- The name of the form to work on.
- The location of a PDF file. The script can predict default URLs for several IRS forms and attempt to download them.
- Which pages of the PDF file to include, useful for worksheets contained within instructions.

The program will automatically save the location of the PDF files and the coordinates of the marked fields to a file, which by default is named "posdata.txt".

5.1 Architecture of the Form Filling UI

Problematically, Ruby is the language in which the other components of this system are implemented, but it lacks good support for PDF processing and user interfaces. Currently I find that Node.js is better for the latter two, because of Mozilla's pdf.js library and Nodegui (which, as of 2021, is still in development but workable). To bridge the gap, the UI script actually runs in two parts: A Ruby wrapper that manages the data files, and a Node-based UI. The two communicate via Unix anonymous pipes, transmitting JSON objects between each other.

If, in the future, Nodegui proves unworkable, it likely would not be difficult either to turn the UI component into an Electron app or to switch to a different framework entirely, retaining the Ruby wrapper as-is.

5.2 Split Lines

One feature that the marking script supports is "split" lines, where a line's text is separated across multiple boxes (for example, one per letter or dividing a social security number across dashes). The "split" checkbox on the UI accommodates this.

Internally, a split line is represented by a flag in the Ruby class Marking::Line. It cannot be an inherited class in order to enable switching a line from split to non-split without creating a new object.

6 Phase Three: Form Filling

To fill in the forms, run the script "ui/fill.rb". Options for the script are provided in the help text.

(Filling the forms used to be much more of an ordeal.)

7 Rethink List

These are architectural aspects of the system that I think I need to reconsider.

Exclusion flag. Right now, if a form is not needed, it is removed from the manager. That's a bit of a problem because a later form may attempt to compute it, meaning that it is computed multiple times. Furthermore, it is probably useful to keep the removed form around (1) to allow other elements of the program to see why it was discarded and (2) to allow the user to confirm that it is not needed.

Retaining the unnecessary form also allows for error checking to make sure that a form is not computed twice improperly.

This should be implementable with just a flag to exclude a form from printing; that flag could be set by default for worksheets perhaps too. But I'd have to think first about the ramifications of including excludable forms, because in some cases the computations rely on the presence or absence of forms in determining what to do.

Hypothetical form computation. Occasionally it is useful to compute a hypothetical form when other data is incomplete. (Form 2441 requires this, for example.) This could be done either by setting some sort of "hypothetical" flag in the manager during these computations (such that any forms produced are only hypothetical) or by making a deep copy of the manager and computing the hypothetical form.

Template forms. There is currently no good, consistent way to ensure that input forms are properly filled, and no good way to show the user what to fill. The documentation below is a halfway solution, but better would be to implement a data description system that would allow for structured filling of those interview-like forms.

8 Wish List

These are features that I'd like to implement some day.

Traceable computations. Rather than storing result data, form lines could store objects that maintain a tree of computations. For example, if line 7 is the sum of lines 5 and 6, line 7 would be presented as an object referencing those two lines and carrying an "add" instruction. This would help with tracing errors and optimizing computations. It will also help with detecting whether form or line information was never used, since that is likely an error.

Sequence numbers. These could most simply be implemented as instance methods on each form class and used to order the output, though that would mean that the sequence numbers are lost after the computation phase. They could also be stored as a line in each form. Or they could be maintained in some external database, though that seems unnecessarily difficult to maintain.

Generalized manager. Rather than having to write a script that computes the 1040, there could be a higher-level form that performs the functions of the script in a more generalized way, among other things computing the 1040. This has the benefit that it could compute ancillary information as a cover sheet, such as a manifest of forms to file, the mailing address, and some summary information about the results.

Better Interviewer questioning. Right now, the questions themselves are the keys for uniquely identifying the questions. This is not great for length reasons. I'd prefer to use short names for the questions, and then have some sort of translation table for presenting the complete question, perhaps with help texts.

Line explanations. I would like to be able to display explanations for each line, to assist in reviewing computation results without having to fill in the forms. This would require some sort of database mapping line numbers to descriptions, either dispersed throughout the form classes or in some unified file.

Forced line aliases. It is possible to make an alias for a line (e.g., "agi" for the relevant line of the 1040), which ensures that other forms remain correct even if the number of that line changes. Currently, a form may reference another form's lines by line number or alias, but the system will issue a warning if an aliased line is reference by number. It would be better if every reference to an external form's lines were by alias rather than number.

Rename managers. The QBIManager class, for example—it would better be named QBIAnalysis.

Push instruction testing. If a form produces a line value that is meant to be incorporated into another form, it would be helpful to have some sort of

assertion that the line is indeed incorporated into that other form. See the section on Push versus Pull Filling.

Appendix

A Additions to Forms

A.1 1095-B, -C

This is the health coverage form. No information other than the items listed below is needed.

hdhp? Whether this plan was a high deductible plan that qualifies for HSA contributions.

coverage "individual" or "family".

months The months of coverage for this plan. (Currently it is assumed that all persons on the form are covered for the same months.) These should be all-lowercase three-letter abbreviations of months, or "all" to indicate all months.

A.2 1098

property The name of the Real Estate form with which this 1098 corresponds.

balance The average mortgage balance on this loan. See Pub. 936. You can compute this by averaging the balances on January 1 and December 31, or you can multiply the interest paid by the lowest interest rate of the year.

A.3 1099-DIV

qexception? If this form shows qualified dividends, this flag must be set to indicate whether an exception applies. See 1040 instructions for line 3a. The exceptions relate to unusual transactions.

A.4 1099-G

payer The name of the payer (usually the state)

A.5 1099-R

2b.not_determined? The relevant checkbox under line 2b.

2b.total_distribution? The relevant checkbox under line 2b.

ira-sep-simple? Whether the checkbox on the form with this name is checked.

destination Where this distribution went. It can be "roth" (currently the only one supported).

B Informational Forms

B.1 Alimony

Alimony received. See Pub. 504.

amount The amount of alimony received.

B.2 Asset

A form for a capitalized asset.

date The date the asset was put into service.

amount The dollar value of the asset when purchased.

179? Whether the asset is a section 179 deductible asset. Generally, tangible personal property and computer software qualify. See Pub. 946 chapter 2.

listed? Whether the asset is a listed asset, see Pub. 946 chapter 5.

dc_category The depreciation category for the asset in DC, see the FP-31 instructions, under Depreciation Guidelines.

dc_type Choose from “reference”, “fixed”, or “other” to categorize the asset for DC Form FP-31, lines 1–3.

description A description of the asset.

B.3 Biographical

Biographical information for an individual tax filer. Spouses should have a separate Biographical form.

whose The individual to whom this form pertains. May be “mine” or “spouse”.

first_name The first name and middle initial.

last_name The last name.

phone Telephone number.

ssn Social Security number, with dashes.

home_address The first line of the home address.

apt_no The apartment number for the address.

city_zip The city, state, and ZIP code.

birthday The person’s birthday.

blind? Whether the person is blind.

occupation Your occupation

B.4 Business Expense

A deductible business expense. See IRS Pub. 535, chapter 11.

date The date of the business expense.

amount The dollar amount of the expense.

category The category of expense. “Meals” and “Utilities” will automatically be halved, the former per the IRS 50% rule and the latter on the assumption that 50% of the expense was for non-business purposes. Other common values are “Supplies”, “Travel”, and “Membership”.

description A description of the expense.

B.5 Charity Gift

A charitable contribution.

amount The dollar value of the contribution.

cash? Whether the contribution was in cash (as opposed to in-kind).

name The name of the charity.

documented? Whether the donation was documented. See Pub. 526. This is checked if the contribution is over the $250 threshold. Currently it is assumed that any documentation meets the requirements of a contemporaneous written acknowledgment (received before filing the return).

For charity gifts that are non-cash that total over $500, Form 8283 must also be filled, which requires additional information on the nature of the gift:

vin The VIN of a donated vehicle.

description A description of the donated goods and their condition. Clothing must be donated in good used condition.

date The date the contribution was made.

date_acq The approximate date on which the property was acquired, or “Various” if the items were acquired on various dates at least 12 months ago.

how_acq How the property was acquired, for example by purchase, gift, inheritance, or exchange.

basis The basis or cost of the property.

method Method for determining fair market value. IRS examples are appraisal, thrift shop value, catalog, or comparable sales.

B.6 Dependent

A dependent.

name The dependent's name.

ssn The dependent's social security number.

relationship The dependent's relationship with the form provider.

qualifying Whether the dependent qualifies for a tax credit. Options are "child" for the child tax credit, "other" for the credit for other dependents, and "none" where the dependent is not qualifying.

B.7 Dependent Care Benefit Use

Information on the use of dependent care FSA benefits.

last_year_grace_period_use The amount carried over from last year's plan that was used this year.

this_year_unused The amount forfeited or carried over to next year.

qualified_expenses The amount spent on qualified dependent care expenses.

B.8 Estimated Tax

Estimated tax payment made to the IRS.

amount The amount of estimated tax paid.

B.9 HSA Contribution

Records of contributions to an HSA account.

ssn The SSN of the beneficiary of the HSA account.

contributions Contributions made to the HSA, but not including employer contributions or HSA/Archer MSA rollovers or HSA funding distributions.

B.10 Home Office

A portion of a home used as an office for a particular business.

type The type of business associated with this home office. Currently the only supported type is "partnership".

ein The EIN of the relevant business.

method Either "simplified" or "actual".

sqft Square footage of the home office area.

daycare? Whether the relevant business was a daycare.

property The name of the Real Estate form with which this home office is associated.

B.11 State Estimated Tax

Estimated tax payment made to a state.

amount The amount of estimated tax paid.

state The two-letter code for the state.

B.12 State Tax

A record of state taxes paid in the relevant tax year (usually for the previous year). This is the amount actually paid to the state tax office, (i.e., the amount due), not the total tax including what was already paid via withholdings.

amount The amount of tax paid.

name A description (not used for computation).

B.13 Traditional IRA Contribution

A contribution to a traditional IRA.

amount The amount contributed.

tax_year The tax year of the contribution.

date The date of the contribution.

B.14 Partner

A single partner within a partnership.

name The partner's name.

liability Either "general" or "limited".

nationality Either "domestic" or "foreign".

type The entity type of the partner. Currently only "Individual" is supported fully. Other possibilities include "Estate". (This should also, in the future, include corporations, partnerships, trusts, nonprofits, and foreign governments, among others.)

share This partner's share of profits and losses, as a decimal fraction such that all the partners' shares add to 1. Currently it is assumed that shares of profits and shares of losses are equal.

capital This partner's percentage contribution of capital to the partnership, as a decimal fraction such that all the partners' shares add to 1.

ssn The SSN of the partner.

ein The EIN of the partnership.

country The country of citizenship of the partner, relevant to Form 1065 Schedule B-1, part II, column iii.

address The first line of the partner's address.

address2 The second line of the partner's address.

active? Whether the partner was active in the partnership's business.

B.15 Partnership

Biographical information for a partnership.

name The name of the partnership.

address The first line of the address.

address2 The second line of the address.

business The line of business of the partnership, Form 1065 line A.

product The product or service of the partnership, Form 1065 line B.

code The business code, Form 1065 line C.

ein The employer identification number of the partnership.

start The start date of the partnership.

accounting "Cash", "Accrual", or other accounting method.

B.16 Real Estate

A description of real property, used in a variety of contexts and to link other forms that deal with a particular property.

property A name for the property that will uniquely identify the property across tax forms.

basis The basis price for the property, i.e., the purchase value. See Pub. 551.

sqft The square footage of the property.

purchase_date The date on which the property was purchased.