RSG
the science of insight

ActivitySim Consortium Updates

MWCOG TFS Meeting

January 26, 2024

Agenda

- ActivitySim Consortium Overview
- Upcoming Release Features
- Future Development

ActivitySim

What Is ActivitySim?

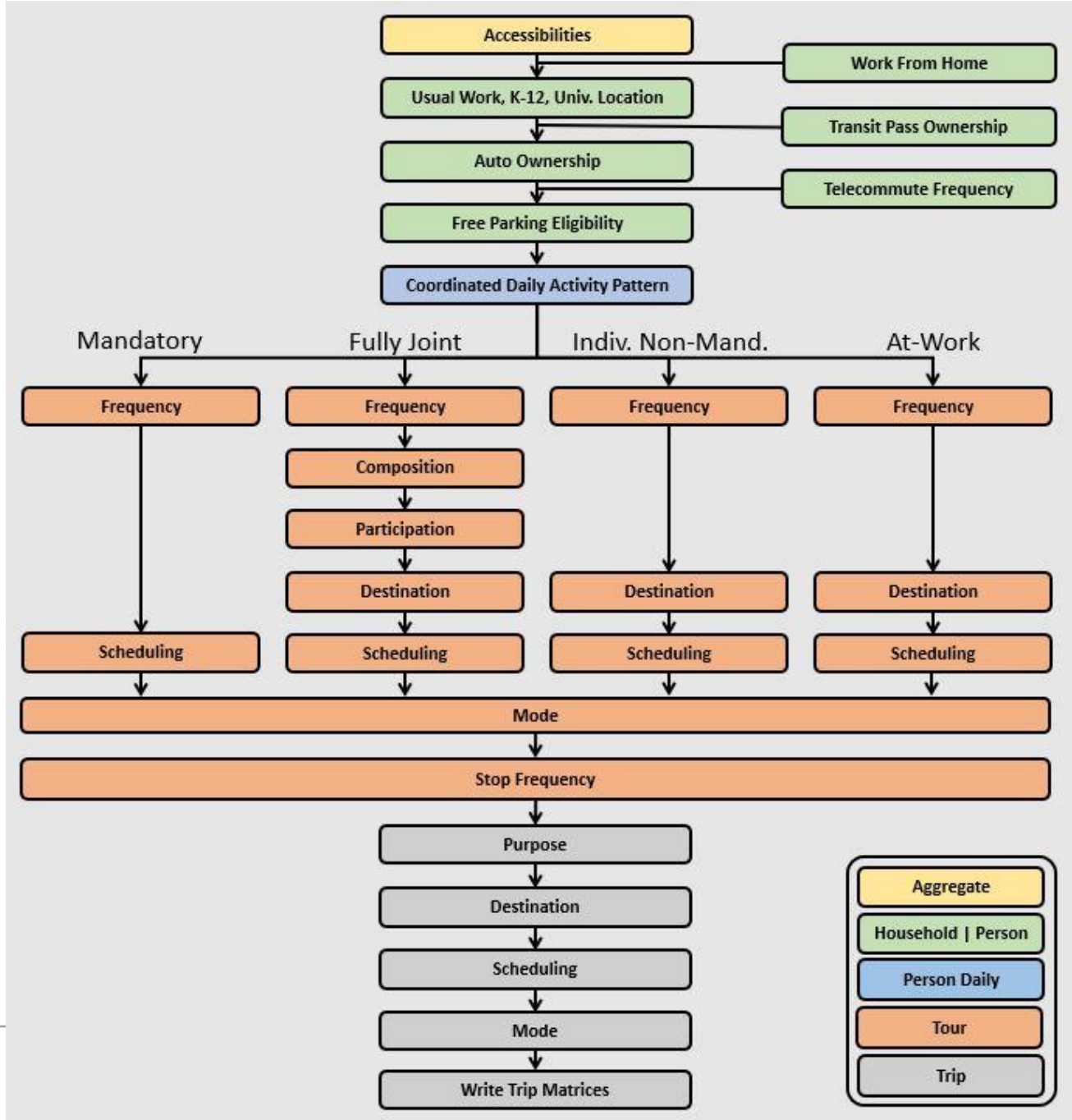
- ActivitySim is…
  - Open source (see code at https://github.com/ActivitySim/activitysim)
  - An Activity-Based Model *framework*
  - Customizable for each user

- ActivitySim Consortium
  - A partnership amongst 14-ish different agencies
  - Partners contribute yearly funds and prioritize development

- ActivitySim Development
  - Consultant bench (RSG, CS, WSP) are primary developers
  - Yearly-ish cycles of development initiatives

What Is ActivitySim?

- ActivitySim framework consists of a set of models that users can build
- Each implementation region will select the models important to them
- Model coefficients and parameters can be re-estimated for individual regions



How does ActivitySim work?

Model settings tell ActivitySim what predictions should be made:

Main settings say what to run:

```
households_sample_size: 1000000
models:
  - initialize_landuse
  - compute_accessibility
  - initialize_households
  - school_location
  - work_from_home
```

Individual models can specify options:

```
SPEC: work_from_home.csv
COEFFICIENTS: work_from_home_coeffs.csv

#LOGIT_TYPE: NL
LOGIT_TYPE: MNL

WORK_FROM_HOME_ALT: 0

DEST_CHOICE_COLUMN_NAME: workplace_zone_id
```

Model specification tells ActivitySim how to make predictions:

Work from home example utility specification:

Description	Expression	Coefficients
Full time worker (1 if true)	@df.pemploy==PEMPLOY_FULL	coef_full_time_worker
Female Worker	@df.SEX==2	coef_female_worker
Accessibility to workplaces of the home mgra	@df.TotalAcc	coef_access_to_workplaces
Age Group - Less than 35 years	@df.age < 35	coef_age_lt_35
Age Group - 35 yrs to 45 yrs	@df.age.between(35, 45)	coef_age_35_to_45
Age Group - 45 yrs to 55 yrs	@df.age.between(45, 55)	coef_age_45_to_55
Age Group - 55 yrs to 65 yrs	@df.age.between(55, 65)	coef_age_55_to_65
Age Group - Older than 65yrs	@df.age > 65	coef_age_65_plus
DC Resident	@df.home_jurisdiction == 0	coef_DC

ActivitySim Phase 8 (2023) Development

Phase 8 Development

- Final code review and merging currently taking place
- Expect a new release in the next month-ish
- Major new features:
  - Input checker
  - Documentation
  - Memory Reduction
  - Upgraded under-the-hood pipeline
- Minor features:
  - Improved trip scheduling algorithm
  - New Joint tour frequency and composition model
  - Faster I/O to disk for checkpointing files

Input Checker

What are the input checker's primary features?

- Ensure input data contains all relevant fields
- The input data fields have acceptable values / ranges
- Make comparison between different columns and different tables
  - e.g. Does the number of people in the household equal household size?
  - e.g. Do all employment fields sum to the total?
- Read in non-ActivitySim data and perform checks
  - e.g. Do all network links have a length > 0?
- Can output both errors and warnings
- Be Fast!
  - Want to be able to run at the start of every ActivitySim run and not add appreciably to the runtime

How does it work? Pandera

```python
class Household(pa.DataFrameModel):
    """
    Household data from PopulationSim and input to ActivitySim.
    Customize as needed for your application.

    Fields:
    household_id: unique number identifying each household
    home_zone_id: zone number where household resides, MAZ in two zone systems, TAZ in one zone
    hhsize: number of people in the household
    income: Annual income in $
    auto_ownership: Seeding for initial number of autos owned by the household
    HHT: Household type, see enums.HHT
    """

    household_id: int = pa.Field(unique=True, gt=0)
    home_zone_id: int = pa.Field(ge=0)
    hhsize: int = pa.Field(gt=0)
    income: int = pa.Field(ge=0, raise_warning=True)
    auto_ownership: int = pa.Field(ge=0, le=6)
    HHT: int = pa.Field(isin=e.HHT, raise_warning=True)

    @pa.dataframe_check(name="Household size equals the number of persons?")
    def check_persons_per_household(cls, households: pd.DataFrame):
        persons = TABLE_STORE["persons"]
        hhsize = (
            persons.groupby("household_id")["person_id"]
            .count()
            .reindex(households.household_id)
        )
        return (hhsize.values == households.hhsize.values).all()
```

```python
households = pd.DataFrame()
validator_class = pandera_checker.Household()
validator_class.validate(households)
```

Input checker will just run the “validate” method that comes with Pandera on the Households object.

Each region / example would have their own set of variables and checks.

Arbitrary checks can be easily made with custom python code

Output Logging

Input checker results are displayed neatly in input_checker.log

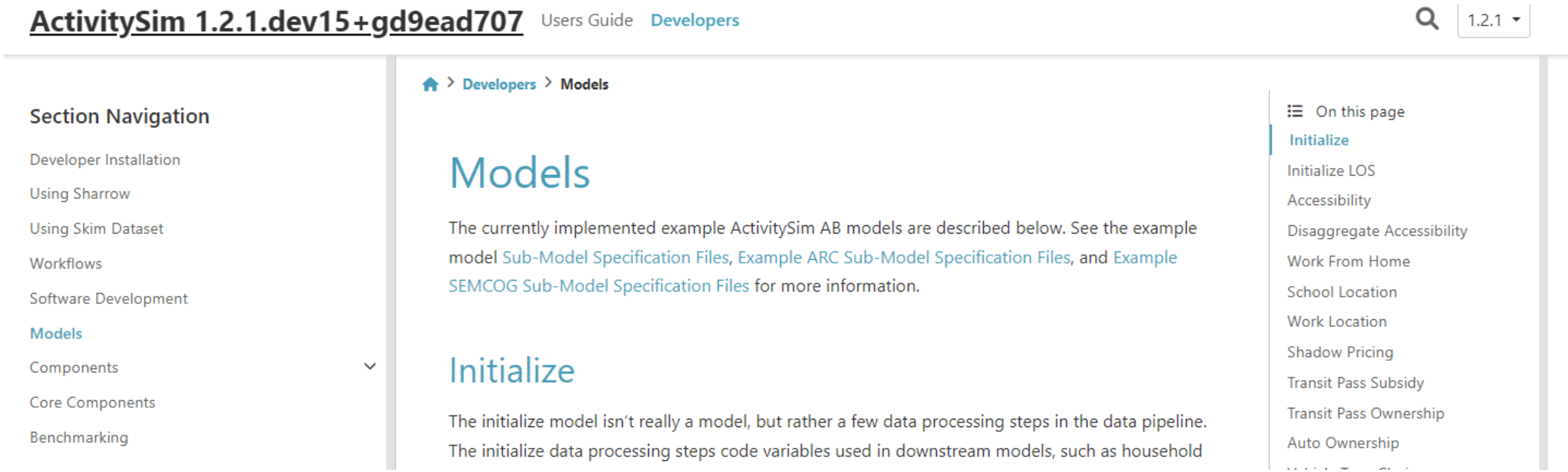
activitysim > examples > production_semcog > test > output > log > input_checker.log

```
Encountered 0 errors and 2 warnings in table households
Encountered 0 errors and 0 warnings in table persons
Encountered 0 errors and 0 warnings in table land_use
Encountered 0 errors and 0 warnings in table Network


##############
  households
##############

households warnings:
--------------------
Failed element-wise validator: <Column(name=children, type=DataType(int64))>
    <Check greater_than_or_equal_to: greater_than_or_equal_to(0)>
    failure cases:
         index  failure_case
    0     14887            -9
    1     14888            -9
    2     14889            -9
    3     14890            -9
    4     14891            -9
    ...     ...           ...
    4586  19543            -9
    4587  19544            -9
    4588  19545            -9
    4589  19546            -9
    4590  19547            -9

    [4591 rows x 2 columns]
Failed dataframe validator: <Check Household children equals the number of child (age<=17) in persons?>



households additional messages:
-------------------------------
Household size equals the number of persons.

All tazes are in landuse file.
```

Documentation

ActivitySim is restructuring its GitHub Documentation

- Allow users to more clearly find what they are looking for



Configuration Settings

ActivitySim settings will now be checked by Pydantic with built-in documentation

- Users can find settings they want to change
- Code will catch unallowed values

field **households_sample_size**: *int = None*

Number of households to sample and simulate

If omitted or set to 0, ActivitySim will simulate all households.

field **checkpoints**: *Union[bool, list] = True* #

When to write checkpoint (intermediate table states) to disk.

If True, checkpoints are written at each step. If False, no intermediate checkpoints will be written before the end of run. Or, provide an explicit list of models to checkpoint.

```
checkpoints: Union[bool, list] = True
"""
When to write checkpoint (intermediate table states) to disk.

If True, checkpoints are written at each step. If False, no intermediate
checkpoints will be written before the end of run.  Or, provide an explicit
list of models to checkpoint.
"""

checkpoint_format: Literal["hdf", "parquet"] = "parquet"
"""
Storage format to use when saving checkpoint files.
"""
```

Under-the-Hood Updates

ActivitySim refactored the underlying data pipeline

- Removed dependency on the Orca python package
- Allows developers much more flexibility for new model designs, data management, CI testing, and code maintenance

Data type improvements to help memory usage

- Strings are changed to Pandas Categorical data
- Seeing 10 - 40% drop in peak memory and run time depending on implementation
  - More testing forthcoming to fully quantify performance gains
- No update to model specification files needed!

ActivitySim Phase 9 (2024) Development

Phase 9 (2024) Development

Primary focus is on performance:

- Want models to run faster with less memory
- Optimize existing model components
- Take full advantage of sharrow

sharrow

The sharrow package is an extension of numba, and offers access to data formatting and a just-in-time compiler specifically for converting ActivitySim-style "specification" files into optimized, runnable functions that "can approach the speeds of C or FORTRAN". The idea is to pay the cost of compiling these specification files only once, and then re-use the optimized results many times.

Other likely upcoming development in 2024:

- Explicit representation of work from home
- Estimation mode enhancements
- Data model development

Questions/Discussion

RSG
the science of insight

David Hensle
Consultant
David.Hensle@rsginc.com