Simpal Users Guide
Version 0.1.0

IllinoisRocstar LLC

October 10, 2016

License

The software package sources and executables referenced within are developed and supported by Illinois Rocstar LLC, located in Champaign, Illinois.The software and this document are licensed by the University of Illinois/NCSA Open Source License (see `opensource.org/licenses/NCSA`). The license is included below.

For more information regarding the software, its documentation, or support aggreements, please contact Illinois Rocstar at:

- **tech@illinoisrocstar.com**
- **sales@illinoisrocstar.com**

0.1 Overview

Simpal is an *IMPACT* service module which provides some basic algebraic operations on DataItems registered with *COM*. Written in C++ using *COM*'s developers interface, *Simpal*'s interface functions take pointers to COM::DataItem objects or scalar numbers as arguments, and should in general be invoked through *COM* using COM_call_function. In the implementation, each process computes on the datasets of its local panes. Most operations implemented in *Simpal* are embarrassingly parallel requiring no inter-process or inter-pane communications.

0.2 Requirements and Conventions

All operations of *Simpal* share similar requirements and conventions on the operands, listed as follows.

- All operands must have the same base data type.
- All nodal or elemental DataItems of an operation must be associated with the same set of nodes or elements.
- At least one of the input operands must be a nodal or elemental DataItem.
- For functions taking two inputs and without the **_scalar** suffix (including **add**(), **sub**(), **mul**(), and **div**()), all input operands must be DataItem objects, and one is allowed to be a panel or windowed DataItem. In the corresponding functions with the **_scalar** suffix, the second input argument must be a scalar number.
- Each DataItem operand has two versions: a scalar version (with only one entry per entity) and a vector version (with only one entries per entity). An DataItem operand can be either one, except that for swap and copy, the two DataItem operands must have the same number of components.
- DataItem operands with more than one data entry per entity (where the entity is a window, pane, a node, or an element for windowed, panel, nodal or elemental, respectively) must have the same number of entries per entity.
- It is legal to use a DataItem with only one entry per entity in place of a DataItem with more than one entry per entity. *Simpal* will use the value of that single entry to compute against with all entries of another operand.
- In most cases, it is legal to mix contiguous and staggered layouts of nodal and elemental DataItems.
- Output arguments are usually the last arguments, except for functions with optional input arguments.

Most of these requirements are checked at runtime by *Simpal* if possible. A violation will cause a run to abort. A user, however, can disable error checking at compile time by passing -DNDEBUG to the C++ compiler.

0.3 *Simpal* Interface

0.4 Supported Operations

The following table lists the operations supported by *Simpal*, in which we use S, W, P, N, and E to abbreviate different types of operands.

S = scalar pointer
W = windowed DataItem
P = panel DataItem
N = nodal DataItem
E = elemental DataItem

$+, -, \times, \div$	$N = N \diamond W, N = N \diamond P, N = N \diamond N;$ $E = E \diamond W, E = E \diamond P, E = E \diamond E$
scalar $+, -, \times, \div$	$N = N \diamond S, E = E \diamond S$
dot	$W/P/N = \langle N, N \rangle, W/P/E = \langle E, E \rangle$
dot-scalar	$S = \langle N, N \rangle, S = \langle E, E \rangle$
2-norm	$W/P/N = \|2\|_2 N, W/P/E = \|2\|_2 E$
2-norm-scalar	$S = \|2\|_2 N, S = \|2\|_2 E$
swap	$N \leftrightarrow N, E \leftrightarrow E$
neg	$N = -N, E = -E$
copy	$N = W, N = P, N = N, E = W, E = P, E = E$
copy-scalar	$N = S, E = S$
axpy	$N = WN + N, N = PN + N, N = NN + N$ $E = WE + E, E = PE + E, E = EE + E$
axpy-scalar	$N = SN + N, E = SE + E$

0.5 *Simpal* API

- void **Simpal_load_mudule**(const char name)
  void **Simpal_unload_mudule**(const char *name)

  Loads/unloads *Simpal* to/from *COM* by creating a service CI window of the given name and register its functions.

- void **add**(const DataItem *x, const DataItem *y, DataItem *z)
  void **sub**(const DataItem *x, const DataItem *y, DataItem *z)
  void **mul**(const DataItem *x, const DataItem *y, DataItem *z)
  void **div**(const DataItem *x, const DataItem *y, DataItem *z)

  Performs the operation $\mathrm{z} = \mathrm{x}\, op\, \mathrm{y}$, where op is $+, -, \times$, or $\div$. The output argument z must be nodal or elemental; one of x and y must have the same type as z, and the other can have the same type or be a windowed or panel DataItem. If all operands are nodal or elemental, then the operation is performed node-wise or element-wise, respectively. If one of the operand is windowed, then its value is used in the operation of every node/element. If one of the operand is panel, then its value in a pane will be uses in the operations on the nodes/elements of this pane. It is legal to have the same DataItem to appear multiple times in the operands.

- void **add_scalar**(const DataItem *x, const void *y, DataItem *z, const char *swap = NULL)
  void **sub_scalar**(const DataItem *x, const void *y, DataItem *z, const char *swap = NULL)
  void **mul_scalar**(const DataItem *x, const void *y, DataItem *z, const char *swap = NULL)
  void **div_scalar**(const DataItem *x, const void *y, DataItem *z, const char *swap = NULL)
  Performs the operation $\mathrm{z} = \mathrm{x}\, op\, \mathrm{y}$ or $\mathrm{z} = \mathrm{y}\, op\, \mathrm{x}$, if swap is null or not, respectively, where op is $+, -, \times$, or $\div$. Here, the scalar behaves similar to a window DataItem in their corresponding functions

without **_scalar**. It is a user's responsibility to ensure that the data type of the scalar is the same as the base data types of the DataItems. Again, it is legal to have the same DataItem to appear multiple times in the operands.

- void **maxof_scalar**(const DataItem *x, const void *y, DataItem *z)
  Performs the operation z = max(x, y). It is a user's responsibility to ensure that the data type of the scalar is the same as the base data types of the DataItems. Again, it is legal to have the same DataItem to appear multiple times in the operands.

- void **dot**(const DataItem *x, const DataItem *y, DataItem *z, const DataItem *mults = NULL)

  Performs the operation $z = \langle x,\ y \rangle$. The inputs `x` and `y` must be nodal or elemental. The output `z` can be windowed, panel, nodal, or elemental. If `z` is windowed, then the result is the dot product for `x` and `y` over the whole window. If `z` is panel, the result is over the each pane. If `z` is nodal or elemental, then the results are over each node or element (i.e., the value associated with a node is treated as vectors).

  The optional argument `mults` specifies the multiplicity of the nodes or elements, i.e., the number of times a node or pane appears in the window. It is useful only when `z` is a windowed DataItem. The product of of the values associated with a node or element will be divided by its multiplicity before being summed. When no value is passed for `mults`, then a multiplicity of 1 is assumed for each node or element.

  Note: An MPI all-reduce is needed for this operation if the solution is a scalar.

- void **nrm2**(const DataItem *x, const DataItem *y, const DataItem *mults = NULL)

  Performs the operation $y = \| x \|_2$. This function works the same as **dot**(), except that it takes only 1 required input argument instead of 2.

- void **swap**(DataItem *x, DataItem *y)

  Swaps the contents of `x` and `y`. `x` and `y` must be nodal or elemental and must have the same number of entries per entity.

- void **neg**(DataItem *x)

  Negate the signs of the values of `x`, where `x` must be nodal or elemental and must have the same number of entries per entity.

- void **copy**(const DataItem *x, DataItem *y)

  Assigns the value of `x` to `y`. The argument `y` must be nodal or elemental. `x` can be windowed, panel, nodal, or elemental. If `x` is windowed, then its value is assigned to every node or element in `y`. If `x` is panel, then each node or element of `y` receives the value of `x` associated with its pane. If `x` is nodal or elemental, then each node or element of `y` receives the value of its corresponding node or element of `x`.

- void **axpy**(const DataItem *a, const DataItem *x, const DataItem *y, DataItem *z)

Performs the saxpy operation $\texttt{z} = \texttt{ax} + \texttt{y}$. x, y, and z must be nodal or elemental. The DataItem a can be windowed, panel, nodal, or elemental.

- void **dot_scalar**(const DataItem *x, const DataItem *y, void *z, const DataItem *mults = NULL)
  void **nrm2_scalar**(const DataItem *x, void *z, const DataItem *mults = NULL)
  void **copy_scalar**(const void *x, DataItem *y)
  void **axpy_scalar**(const void *a, const DataItem *x, const DataItem *y, DataItem *z)

  Has the same semantics as their corresponding version without **_scalar**, except that the scalar argument acts in place of a window DataItem.

0.6 Building and Testing *Simpal*

Simpal is integrated and built as a part of *IMPACT*. *Simpal* comes with a test program named blastest.C in the test subdirectory, and the test is built automatically with *IMPACT*. The test program takes no command-line arguments. Instead, it will prompt for a user to choose interactively the types and layout of operands and the operations to be tested.