1st Draft
State Representation and Data Transfer Component

1. Functionality.

Any implementation of any physics model already includes mechanisms for dealing with inputs and outputs, otherwise it could not function. Many physics models also have *state*, i.e. dependence on the results of prior iterations or time steps. Historically, software implementations of physics models have dealt with input/output, and state if necessary, each in their own way.

During the design and implementation effort leading to the creation of tokamak transport code modules in the NTCC project [http://w3.pppl.gov/NTCC] it was noted that there exists a great deal of commonality of form of the constituent parts of the input, output, and state datasets associated with the various tokamak oriented physics models and their code module implementations. Although details of representation varied from model to model, it was found that mappings between representations could be defined either by direct interpolation or by interpolation after a coordinate transformation mediated by the plasma flux surface geometry, i.e. the plasma MHD equilibrium.

The commonality of requirements across physics models motivates the design and implementation of a shared component, a module with the following functionality:

- Representation of tokamak MHD equilibrium in a standardized form.
- Implementation of common coordinate grids and mappings between real space and magnetic coordinates e.g. $(R,\phi,Z) \Leftrightarrow (\varphi,\phi,\theta)$ as defined by the equilibrium.
- Ability to define, label, store and retrieve named coordinate grids in velocity space as well as real space; allow multiple griddings of the same coordinate to be associated with each other.
- Ability to define, label, store, retrieve and interpolate sets of gridded profiles defined with respect to any combination of real space or velocity space grids.
- Ability to evaluate derivatives of profiles up to the order supported by the chosen interpolation method.
- Ability to evaluate certain common integral operators on combinations of profiles and/or grid coordinates, such as flux surface averages.
- Provision of simple methods for saving datasets to or retrieving datasets from long term storage.

1.1 Mathematical Description.

In order to describe the role of the State Representation and Data Transfer Component, hereafter referred to as the *SDC*, it will be useful first to introduce definitions of two

types of physics model components. The *stateless component* describes software which produces an output dataset $\{O_k\}_{k=1}^{N_O}$, i.e. a collection of scalars and/or profiles, whose content are determined solely by an input dataset, a collection of scalars, profiles, control variables, reaction rate database references, etc., denoted as $\{I_j\}_{j=1}^{N_I}$. It can be described in operator notation:

$$\mathbf{C}:\{I_j\}_{j=1}^{N_I} \rightarrow \{O_k\}_{k=1}^{N_O} \tag{1}$$

Stateless components typically deal with phenomena that are considered to be on a fast timescale relative to some specified context. For example, in the context of a typical tokamak transport code, RF wave field solvers, radiative transport, neutral beam deposition and neutral gas transport models are stateless. These models depend only on their inputs. Evaluation of the MHD stability of a plasma configuration is usually stateless. On the other hand, evaluation of the non-thermal fast ion response to RF wave field, and/or the accumulation of fast ions due to beam injection or fusion reactions, and the response of the target plasma itself to heating and current drive, these all happen over longer timescales and their implementations as physics models generally have state.

The *stateful component* describes the software implementation of a physics model which produces an output that depends not only on the inputs (provided from outside the model) but also on its state, as developed internally during previous iterations or calls to the model. Typically this happens in a model which is explicitly tracking the time dependence of "slowly evolving" quantities. Assuming this is the case, the model goes through a series of states $\mathbf{S}_i = \{S_l\}_{l=1}^{N_S}$, i.e. a series of datasets each consisting of a known collection of scalars and profiles with evolving values, associated with a sequence of physical times t_i. Each state is an *initial condition* for calculation of the subsequent state. Then, the stateful component can be described in operator notation:

$$\mathbf{C}:\begin{cases} t_i \rightarrow t_{i+1} = t_i + \Delta t \\ \left[\mathbf{S}_i, \{I_j\}_{j=1}^{N_I}, \Delta t\right] \rightarrow \left[\mathbf{S}_{i+1}, \{O_k\}_{k=1}^{N_O}\right] \end{cases} \tag{2}$$

An *integrated physics simulation* can be composed of components by arranging for the execution of an ordered sequence of operators $\{\mathbf{C}_1, \mathbf{C}_2, \mathbf{C}_3, ..., \mathbf{C}_N\}$, each stateless or stateful as the case may be, each representing a self contained piece of physics, and all involved in the time step advance of the integrated simulation. The decomposition of the simulation into components allows each component to be developed independently. But, the components must share inputs and outputs. By providing the State Representation and Data Transfer Component, the SDC, a separate module to act as a repository for data communications across physics components, the communications overhead associated with the simulation can be reduced from $O(N^2)$ to $O(N)$, because each component interacts just with the SDC, rather than every component having to interact with every

other component. The SDC manages an evolving dataset $\{D_i\}_{i=1}^{N_D}$ in which each of the individual component input and output datasets are subsets.

It might appear that state datasets could be handled separately. To the extent that state data is truly private to a single component that is indeed the case. In practice however, it is not unusual for there to exist significant overlap between components' state and output datasets; for example, a plasma fluid component incorporating a density prediction model would probably need to consider the electron density n_e as both output and part of its state. Therefore, the unification of the data transfer and state representation capabilities into a single component often makes sense. On the other hand, some components may require portions of their state to be represented in very specialized ways which are truly private; in such cases it might be better to have the component manage these portions of its state internally.

In practical code development efforts, and in many operational/production environments as well, the ability to *save* state to long term storage is crucial. This is the mechanism by which a simulation is made *restartable*. Not only does this allow recovery from systems or hardware failures, such restartability greatly aids debugging, since the reproduction of a failure mode and testing of repairs can be carried out starting from an intermediate state of the calculation, rather than requiring rerunning from the start each time. In large multi-component simulations with the potential for complicated failure modes, such capability, and the ability to debug separate components in isolation, is indispensable.

The unification of the intercomponent communications in the SDC might make an eventual distributed deployment of an integrated simulation simpler—any protocol for network data transfers could be isolated within the SDC component, rather than having to handle these separately in each physics component.

At this point it must be noted that a complete description of the SDC requires that all possible forms of elements of the shared data elements in the set $\{D_i\}_{i=1}^{N_D}$ be specified. As this depends on a specification of all the forms of all the inputs and outputs of all physics components, it cannot be done in an isolated description of the SDC alone. However, a plausible collection is provided here:

- Lists of labeled scalars having integer, character string or floating point values. Example: a list of plasma species each with a label, a charge Z and an atomic mass A.
- Coordinates grids, each a labeled monotonic increasing set of numbers spanning a specific range of values. Examples: major radius R, toroidal angle ϕ, radial magnetic flux coordinate φ, poloidal angle coordinate θ …
- Gridded profiles defined as splines over one or more of the above coordinates, e.g. electron density $n_e(\varphi)$, beam injected fast ion density $n_b(\varphi,\theta)$.

- Some profiles have reserved status, e.g. a certain collection of profiles $\left\{\varphi(R,Z),\left\{R_s(\varphi,\theta),Z_s(\varphi,\theta)\right\},q(\varphi),g(\varphi)\right\}$ as an axisymmetric tokamak MHD equilibrium (in this case, an "inverse" equilibrium representation).
- Coordinate transformation methods based on the MHD equilibrium.
- Magnetic flux surface integrals and averages as needed by components making use of the SDC.
- Velocity space grids.
- Profiles over both real- and velocity space grids.
- Specialized data structures known to and shared by multiple components. Possible examples: a vacuum vessel geometry description, an RF antenna geometry description.
- Opaque structures, e.g. a block of known size of integers or floating point numbers for which the SDC merely provides storage and retrieve capability.

This list could become quite lengthy, and may be a hazard to the health of SDC component software.

Successful project-wide *use* of an SDC component will likely require adoption of certain project-wide standards, such as:

- Physics units standards (MKS/KeV) for component i/o.
- Naming convention for component outputs, since the SDC will expose a globally shared namespace.
- Some agreement or standardization on definition of magnetic flux coordinates.
- Possibly, agreement on definition of velocity space coordinates as well.

1.2 Code for Initial Implementation of the Component.

XPLASMA

The XPLASMA NTCC module (component) http://w3.pppl.gov/ntcc/Xplasma/ has many but not all of the features described as required for the SDC component.

XPLASMA is currently in production use; it is the main mechanism for communication of inputs and outputs between TRANSP and its existing constituent RF and neutral beam source model components, such as the NUBEAM fast ion module which is also available at the NTCC website. XPLASMA is used in the communication of numerical equilibria to the TORIC ICRF full wave solver, both within TRANSP and in offline applications based on EFIT or other equilibrium sources. XPLASMA is also used to communicate data to/from NUBEAM from/to the ONETWO plasma transport simulation code at GA. As the software has been available as a downloadable NTCC module for several years now, there may be other applications of which the authors are unaware.

The full XPLASMA module interface is described at the NTCC website; only examples of use are shown here. Although written in fortran-90, fortran-77 argument types are used, and a C/C++ callable interface exists.

When the XPLASMA software is initialized, it is empty of data. As named data items are added, each receives a unique integer identifier. Names are case-insensitive (like fortran variable names). For efficiency reasons (i.e. to avoid the cost of repeated name translation and lookup operations), most XPLASMA subroutines require data references to be specified by integer identifier. Lookup routines provide the name to identifier translations; reverse lookups (i.e. recovering the data item name from its integer identifier) are also available:

```
integer :: id_q,id_rho        ! data item ids are integers
character*32 :: zname         ! names up to 32 characters long

call eq_ganum("__RHO",id_rho) ! retrieve radial flux coord ID
call eq_get_aname(id_rho,zname)     ! reverse lookup

call eq_gfnum("q",id_q)       ! retrieve ID for q profile
call eq_get_fname(id_q,zname)       ! reverse lookup
```

Although the XPLASMA interface was coded with 3d MHD geometries (i.e. stellerators) in mind, only the 2d (axisymmetric tokamak) capability is implemented, so, there are no profiles over any toroidal angle coordinate such as ϕ.

MHD equilibria can be loaded from various experimental data sources. The subroutine call:

```
call eqm_fromgeqdsk(<EFIT-data-path>,…)
```

enables an immediate load of an experimental equilibrium from an EFIT "g" file or MDS+ tree. Equilibria can also be loaded from TRANSP output files or MDS+ trees, although an additional TRANSP-related NTCC software module (TRXPLIB) needs to be linked. Calling an equilibrium load subroutine causes a standard set of named grids and profiles to be defined in XPLASMA.

Although not strictly enforceable by XPLASMA, all currently known XPLASMA users have adopted MKS modified with "KeV" for temperatures as the physics units standard for communication with XPLASMA. The radial flux coordinate grid **"__RHO"** is normalized, always covering the range $[0,1]$ with the definition:

$$\rho = \sqrt{\frac{\Phi}{\Phi_{bdy}}} \tag{3}$$

This is the square root of the normalized toroidal magnetic flux enclosed by each flux surface. It should be noted that when starting with a free boundary equilibrium such as is provided by EFIT, XPLASMA generally has to define the boundary "in somewhat" from

the precise separatrix-defined last closed flux surface, so that the boundary surface is "smooth enough" to be used by all affected models. The magnitude of this offset is controlled by arguments to **eqm_fromgeqdsk**.

Although the procedure to set up the equilibrium representation caused **"__RHO"** to be defined with a particular discretization, alternate discretizations, which need not be evenly spaced, can easily be defined and associated with ρ, but, each discretization is a strict increasing sequence of numbers covering the exact range $[0,1]$. Let N_ρ be the size of one such alternate discretization. If the integer fortran variable **NRHO2** has value N_ρ, the alternate grid id is **ID_RHO2**, and a (real*8) floating point array **DATA(1:NRHO2)** defines a profile that needs to be added to the current XPLASMA set, then the following fortran call will do it:

```
call eqm_rhofun(ICUBSPL,ID_RHO2,"my_profile",DATA,…,MY_ID,IERR)
```

The leading integer argument is a parameter with value (2) which specifies that the profile should be represented as a cubic spline; this implies that boundary conditions are needed (*not shown here* but explained in the NTCC module documentation). Piecewise linear or Akima-Hermite interpolation are also available as options; splines are continuously twice differentiable, hermite functions continuously once differentiable, and piecewise linear functions merely continuous. The interpolation method is defined when the profile function data item is created. If the operation is successful, **IERR=0** and a positive profile **MY_ID** value is returned; if the operation fails, e.g. if **ID_RHO2** were incorrectly specified, **MY_ID=0** and **IERR>0** would be returned, and an explanatory message written (by default to **stdout** but can be redirected).

Similar methods exist for defining profiles vs. 2d magnetic flux coordinates or 2d axisymmetric cylindrical coordinates, i.e. (ρ,θ) or (R,Z).

Suppose that a particular physics simulation calculates the trajectory of a straight line in cyclindrical coordinates, discretizes this into **NSL** data points, stores the results in **RSL(1:NSL)** and **ZSL(1:NSL)**, and needs to lookup the value of the **MY_ID** profile at each of these points. Since **MY_ID** defines a profile over the magnetic coordinate ρ, a preliminary coordinate transformation is needed. The XPLASMA code sequence will look something like this:

```
Real*8 PHI(1:NSL),RHO(1:NSL),TH(1:NSL),RESULT(1:NSL)TOL
Integer :: REGION(1:NSL),IDERIV,IERR)

PHI(1:nsl) = 0    ! toroidal coordinate ignored: axisymmetry
TOL=1.0d-7  ! lookup tolerance
call eq_inv(NSL,RSL,ZSL,PHI,RHO,TH,REGION,IERR)  ! R,Z->RHO,TH

ideriv=0    ! just fetch profile value, not derivative…
call eq_rgetf(NSL,RHO,MY_ID,IDERIV,RESULT,IERR)  ! eval f(rho)
if(ierr.ne.0) write(6,*) " error!"
```

Note that this sequence involves evaluation of a *vector* of coordinate transformations, followed by a vectorized spline profile evaluation. (Because of caching effects, vectorization significantly speeds performance here, even on commodity hardware). Each coordinate transformation involves a Newton iteration to find the root of the nonlinear 2x2 system:

$$\begin{bmatrix} R(\rho,\theta) - R_{in} \\ Z(\rho,\theta) - Z_{in} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \tag{4}$$

It is possible that some of the pairs (**RSL(j),ZSL(j)**) are beyond the core plasma boundary, in which case **REGION(j)=2** and **RHO(j)=1.0** were set, and the eq_rgetf call would return the outer boundary value of the **MY_ID** profile.

The above example shows interpolation of a profile after a coordinate transformation. In many cases, however, all that is needed is a remapping of a profile from one ρ grid to another. In this case, eq_rgetf can be used, with the target grid specified in the arguments.

XPLASMA was originally designed as a data mapping or interpolation tool, and not specifically as a tool for saving and restoring the state of a calculation. Nevertheless, the entire contents of an XPLASMA configuration, i.e. all of its stored data elements, can be saved "to disk" in a single call:

```
call eq_save(<filename>,ierr) ! save XPLASMA state, ierr=0 if OK
```

Conversely, a single call restores an XPLASMA state (but be careful this *overwrites* any XPLASMA state previously in memory):

```
call eq_restore(<filename>,ierr)    ! restore XPLASMA state
```

Then, if the calculation's state consists of a collection of profile functions $\{ f_j(\rho) \}$ over some particular ρ grid, then its state can be restored from the XPLASMA data by first retrieving the precise grid:

```
call eq_ganum(<gridname>,id)
call eq_ngrid(id,<grid-size>)
allocate(rho_grid(<grid-size>),…)
call eq_grid(id,rho_grid,<grid-size>,<grid-size-confirmed>,ierr)
```

and for each gridded profile function $f_j(\rho)$:

```
call eq_rgetf(<grid-size>,rho_grid,id_fun,0,fun_data,ierr)
```

which will return precisely the same values that went into the original eqm_rhofun call to set up $f_j(\rho)$, provided the profile was in fact originally defined over the specified grid.

In the TRANSP support environment, test programs for NUBEAM and for RF components are run routinely to track down problems that occur at runtime during production operations (1000s of TRANSP experimental data analysis runs are executed by a geographically dispersed group of scientists every year; most of these use NUBEAM and/or various RF source calculations).

For the reduced test program to work for debugging, precise bit-for-bit reproduction of the failed calculation from the TRANSP time step must be achieved. In the case of NUBEAM, this is done with a hybrid procedure which uses XPLASMA for exact regeneration of the input profiles and MHD equilibrium, and a NUBEAM private state file which contains NUBEAM-specific controls as well as the fast ion distribution as represented by a list of Monte Carlo test particle weights, positions and velocities. Provided the test program is running on compatible hardware and is built with the same compilers, the bit-for-bit reproducibility works. It is foreseeable that these kinds of capabilities will be needed by the CSWIM FSP project.

2. / 3. Ideal SDC Interface / XPLASMA Implementation.

In the following bulleted list, an interface to support SDC functionality (as defined in the prior section) is sketched in an approximate manner, and this is compared with what actually exists in the XPLASMA implementation (as of 9 December 2005). At the end of this section, potential limitations of XPLASMA relative to an ideal SDC are described.

- Lists of labeled scalars having integer, character string or floating point values. Example: a list of plasma species each with a label, a charge Z and an atomic mass A.
    - Ideal Interface: methods to *store*, *retrieve*, and *modify* lists.
    - XPLASMA: *store* and *retrieve* OK, *modify* easily added. Each list has a name (32 characters), label (60 characters), and size (number of elements). Each list element consists of its own name, label, an integer scalar, and a floating point scalar. Other definitions of "list" are readily imagined.
- Coordinates grids, each a labeled monotonic increasing set of numbers spanning a specific range of values. Examples: major radius R, toroidal angle ϕ, radial magnetic flux coordinate φ, poloidal angle coordinate θ …
    - Ideal interface: store and retrieve grids associated with specific coordinates; allow multiple griddings of the same coordinate to be grouped together.
    - XPLASMA: supports these capabilities for axisymmetric tokamak configurations: (ρ,θ) and (R,Z).
- Gridded profiles defined as interpolating profiles over one or more of the above coordinates, e.g. electron density $n_e(\rho)$, beam injected fast ion density $n_b(\rho,\theta)$.
    - Ideal interface: store, retrieve, modify, and interpolate profiles and their derivatives.

    - XPLASMA: supports these capabilities for axisymmetric tokamak configurations: (ρ), (ρ,θ) and (R,Z).
- Some profiles have reserved status, e.g. a certain collection of profiles $\{\varphi(R,Z),\{R_s(\rho,\theta),Z_s(\rho,\theta)\},q(\rho),g(\rho)\}$ as an axisymmetric tokamak MHD equilibrium (in this case, an "inverse" equilibrium representation).
    - Ideal interface: create standardized equilibrium representation from representations commonly in use; modify equilibrium.
    - XPLASMA: supports EFIT internally; TRXPLIB extension supports TRANSP; I2MEX extension supports JSOLVER, CHEASE.
        - *Ability to modify equilibrium "in place" needs to be improved*.
- Coordinate transformation methods based on the MHD equilibrium.
    - Ideal interface: accurate transformations between Cartesian, cylindrical, and magnetic coordinates.
    - XPLASMA: supports such transformations.
- Magnetic flux surface integrals and averages as needed by components making use of the SDC.
    - Ideal interface: ready interface to all flux surface averages typically needed by plasma simulators, on any specified ρ grid.
    - XPLASMA: supports a large set including all known flux surface averages needed by NCLASS and other neoclassical tokamak transport models.
- Velocity space grids.
    - Ideal interface: treatment equivalent to spatial coordinate grids.
    - XPLASMA: *this is not implemented*.
- Profiles over both real- and velocity space grids.
    - Ideal interface: store, retrieve, modify, and interpolate profiles and their derivatives.
    - XPLASMA: *not implemented for general profiles with velocity space dependency*. A particular representation of fast ion distributions is supported however (see next bullet).
- Specialized data structures known to and shared by multiple components. Possible examples: a vacuum vessel geometry description, an RF antenna geometry description.
    - Ideal interface: store, retrieve, modify.
    - XPLASMA: Monte Carlo fast ion distributions $f(E,v_{pll}/v,\rho,\theta)$ (computed by NUBEAM) are handled by this method.
- Opaque structures, e.g. a block of known size of integers or floating point numbers for which the SDC merely provides storage and retrieve capability.
    - Ideal interface: store, retrieve.
    - XPLASMA: *not implemented*, but easy to do.

General discussion of XPLASMA limitations with respect to an ideal SDC capability.

Another requirement is to be able to save/retrieve SDC datasets to/from long term storage. Although well supported by XPLASMA, the method used will not scale to high volume datasets. The method is oriented to a disk file on a single system. Considerations of high volume datasets and/or distributed computing, such as are likely to be required by FSP at some point, will require reworking of the XPLASMA long term storage or network communications interface and implementation.

As has been mentioned previously, although the XPLASMA interface was designed with non-axisymmetric plasma configurations in mind, only the support for axisymmetric configurations has actually been implemented.

The interpolation capabilities of XPLASMA are built over the NTCC PSPLINE library. The library supports gridded interpolating functions of up to three scalar coordinates only. Some distribution function representations may exceed this limit. There is no easy fix. For this reason, the 4d NUBEAM distribution function $f(E, v_{pll}/v, \rho, \theta)$ is treated as a specialized data structure.

Relative to likely use of an ideal SDC, XPLASMA has historically been used primarily for intermodule communications, less so for managing persistent state. Its typical usage (e.g. within TRANSP time dependent transport simulations) takes the following course:

1. MHD equilibrium is updated. New XPLASMA instance is created; equilibrium stored.
2. TRANSP adds list data and plasma parameter profiles to XPLASMA.
3. TRANSP calls NBI or RF module with reference to XPLASMA data.
4. NBI or RF module computes new profiles and adds these to the XPLASMA.
5. TRANSP reads NBI and RF results, uses these as sources to advance the transport and current diffusion equations.
6. TRANSP discards current XPLASMA instance and moves on.

In other words, TRANSP itself (not XPLASMA) is "in charge of" managing the state of its time dependent simulation. It writes its own check-point restart files, as it has done since long before XPLASMA was created.

XPLASMA files are routinely used to store "transient states" such as the input datasets for the next module or component calculation. Should such a calculation fail, and this is not unusual as research codes are involved, the XPLASMA file is typically part of a larger set of files that allow the component failure to be accurately reproduced in a standalone context for debugging.

However, XPLASMA has not acted in the capacity of a primary state-save / state-restore for an integrated time dependent simulation. It will require some re-engineering to

support this—mainly to make it easier to update its contents, as opposed to discarding and starting from scratch over each time advance cycle.

It should also be noted that XPLASMA as currently coded is based on a static module—multiple instantiations of XPLASMA datasets cannot currently coexist within a single process. XPLASMA internals could be reengineered to support a more object oriented implementation, but a significant effort would be required.

Some physics components may use advanced data structures such as unstructured triangulated meshes. These are not known to XPLASMA.

4. Expectations about input data.

The set of data objects to be supported by the SDC needs to be defined by the project. This definition can evolve as the project evolves, but a coding effort will be required to add each new type of data object. Some "restraint" is advisable; any attempt to define SDC as a completely general repository for any conceivable mathematical data structure will likely yield an unmanageable result. In all likelihood, constituent components of the CSWIM FSP will need some specialized elements of "private state" that are supported at the component level, not at the SDC level.

5. Shared infrastructure requirements.

The SDC *is* shared infrastructure. It will depend on some subsidiary components or libraries e.g. to implement standard interpolation capabilities. These subsidiary components would seem to be readily available, although this could change as the range of data objects supported by SDC expands.

6. Development needs; distributed or parallel computing considerations.

Further SDC development requirements will be derived from the considerations in section 2/3; much depends on the set of data objects required to be supported, and on the total volume of data ultimately arrived at. This will become clear as the CSWIM FSP evolves.

XPLASMA uses NetCDF for its underlying communications with long term storage. It has been linked to a distributed version (LORS NetCDF) that could allow an XPLASMA data resource to be shared across multiple machines over a network. The semantics would be: multiple readers, only one process at a time with "write" access. Some mechanism for passing write access from process to process would need to be invented.

Good performance with LORS NetCDF will require reengineering of XPLASMA input/output internals, to better enable access to XPLASMA data “piece by piece”, so that only pieces that are needed are drawn across the network.

XPLASMA has no support for parallelized input/output at this time. There is no known specification for such a requirement at present.

7. Computational and memory requirements.

The computational requirements of the SDC are modest. It is basically an input/output component. Memory requirements could be substantial, but these are essentially the memory requirements of the constituent components of the CSWIM FSP.

Although the SDC will necessarily encapsulate some computational capabilities that are rather specific to plasma physics and/or the integrated plasma simulator application, the SDC does not present severe challenges in its own utilization of mathematical physics. The challenges for the design and implementation of the SDC are primarily in the realms of computer science and software engineering.

The author (Doug McCune) has some concern about his ability to extend XPLASMA to support the full SDC requirement without help. He would like particularly to enlist the aid of the Computer Science partners of the CSWIM FSP collaboration, if a specific path forward can be identified.