Introducing
Project Organization Stabilizing Tool (POST) system *ReviewedRelease v1*
for evolving order and stability from innovation in chaotic environments

Loewe et al. (2016-09-12) Brief title "**POSTsystem *RRv1***"
Citation: http://dx.doi.org/10.1111/nyas.13192 – Updates: http://evolvix.org/post

Abstract

Organizing is the art of avoiding inessential complexity. Organizing becomes more difficult with project size, as increasing numbers of moving parts tend to throw everything into disarray more quickly. Good abstractions can enable amazing efficiencies by balancing the needs of standardizing and customizing, but they can be (too) costly to find. Much of this cost is related to naming, which in turn is closely related to organizing. Names are made to cut search times by labeling boxes of organized content. Poor naming strategies result in poor organization and increased search times. The quality of a naming/organizing strategy shows as numbers of items increase. Almost no strategy is needed for a few items (tempting many to regard naming as trivial), but finding appropriate strategies for millions of items or more is usually difficult (some say impossible). The lack of sensible strategies can frustrate much further research, a research potential that can be unlocked by the proposal of a workable nomenclature (i.e. naming strategy; see, for example, how modern biology was impacted by the organization and names proposed by Linnaeus' taxonomy).

Cancer cell biology, evolutionary systems biology, and many other areas of current biology are in great need of naming/organizing the millions of parts required for mechanistic computer simulations of relevant biological systems that incorporate all current knowledge. Many diverse ontologies, taxonomies, databases, genome-browsers, models, tools, simulations and other projects have made great progress in consolidating the dizzying jungle of synonyms in biology. However, their independent origins have also generated a new jungle of (near-)synonyms, as each project tends to use idiosyncratic ways of handling equivalent tasks, such as tracking bugs, tasks, reliability, versions, recurring types of modifications and more. These and many more inessential differences make it near impossible to develop programs that treat the scattered big data of biology in a uniform way. Most importantly, there is no universally agreed stability scale that enables researchers from distant fields to easily track the reproducibility, maturity, and reliability of a given result as assessed by relevant experts. A similar stability scale is also essential for developing programming languages aiming to respect the time of their users by not breaking the code of programs when releasing the next version (i.e. by providing long-term backwards compatibility).

The need for such a high-quality stability scale in the modeling language Evolvix inspired the development of the POST system presented here. It uses the BEST Names concept that disentangles conflicting naming priorities by distinguishing Brief, Explicit, Summarizing, and Technical (BEST) Name Dialects. Most Brief Names in POST are double-capital letters such as *RR*, *SS*, and *TT* that mirror Explicit Names like *ReviewedRelease*, *StableSource*, and *TrustedTested* respectively, where *TT* marks the difficult to achieve long-term backwards-compatible end of the POST sequence of stability levels that starts with *MM* for *MockupModel*. The rest of POST evolved around this *StabilizingZone* in order to provide the project organization necessary for the development of the Evolvix programming language and to support the simulation system it implements.

... from so simple a beginning endless forms most beautiful
... are being evolved.[1]

1. Preface[1]

It is a truth universally acknowledged, rarely pure and never simple: managing complex projects is … complex. Surprisingly, none of them start that way. Even the most complex projects are born from simple, elegant ideas in the mind(s) of their initiators before they begin to grow and evolve.

As humans, we generally abhor (ugly) complexity and admire (elegant) simplicity. Thus we will continue to start new projects as we try to 'stand out' while 'blending in', and 'be extraordinary' while 'remaining normal' (defined in as many ways as people exist on the planet).

These tensions are not new, and those before us have pioneered two response strategies: *standardize* and *customize*. *Standardizing* helps with simplicity, blending in, and compatibility to others, while *customizing* enables extraordinary feats of outstanding innovation – albeit at the cost of having to tolerate some inessential and annoying complexity.

Take cars, for example. Most people want a car (*standardize*), usually one that is different from others (*customize*). The pattern continues for driving behavior: roads require driving on the right side (which-ever that may be; *standardize*), while most people are free to choose which road they take (*customize* their journey). Our social contracts to pre-decide on which side of the road to drive are not imposed on us by the physics of cars; yet they have huge practical and technical consequences that simplify and speed up many decisions, save countless lives – and even affect cars physically.

We suggest that parallels exist to navigating multiple complex information-based projects efficiently. When working with multiple projects, it cannot be efficient when each project stores its tasks in a different location or has a different system for indicating stability. The POST system distills the best ideas for standardizing the organization of complex information processing projects of any kind.

---

[1] **Who may want to consider using POST?** We know from experience that most researchers, developers, and organizers are right in *not* looking for systems that can manage multiple complex projects; mostly, because what they already use works well for them and searching for better systems would waste much time (see below for reasons). The POST system was developed because of very real needs encountered in developing Evolvix; we realized only later that POST is much more widely applicable, more flexible, and easier to use than we had thought (and most people suspect). To highlight this generality, we added this broad introduction that explains how POST could be used efficiently for very many information based projects (see text up to example in Figure P1). However, we do not wish to imply that all such projects would benefit from switching (potentially at great cost). POST was designed for complex networks of interacting projects that have much to win from standardizing (see the computational biology use-cases described), and it may also get used by those who like the POSTcode names we found; but *if there is no need for improving coordination, there is little need for considering the POST system presented here.*

Why would most people be wasting time when evaluating systems for managing multiple complex projects? Such searches tend to be too tedious for anybody who wants to get started with actual work. So much so, that *ad hoc* strategies 'made-up along the way' quickly become irresistible; there is rarely a point in paying the large up-front costs of evaluating and introducing a complex system while a project is small and uses only a fraction of the capabilities. In comparison, it is much more efficient to use most capabilities of a system that has no up-front costs (such as naming a few folders and keeping a few tasks on a list). The *adhocery* of the latter does not matter if the project remains small or both grow together; transitioning later to an appropriately capable management system is usually not prohibitive for one such project. Attempting to avoid that cost is tricky, as there is no guarantee that the choice of a given large complex management system will pay off: it may turn out to lack *one* critical feature that forces complete re-organization or limits further growth. Given how unpredictable such critical features are in real life, most small projects do best to pick any system available at (next to) no startup cost – lest the project itself will never get started. Exceptions to these rules of thumb are projects aiming to set up structures that are intended to become long-term backwards compatible and/or projects that need to coordinate across many potentially independent subprojects, and thus require replicated and dependable structures (such as Evolvix).

Thus, a firm requirement for developing this POST system has been to welcome new users by keeping initial startup simple, albeit without killing projects indirectly later by no longer being able to support their growth.

2. Overview

POST aims to *standardize* aspects of organizing that do not really change between projects, and 'pick one side of the road' by choosing standardized names for frequently recurring needs to store certain types of information. It then offers projects to *customize* any given POST Home Folder for its specific purpose by allowing certain types of folders to be present or not. Using POST is no guarantee for the success of a project like driving on the right side of the road will not guarantee that a journey serves its purpose. However, those who give the POST system a try may find some of its catchy Brief and Explicit Names to be memorable, and the organizational support offered to be surprisingly useful. Progress in projects becomes much simpler if there is no need to arrive at customized decisions about which side of the road to pick on each turn of each project's journey. This more general perspective may help to set the frame for the more specific introduction of POST that follows next.

The Project Organization Stabilizing Tool (POST) system has been designed for any

- **Project** or complex set of tasks that depend on information processing in need of good
- **Organization** for improving work efficiency and reducing inessential complexity by
- **Stabilizing** flows of information using well-defined containers, building an automatable
- **Tool** that is easily customized and can help to separate the wheat from the chaff in messy project data while simultaneously encouraging creativity, rapid innovation, testing, and the slow emergence of reliable standards with long-term stability.

First we will introduce the POST system with a brief overview aimed at a more general audience; then we will highlight the original use-case it was developed for in computational biology, before we will switch gears and summarize important features of its design. We will also address the question of how to define progress towards long-term backwards compatibility, including an assessment of its current status, and areas of further development, all of which will be important as the POST system evolves towards increasing stability. An overview of the next sections follows:

This overview and the last two highly condensed pages provide perspectives on the POST system, that is defined by its BEST Names that specify a nucleus for developing POST standard semantics.

3. Introduction

This part is aimed at a more general audience of potential POST users.

3.1. Motivating problem: the need to organize

Ever wondered how to stay on top of a dynamic project that requires collecting and processing a lot of information? How to track the many improvements, small and big, that make a polished final version? Where to keep to-do lists, lessons learned, current problems? How can the final product be separated from all the information needed to produce it without making further improvements cumbersome? These challenges compound as projects become more complex, more contributors join, and the quest for quality intensifies in order to develop an outstanding product meant to last for a long time. The reason is:

Every contributor needs the right information at hand to make a difference and be efficient.

It is natural for many people to sort information into hierarchical *ad hoc* file trees and other organizational structures created on the fly. However, such improvisation can make it very difficult to work together efficiently, especially if organizational problems are solved in very different ways by the people involved. For example, different contributors might regularly prepare files with tasks to be delegated to other contributors, but everybody stores them in a folder with a different name, such as 'Delegated', 'Tasks', 'Todo', 'Work', 'Actionable', and so on. As naming preferences multiply, collaborators will spend an increasing amount of time communicating where files are (or risk that tasks remain undone and work is being misplaced). If this confusion is to be avoided a common project management strategy is required, but there is rarely enough time to develop it when the project "just needs to get done". To adopt an existing strategy may trigger a possibly prohibitively complex evaluation process to ensure that the system of choice meets all needs without imposing undue costs from inessential complexity.
Our efforts to support programming language developers and users have revealed the need for a simple-yet-scalable organizational scheme that makes it quick to start small projects in a way that does not prohibit them to gradually grow until they become complex long-term projects. Ideally, this scheme also quantifies the stability of different parts of large, complex projects and provides stable points of reference that can be used to measure progress towards aims of long-term stability (and backwards compatibility) without stifling the innovative processes that put a project on the map in the first place (the *"StabilizingZone"* introduced below aims to accomplish this).

3.2. Solution: organize project content using BEST Names

We developed the POST system described below to organize the development of the Evolvix programming language and its many nested smaller projects in a stable network of well-defined containers. We aimed to avoid recurrent, costly episodes of complete re-organization (such as the unavoidable one that triggered our work on POST). Due to the broad scope of Evolvix, we designed POST to offer a flexible infrastructure for supporting the development of very diverse project types. At its core is a set of carefully chosen standard "*StabilityCodes",* or *"POSTcodes"*, aimed at separating highly refined information that moves very slowly from more transitory to more stable states as it is reviewed with increasing rigor. These StabilityCodes form a sequence that follows the alphabet and defines "DoubleCaps" as keywords, starting with MM, NN, OO, … and ending with … RR, SS, TT:

MM	MockupModel	MockupModel_UsedFor_RapidPrototyping_InformalLearning__ __ExpertimentsToBeThrownAway_StabilizingDesignNotCode
NN	NewNonfunctional	NewNonfunctional_UsedFor_NotYetFunctioning_DeepFoundations__ __ForLargerStableDesigns_ThatDoNotYetWorkForUsers
OO	OperatesOften	OperatesOften_UsedFor_Systems_PartiallyWorkingForEndUsers__ __while_StillMissing_ImportantFeatures_ToBeImplemented
PP	PreProbing	PreProbing_UsedFor_Preparing_PeerReviewAndPublicProbing__ __by_PolishingExistingFeatures_UntilSubmissionFor_Questioning
QQ	QualityQuest	QualityQuest_UsedFor_Questioning_AxiomsDataScienceAccuracy__ __RigorClarityUsability_InMany_ExpertBeginnerReviewRounds
RR	ReviewedRelease	ReviewedRelease_UsedFor_NewReleasesRecommended_by__ __QualityQuestEditors_after_AnsweringAllReviewerQuestions
SS	StableSource	StableSource_UsedFor_StunningSoftware_RunningInProduction__ __with_LongTermSuccess_and_VeryRareRevisionRequests
TT	TrustedTested	TrustedTested_UsedFor_Marking_VeryLongTermStableDesigns_in__ __WellUnderstoodDomains_AllowingBackwardsCompatibleGrowth

Table P1: All *StabilityCodes* (MM…TT) defining the *StablizingZone* of POST.

Other *POSTcodes* denote other aspects of project management that are independent from these *StabilityCodes* codes, but also reflect in some ways on the stability of the information they annotate (such as *AnyArrival* for arbitrary project relevant material that has just arrived, or *VersionVariant* for historic releases of variants). Drawing on the alphabet's familiar progression, the *POSTcodes* and *StabilityCodes* roughly organize the initial project phases with earlier letters and the increasingly mature phases with later letters.

These POSTcodes annotate containers, such as folders, with a short description of their intended purposes, and they exist as the first three standardized synonymous names defined by the BEST Names concept:

- **Brief** Names: here mostly double capital letters for speed, brevity, and memorability;
- **Explicit** Names: here mostly memorable keywords reflecting the capital letters; and
- **Summarizing** Names: cheat-sheet-like summary sentences that act like mini-manuals.
- **Technical** Names will be defined later to serve as core for corresponding StableNames.

Our use of BEST Names aims to enhance the intuitiveness and memorability of *POSTcodes*; indeed, without BEST Names it is unlikely that POST would have been developed. Many of its use-cases require very frequent use of POST codes at very short notice, that neither leave much room for typing nor much time for remembering cryptic names, as seen in the example given next.

3.3. Example: Using a POST Home Folder to write a paper together

It is simple to start using POST. It can be as easy as moving old variants of processed files to a *HistoryHeap* (*HH*) folder for decluttering a project folder to increase efficiency (and avoid the much larger cost of determining whether historic variants of a file will ever be needed again for any purpose; storage is usually cheap). Additional folders follow as needed. To select them, the Summarizing Names from the Brief Dictionary of POST in Figure P3 can be extremely helpful. Here is a short list:

AA	AnyArrival	AnyArrival_UsedFor_Inbox_QuickDropping_Items_to_ProcessLater
AAA	AnyAimsAdmin	AnyAimsAdmin_UsedFor_Managing_StrategyAims_ProjectPlans__ __Workflows_RoleDefs_Tracking_TasksDeadlinesFunds__ __RecruitingTraining_Delegation_CallBacks_Prioritizing_KeyGoals
BB	BackBurner	BackBurner_UsedFor_Storing_SeasOfGoodIdeas_to_RevisitLater
AHA	AnyHelpArea	AnyHelpArea_UsedFor_OnlineHelp_UserManuals_InfoMessages__ __TeachingOutreachMaterial_Translatable_by_CountryCode
CC	CollectedContent	CollectedContent_UsedFor_Collecting_External_TopicIdeas_Data__ __Methods_Evidence_Refs_Reviews_Clarity_CodeExplanations
FF	FeedbackFlow	FeedbackFlow_UsedFor_Logging_AnyInputCommentCritiqueBug__ __Ideas_from_ReviewerExpertUserFriendFoeFans_Worldwide
GG	GrandGallery	GrandGallery_UsedFor_Presenting_BestExhibits_LatestResults__ __FinalKeyDeliverables_without_NeedForLongTermStability
HH	HistoryHeap	HistoryHeap_UsedFor_OldVersionBackup_ProbablyDeletableFiles__ __HoardedDataWithLimitedOrder_KeptFor_PotentialUsefulness
JJ	JammedJob	JammedJob_UsedFor_Documenting_BugReports_DecisionNeeds__ __and_ProgressOn_SolvableProblems _that_NeedSolutionsSoon
LL	LabLog	LabLog_UsedFor_Logging_AnyLabLabor_Dated_YYYY_MM_DD__ __ForHistoricSequenceOf_AllOutcomesOf_AllTests_IncludingFails
UU	UploadUsed	UploadUsed_UsedFor_Archiving_ExternalCommunicationLog_of__ __FullUploadsOf_InputTo_or_OutputFrom_GG_etc

Table P2: Brief, Explicit, and Summarizing Names of basic *POSTCodes* used for choosing in Fig P1.

For example, a research paper in the folder of *MyResearch* might select from this list the folders shown in Figure P1 below. Usually *MyResearch* starts with a very small folder, only storing any arriving ideas and information in an unstructured form in the *AnyArrival* (*AA*) drop box. As the collection of content continues, arriving files are sorted into *CollectedContent*. Planning prioritizes aims in *AnyAimsAdmin* (*AAA*) structures. The final latest and greatest version of the product produced will grow and reside in the *GrandGallery* (*GG),* and a *LabLog (LL)* may help record a chronology of the work done in preparation for the big finale. To declutter, old variants or unused files are moved from *GG* to the *HistoryHeap* (*HH*) and good ideas for later from *AAA* to *BackBurner* (*BB*). Writing and working often result in *JammedJobs* (*JJ*) that need solutions, and increasing project complexity might lead to a reorganization of content that results in new *GG* folders for different workspaces. Shared manuscript variants are stored as *UploadsUsed* (*UU*), and incoming comments collected in *FeedbackFlow* (*FF*). See Figures P2 and P3 for more potential workflows.

The complete POST system includes many more features than are needed in most use cases and is being designed to provide developers with powerful abstractions for facilitating the automation of data flows. However, none of this needs to infringe on personal organizing preferences for users who do not need automation. POST's beauty is in its flexibility: it encourages using of *StabilityCodes* as needed and ignoring of all others – they will wait until content arrives that they can organize.

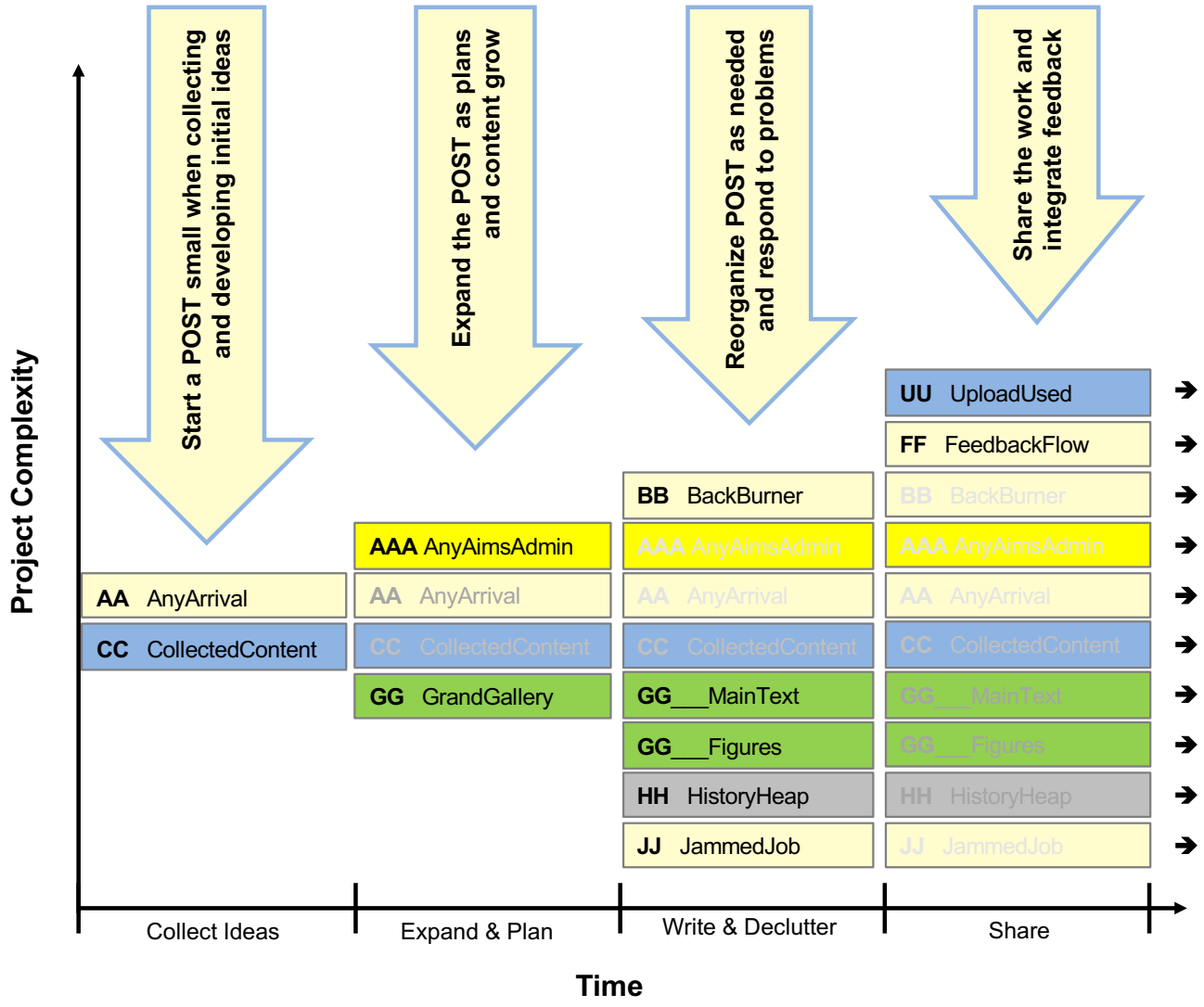

Figure P1: Using a POST Home Folder for organizing a mini-project such as writing a paper. Exemplary activities are shown in columns and the folders they generate in rows as they persist. The names presented here were found to be easy to understand with minimal explanations in our internal tests. More complete definitions using Brief Names (e.g. “AA”), Explicit Names (e.g. “AnyArrival”) and Summarizing Names (e.g. giving a short description of how to use “AA”) are given in Table P2 and the overviews of the POST system in Figures P2 and P3 below. Brief and Explicit Names are given here for clarity, but this would not usually be done in practice (Brief is enough). The POST system can tolerate a great number of variations when used manually, but many contributors usually benefit from stricter following the naming rules to avoid confusion, while POST automation requires following all formal rules. POST can help organize data in diverse storage media, including manual use on paper.

3.4. High-level structure of the POST system

Generality, flexibility and origin. The POST system is meant to be as general as reasonably possible so that anyone can use it for efficiently managing projects, small or large. This is the case even though (or rather because) it was primarily developed for supporting the implementation infrastructure of the general-purpose programming features of the Evolvix modeling language. This language aims to support a very broad set of use cases to enable efficient work in computational biology (hence the generality, which is needed to accommodate the broad diversity of biological use cases). The POST system facilitates standardizing and customizing development work across complex and independent code, text, and project management structures. Users can drop features they do not need initially, only to introduce them as necessary when a project keeps growing in complexity.

POST Home Folders (PHF), manual POSTboxes, and automated POSTnets.
The POST system is designed to encourage automated file management while also providing an effective structure for manual organization that makes it easy to set up and manually operate a single POSTbox that consists of a single POST Home Folder. Simultaneously it aims to support the growth of massively distributed collaborative projects that eventually combine many automatically managed PHFs into larger POSTnets (e.g. to meet citizen science project requirements described elsewhere [2]). Such networks facilitate globally distributed collaboration by automating well-defined information flows between individual PHFs, while clearly communicating the stability of the code involved (albeit without burdening users who do not need this for their POSTboxes).
We define

- a **POST Home Folder** (PHF) as the one top-level folder of a given POSTbox, containing its own independent active area, ZZ* folders and POST_Cabinet folder (which in case of automation must be stored under the name "_POST" or "a_POST", if leading underscores are forbidden; manual use is more flexible); all folders and data in a POSTbox is stored relative to the PHF, which serves as a local anchor in its broader context;
- a **POSTbox** as the implementation of a single PHF that allows for manual management – just like the old-fashioned mail service that requires hands-on file management and is simple to use; a POSTbox may have links to other POSTboxes (see ZZ, ZZA, ZZ* folders);
- a **POSTnet** as a combination of many PHFs that can easily become very intricate and difficult to keep in a consistent state manually; to maintain consistency, POSTnets are usually best managed automatically, especially when multiple PHFs are nested, interact in complicated ways, and/or contain data that is costly to re-synchronize after incomplete manual operations;
- the **POST system** (or simply 'POST') as the abstract type specification defined in this document or a future POST standard; it shows how to implement a POSTbox or POSTnet.

Nesting and automation. PHFs can be nested inside and linked to other PHFs, thus facilitating great flexibility as they can replicate and network at all levels of a project hierarchy. While such networks can be managed manually in principle, it is more likely that they require automated POSTnets to reduce operational errors.

Simple decisions before starting a POST Home Folder. A project using POST should consider:

- Will it be operated manually, like most project, and thus use the "*GrandGallery*" as place for developing and updating the latest content, adding other folders only as necessary?
  For a simple example, see the Figure P1 above.

- Will it aim for some form of stability or backwards-compatibility to help guard the investments of outside users who rely on the stability of the system developed by this POST? Then using the *StabilizingZone* is recommended to encourage stability, even if *TT* may not be the goal. Thus, new content is developed in *MM* or *NN*, and moved up the rungs of the *StabilizingZone* as it matures up to RR or SS, possibly with automated support.
- Will the PHF be automated eventually or manually managed throughout its life-time? Automated PHFs follow stricter rules that are easier to enforce from the start, even if managed manually. Costly re-organizations may be avoided by following the rules right away.

Rigidity of structure. If automation is not needed, then users of a PHF may interpret POST related structures more flexibly. Humans are much better at navigating ambiguity than computers, albeit consistent use has substantial advantages for humans as well, especially when many contributors are involved. For example, it is easy to append tags that serve as *WorkspaceIDs* to further break down navigation complexity, even if not yet supported by automation (e.g. see "*GG*-Figures" in Figure P1); it may not matter to humans, whether peer reviews are stored as *UploadUsed* (by reviewers) or next to potentially a lot of files inside of *FeedbackFlow* from the web, but automated PHFs need a standard answer allowing them to expect defined content in a fixed place and thereby reducing inessential code complexity. Most areas where the POST system requires additional work are such decisions of locating within the folders already defined in Figures P2 and P3. Such choices do not alter the core meaning of POST *StabilityCodes* and hence rarely matter in a manual POSTbox.

3.5. Overviews of POST system definitions using BEST Names

Nucleus for a POST system standard
While Figure P1 above provides an incomplete introductory overview of POST parts frequently used in simple projects, the Figures at the very end of this text provide complete overview perspectives of the core of all top level *POSTcodes,* including the *StabilityCodes* that facilitate the project organizing and stabilizing aspects of POST:

- Figure P2: Information flow overview illustrating some inner connections between *StabilityCodes* and which folder is located where (if a POST is implemented as folders in a file system, which is not the only possible option).
- Figure P3: Brief Dictionary of all core *POSTcodes* giving interchangeable BEST Names

The core *POSTcodes* presented in these figures can be complemented by more specific *POSTcodes* that belong to reserved and controlled vocabulary lists of keywords and that streamline the naming of more specific areas, such as organizing observed data, estimates of parameters and simulation results. Discipline, research-area or industry specific keywords lists are conceivable, whose applicability to a given POST Home Folder is defined by the type assigned to the PHF. Overall, the POST system design aims to minimize the number of these keywords; if possible to maintain, POST development will aim to make these sets of words non-redundant and free of “near-synonyms” to ensure overall simplicity of use by maximizing the re-use of core names.

The core names defined in these lists also form the core material for generating Technical Names that become part of the StableName and StableMeaning as defined by the BEST Names concept to ensure unambiguity in the presence of potentially many translations and (perfect) synonyms.

4. Computational biology use-cases

Next we will present details of the the use-cases that motivated the development of POST.

4.1. Why *TrustedTested* (*TT*) Stability could be pivotal for personalized medicine and evolutionary systems biology.

Not all software systems let alone non-IT systems require *TrustedTested* or *TT* stability. However, the equivalent of *TT* stability is critical for systems that depend on the results of very large numbers of contributors who need to build upon each other's work over very long periods of time to achieve overall success. This requires using a common language to improve their efficiency of communication; if the number of required contributors is so large that newcomers need to be recruited, then chances of success will increase dramatically if learning this shared language is easy. Modeling biological systems has shown that computational techniques can in principle be applied to vastly more biological systems than currently modeled. While this discrepancy may have many reasons, on important reason is certainly the lack of a *TT* stable modeling language that efficiently helps biologists to formally describe their mechanistic understanding of molecular and other systems they investigate (including experimental results) to allow for corresponding computational systems biology analyses. The emergence of some standardization (e.g. see SBML.org [3]) is encouraging, but there is a long way to go until the computational tools for biology can integrate current biological understanding well enough to

- enable **personalized medicine** analyses that efficiently build on all known evidence (and not only the datasets that happen to be available),[4] or
- enable **evolutionary systems biology** analyses predicting phenotypes and fitness from genotypes and environments, in order to simulate how populations are likely to evolve,[5]

or enable similarly bold and challenging visions in predictive cancer research, conservation biology, or development of policies to slow antibiotics resistance evolution in bacteria. All these areas will eventually need to combine biological observations, models and results from many decades in the past and in the future. Only a reliable computational integration can enable future biologists to efficiently build on the foundation of past results without losing them to semantic rot or requiring the steep cost of continually re-implementing past modeling code. Experiences with the first simulation project that used Evolvix [6] highlighted the potential for automating such an integration.

To efficiently enable personalized medicine or evolutionary systems biology it is essential to find a way to keep active the models that are currently being buried in immutable online publications. Activation could be greatly simplified by implementing models in code that is easy to read and has the *TTv1* stability allowing future biology students to easily extend old model code using new *TTv1* downloads.

This vision drives Evolvix, and inspired the quest to define a StabilizingZone leading to TrustedTested and the POST system. Evolvix adopts POST to increase code stability. There is no reason why other projects cannot use the StabilizingZone and POST to improve their stability as well.

4.2. Area-specific controlled vocabulary lists in POST

To avoid a confusing chaos of folder names, POST defines some keywords for the purpose of streamlining the locations that programs would have to know in order to find corresponding files. For example, in POST, simulation results should always be in a folder that has "**Sim**" in the path, even if many other aspects remain flexible, while "**Obs**" always denotes some observation of a system that is not a simulation. The following list of Brief and Explicit Names is given to reduce confusion. In addition, we offer a short explanation that will be used to create a Summarizing name at a later point.

This list is by no means complete. As long realized in biomedical research, there are numerous benefits to using a system that can combine terms and fragments to derive new composed terms denoting new meaning (often based on Latin or Greek roots [7]). POST will not need all conceivable terms (which gives away the advantage of simplicity); indeed, a case can be made for introducing discipline/application/industry specific vocabularies, which should be defined in combination with the specification of a controlled type for the PHF. Ultimately any such efforts require the construction of an ontology [8, 9] or type system to keep POST well defined.

4.2.1. Top Level Key words in Brief Dictionary

Other Keywords with reserved top folders and independent subfolder namespaces:

- **Def** Definition,
- **Est** Estimate,
- **Obs** Observation,
- **Sim** Simulation,
- **Res** Result,
- **Lab** LabAutomation,
- **Log** LogAutomation,
- **WebS** WebSubscribed,
- **WebP** WebPublished,
- **MO** Mode_of_Computing,
- **FSM** FiniteStateMachine,
- **Ma** Machine,
- **Me** Memory,
- **Sp** Space,
- **Fu** Function,
- **Te** Template,
- **Ro** Role,
- **QAS** QualityAssessmentSummary (Human),
- **Pbc** ProblemTreeCompilation (Automated),
- **Data** Data (Unspecified),
- **Type** Type (Unspecified),
- **Code** Code (Unspecified),
- **Idea** Idea (Unspecified),
- **Opi** Opinion

4.2.2. Extensions of Top Level Key words in Brief Dictionary

Combine the given 'NameStarter' from above with any of its reserved follow-on 'KeyNameFragment' below (or use subfolders; defining the full intended meaning in Def/POST):

- **Preprocessed (Obs-)**:
  - **Adj** Adjusted,
  - **Chk** CheckedIntegrity,
  - **Maj** Majority,
  - **Mis** Mistake,
  - **Mod** Modified,
  - **Odd** Outlier,
  - **Raw** AsReceived,
  - **Sca** Scaled,
  - **Tra** Transformed,
  - **Try** Risk;

- **Estimation (Est-)**:
  - **Gue** Guessed,
  - **Pre** Predicted (=Gue*Lkh),
  - **Evi** Evidence,
  - **Upd** Updated,
  - **Lkh** Likelihood,
  - **Xpl** Explained,
  - **Fct** Forecast,
  - **Dst** Distance,
  - **Add** Addition
  - **Mul** Multiplication
  - **Dif** Difference,
  - **Div** Division
  - **Difl** Differential,
  - **Cal** CalculatedAnalytic,
  - **Sim** SimulatedComputationally,
  - **EIndx** EstimatedResultsIndex;

- **SimulatedModelResult (Sim-):**
  - **MFr** ModelFrame,
  - **MSt** ModelStructure,
  - **MVa** ModelVariant,
  - **MRe** ModelRepeat,
  - **MIndx** ModelResultsIndex;

- **ProblemTreeCompilation (PBC-)**:
  - **Err** ErrorTree,
  - **Wrn** WarningTree,
  - **Rpt** ReportTree,
  - **Ifo** InfoBlockTree,
  - **Why** WhyCausalTree,
  - **Slo** SlowProgressTree
  - **Dbg** DebugTree,
  - **Ast** AbstractSyntaxTree;

  PBC is to be expanded substantially. All these entries are compiled automatically (in contrast to Pbh entries below, which are entirely based on human annotation).

- **QualityAssessmentSummary (QAS-)**:
  - **Pbh** ProblemTree (Human annotations),
  - **Lim** LimitationsTree,
  - **Cal** CalendarTree,
  - **Mat** MaturityTree,
  - **Exi** ExistenceTree,
  - **QAT** QualityAssessmentTransformation;

  QAS is to be expanded substantially to facilitate a wide range of human annotations using controlled vocabulary for quickly annotating repeating quality assessment scenarios, including diverse fuzzy situations, where it is difficult to assess quality because of many unknowns.

4.2.3. Flexible Modifiers of other keywords in the Brief Dictionary

These modifiers cannot stand alone, but otherwise work very well in combination with some of the keyword fragments defined above:

- **Compared**:
  - **U** Usual,
  - **Q** Other
  - -
  - **Wt** Wildtype,
  - **Mt** Mutant,
  - **Wtl** Wildtypelike,
  - **Mtl** Mutantlike,
  - -
  - **Ni** Noise,
  - **Si** Signal
  - -
  - **Tst** TestExperiment,
  - **Ctr** ControlExperiment;

- **Seq**uence**Chain**:
  - **Frst** First,
  - **Prev** Previous,
  - **Curr** Current,
  - **Next** Next,
  - **Last** Last,
  - **CIndx** ChainElementIndex;

- **Relevance**:
  - **Core** KeyResults,
  - **Keep** InArchives,
  - **Late** LatestFewResults,
  - **Logg** LoggingAllResults,
  - **Fail** FailedTriedRisk,
  - **Bugs** BugsAnyReport
  - **Prob** ProblemsInModel,
  - **Reso** ResourceWarning,
  - **Temp** TemporaryIntermediateResults;

4.3. Defining *TrustedTested* in POST

The unusual nature and long-term importance of *TrustedTested* as a *StabilityLevel* merit a separate list of requirements and comments to explain how *TT* works.

Projects aiming to reach *TT* are encouraged to ask questions about how to enable long-term stability as early as possible and are cautioned against elevating code too quickly to *SS* without evidence that its design has what it takes to go all the way to *TT*; thus *SS* can be used as a testing ground that buffers *TT* from integrating solutions that have not received sufficient review. There is no need to risk elevating new features too quickly to *TT*. Many successful software products have demonstrated that remaining at the equivalent of *RR* does not hamper success, and many high-quality IT standards that provide stability at *SS* do so without the need for *TT* promises. *TT* stability is very difficult but not impossible to achieve, as IT systems change continually and *TT* systems need to anticipate and abstract these changes well enough to isolate them from affecting any code written for such *TT* systems. Thus, in the absence of evidence for extraordinary stability or if in any doubt, software projects providing a single high-quality implementation of their design are recommended to stay at *RR*, and official standards without guarantees for long-term backwards compatibility at *SS*. However, a precise definition of the POST requirements for transitioning from *QQ* to *RR* to *SS* and to *TT* is beyond the scope of this study and remains to be reported elsewhere.

a. **Simplify use for outsiders.** Any downloadable system developed using POST and marked "*TTv1*" indicates that it belongs to the version variant family
   "*TrustedTested version 1*" = "*TTv1*" expect long-term stability
   which follows a well-defined set of requirements providing the following capabilities (where some details may need to be defined by the corresponding POST project):

b. **Work with all older code written for stability.** The latest downloadable release or patch of a *TT* version variant family can correctly interpret any code produced for any *previous* release or patch of this *TT* version variant family (e.g. *TT* version 1 can interpret any code for any release or patch back to v1r0p0, the original *version* 1 *release* 0 *patch* 0)

c. **Use Stabilizing Versioning.** It is beyond the scope of this study to describe the stabilizing version variant naming system implied here; once fully defined, it is to become part of POST.

d. **Avoid breaking backwards compatibility.** In the POST system standard, breaking the compatibility of a new *TTv1* release with any previous variant of *TTv1* is not allowed and requires the definition and implementation of *TT* version 2 with an automated translator that correctly produces *TTv2* code from any previous *TTv1* code that is no longer compatible. Thus the release of any new *TT* version essentially terminates the series of releases for the previous *TTv* and triggers the need to automate the migration of a potentially very large code-base (making this a very costly operation not to be undergone lightly if unavoidable at all).

e. **Clearly mark code that is not (yet) stable** either as *TTv0* or by omitting *TT* or by adding other *StabilityLevels*:

"*TTv0*"	do not yet expect **long**-term stability
"*SSv1*"	do not expect **long**-term stability
"*RRv1*"	do not expect **medium**-term stability
"*QQv1*"	do not expect **short**-term stability
…	expect less and lesser **short**-term stability
"*MMv1*"	expect least **short**-term stability

Given the many strict requirements and prolonged review processes required for developing code at the *TT StabilityLevel*, projects aiming for *TT* first need to denote many version variants that do not yet meet *TT* criteria. Code that does not contain any *TT* level code is to be annotated at the appropriate level (*MM*…*SS*). Code modifying any pure *TTv1* variant with less reliable changes is to be denoted by adding the *Brief Names* of its lower *StabilityLevels* to the version variant label, thereby indicating the loss of *TT* stability. For example, in

"*TTv1r2* "	has only features of *TT* version 1 release 2
"*TTv1r2_OOv3r4*"	*TT* core (v1r2) with *OO* extensions v3r4
"*TTv1r2_OOv3r4_MMv0r0p3*"	further extended by a hack at *MM* level,

the overall stability is specified by the lowest *StabilityLevel* present, even if parts of the system are more stable since they were not modified. However, the presence of any modifications makes it difficult to exclude interactions that destabilize the whole system. Hence all modifications of any code above *TTv1r0p0* require full review in the context of how they affect the whole system they have been added to.

f. **Minimize contradictions**. All output from *TT* systems is required to be accurate to the best of current knowledge, which may advance as research progresses. This implies the emergence of new algorithms and/or fixes for bugs in known algorithms. Bug-fixing *TT* patches can in principle change output if the older system produced output that is demonstrably wrong. However, appropriate review at other stability levels is expected to catch such bugs before an algorithm is advanced to *TT*. In areas that are known to be difficult to standardize, such as numbers and arithmetic systems, alternative systems can be distinguished by corresponding differences in names, thus enabling the addition of new interpretation systems without breaking the compatibility of code that rely on previously implemented systems. Similarly, the discovery of improved algorithms does not require abandoning the possibility of using previously used algorithms. However, new algorithms shall eventually result in *TT* releases that automate running new algorithms in addition to old ones, facilitating comparisons of results. Design and algorithm choices that are *TrustedTested* are expected to have survived multiple rounds of rigorous conceptual and usability review, repeated design simplifications, prolonged use in professional production environments, and many automated test cases. A full list of criteria for justifying the progression of code to *TT* is beyond this study, except to say that it should not happen too fast to minimize contradictions and to simplify design (see next).

g. **Use rigorous review for reducing clutter in namespaces**. Careless design decisions can quickly and irrevocably clutter the *TTv1* namespace of a given project and thereby increase its inessential complexity[10]. This can quickly degrade the prospects of long-term survival for the project. The special nature of *TTv1* features causes this, features that signal to users and

programmers that they will all remain available on the long term. Thus the leadership of POST based projects is recommended to be slow and careful when allowing names to enter into the *TTv1* namespace. The POST *StabilizingZone* does not require such rigor for *TTv0* or any full versions of other stability levels (*MM* … *SS*), providing many opportunities for experimenting with mutually incompatible competing implementations of a new feature before selecting one of them for *TTv1*. While POST generally expects an increase of stability from *MM* to *SS* and with increased version numbers within a *StabilityLevel*, it expects even more that systems become public at *RR* and noticeably reduce the changes that remain possible as stability moves through the levels *RR* -> *SS* -> *TT*, while incorporating worthwhile improvements suggested in a *FeedbackFlow* from public users.

The path to TrustedTested: an overview of the *StabilizingZone*.

Here *GG* serves as the "Null-Element", that moves equivalent content outside of the *StabilizingZone*, if there is no aim to ever attain long-term backwards compatibility. There is no difference in the assurance of stability between GG and MM, only a difference in ultimate intention.

VV serves as the installation location and ultimate archive for any products produced by the *StabilizingZone*.

XX provides temporary build space needed for generating the final installation-ready files (its stability will be copied from independent upstream code that produces it, hence XenoXero).

GG	GrandGallery	GrandGallery_UsedFor_Presenting_BestExhibits_LatestResults__ __FinalKeyDeliverables_without_NeedForLongTermStability
MM	MockupModel	MockupModel_UsedFor_RapidPrototyping_InformalLearning__ __ExpertimentsToBeThrownAway_StabilizingDesignNotCode
NN	NewNonfunctional	NewNonfunctional_UsedFor_NotYetFunctioning_DeepFoundations__ __ForLargerStableDesigns_ThatDoNotYetWorkForUsers
OO	OperatesOften	OperatesOften_UsedFor_Systems_PartiallyWorkingForEndUsers__ __while_StillMissing_ImportantFeatures_ToBeImplemented
PP	PreProbing	PreProbing_UsedFor_Preparing_PeerReviewAndPublicProbing__ __by_PolishingExistingFeatures_UntilSubmissionFor_Questioning
QQ	QualityQuest	QualityQuest_UsedFor_Questioning_AxiomsDataScienceAccuracy__ __RigorClarityUsability_InMany_ExpertBeginnerReviewRounds
RR	ReviewedRelease	ReviewedRelease_UsedFor_NewReleasesRecommended_by__ __QualityQuestEditors_after_AnsweringAllReviewerQuestions
SS	StableSource	StableSource_UsedFor_StunningSoftware_RunningInProduction__ __with_LongTermSuccess_and_VeryRareRevisionRequests
TT	TrustedTested	TrustedTested_UsedFor_Marking_VeryLongTermStableDesigns_in__ __WellUnderstoodDomains_AllowingBackwardsCompatibleGrowth
VV	VersionVariant	VersionVariant_UsedFor_InstallingAllLocalProducts_FullyArchived__ __Checksummed_ReproduciblyWorking_AllFiles_Ready2Publish

5. Design

The following information is advanced and meant for software architects, designers, and developers; it is not intended for a general audience.

5.1. Functional requirements and features

The POST system was optimized in a trade-off between the following requirements:

a. **Easy entry point** for newcomers with as few required structures as possible (none).

b. **Simple growth** by adding only structures that are needed locally (one by one).

c. **Conceptual clarity**, providing roles for each structure that are well defined and of the highest importance for corresponding large projects.

d. **Brief memorable keywords**, including the use of all double capital letters of the English alphabet as *Brief Names* for *StabilityCodes* with matching *Explicit Names* carefully chosen as memorable reminders of their meaning (example: *HH* for "*HistoryHeap*").

e. **Memorable organization** of *StabilityCode* names along the alphabet roughly tracking stages of project progression (start with early letters for early stages; minimize exceptions).

f. **Use of the alphabet** to group *StabilityCodes* by types for well-defined use cases (choosing the clearest *Explicit Names* with corresponding initials to sort accordingly with the alphabet).

g. **Flexible** to allow extremely complex or simple project structures to exist concurrently.

h. **Clear transitions and support for both manual and automated operation** of POST in many diverse storage structures (folders in file systems, Git repositories, sets with nested subsets and elements, arbitrary data structures, diverse off-line paper-based systems, etc.)

i. **No upper limits for nesting** full-scale POST systems within other POST systems unless explicitly named (such as absolute file path length limits; storage size limits, etc.).

j. **Clearly defined rules for making it easy to create tools for automating** tedious tasks (such as activating or versioning the content of folders with various levels of stability).

k. **Support for all key aspects of system development** from first idea over the various stages of the software development life cycle to archiving the last results.

l. **Defined information flows** between various containers with different *StabilityCodes* (specifying which flows are allowed and which are not).

m. **Utmost reduction of inessential complexity to maximize usability**, but without sacrificing scalability or other functions that are essential for helping to organize and stabilize projects

(together with alphabetical and mnemonic requirements this resulted in considerable naming challenges, especially for identifying memorable *Explicit Names*; we cannot see how to meet these challenges without at least separating *Brief* and *Explicit Names*; much of the value of POST is defined by the quality of this integration task).

n. ***Summarizing Names*** for *StabilityCodes* that serve as "mini documentation", reduce the need to consult a manual, and provide a nucleus for standardizing POST system semantics.

o. **Define a *StabilizingZone* with defined *StabilityLevels*** to serve as anchor for a stabilizing version variant numbering system that can help software projects to steer code towards long-term backwards compatibility by explicitly disentangling the speed of fast 'agile' prototype development (such as *MockupModel, MM*), from slow highly organized 'waterfall' project development (typically published as *ReviewedRelease, RR*), and in turn from the glacial pace of development of most international standards (usually *StableSource, SS*, albeit without excluding future changes or promising migration capabilities); keep these use cases separate from the endpoint of the *StabilizingZone* (*TrustedTested*, *TT*), which is required to remain 'stable forever' to the best of current knowledge or provide a migration path (see below).

p. **Encourage long-term backwards compatibility** of systems developed with POST by helping end users to quickly recognize version variants offering such maximal stability by providing a clear and memorable label (*TrustedTested*, *TT*) linked to a rigorous review process.

These requirements must be included in future extensions of the POST system that ideally extend and refine this list in order to move POST closer to the stability denoted as *TrustedTested*.
The POST definitions given here are not yet at *StabilityLevel TT*, but were designed with the aim to encourage and measure progress towards developing *TT* stable code in and for Evolvix (in order to reduce the cost of irreproducible research). This implies that the POST system defined here (and in particular its *StabilizingZone*) is closer to *TT* than any other code developed in or for Evolvix (see statement on status in section below). Thus, good reasons need to be provided for either

- **removing** any requirement specified above to justify why the corresponding POST use cases are best expressed in other ways, or
- **adding** requirements to justify why the corresponding use cases are important enough to merit inclusion in POST more than they merit removal to serve the aim of reducing the cognitive complexity for all users of POST (since beginners and experts pay for clutter caused by inessential complexity in POST, albeit in different ways).

5.2. Rules for automating and extending POST Home Folders

To be automatable, a PHF such as '*MyProjectFolder'* must adopt these structures and rules:

File system path to folder	***!l Comment explaining the folder on this line***
MyProjectFolder	*!l The POST Home Folder is the anchor link base for automation.*
…/_POST	*!l POST_Cabinet holds folders for all StabilityCodes as needed.*
…/_POST/AHA	*!l All folder names beginning/ending with Brief Name*
	!l StabilityCodes are reserved in here*, and populated as needed.*
*…/_POST/ZZ**	*!l Reserve names starting with "ZZ*" in all POST_Cabinets; ZZ is*
	!l always outside; other ZZ folders may be inside or outside.*
*…/_POST/**	*!l To enable the development of a controlled POST vocabulary,*
	!l reserve all remaining names in all POST_Cabinets for future
	!l use (see Section 4.2 for examples); the names can be
	!l "borrowed", as long as it is understood, that POST may claim
	!l any name inside _POST at any time. When that happens, the
	!l PHF will be equipped with a well-defined type that specifies all
	!l names that have been reserved (and the remaining liberties).
…/…	*!l Active area of the PHF, that can be replaced by content of*
	!l another StabilityLevel or VersionVariant.
*…/*ZZ	*!l Content stored here is 'inside', but not managed by this PHF;*
	!l ZZ can contain anything, including other PHFs; ZZ is never
	!l replaced with content of other StabilityLevels or VVs.
*…/*ZZ*	*!l Reserve names starting with "ZZ*" (+lower case) in all PHFs.*
*…/*ZZA	*!l Anchors: declared links to folders or files outside of PHF;*
	!l any file/folder/web/etc link referenced in this PHF that is not
	!l inside and controlled by this PHF must go through ZZA.

Naming rules for files/ folders or elements/sets aiming to avoid name clashes with PHF POST infrastructure shall exclude these reserved POST names (in *all* upper and lower case variants):

a. Core PHF names: '_POST', 'ZZ', and each name starting with 'ZZ' (without separators)

b. Core names inside of _POST: each name that is, starts or ends with any Brief or Explicit Name of any *StabilityCode* in Figures P2 and P3; and 'WEB', 'POST', 'BOX', 'NET', or any combination of the last three; where '_', or '.', or '-' can lead, or trail, or separate these labels from the remainder of the name (or be the name) in any combination; also, each ZZ* name.

c. When developing scripts for a PHF, it is currently recommended to also avoid in the PHF namespace all core names explicitly defined for the inside of '_POST' (i.e. the POST_Cabinet), or to use them there as defined by POST.

d. Rules for names inside of automated top-level POST_Cabinet folders (e.g. 'AAA','HH', etc) have not yet been specified and will restrict some of their namespace while automation of the corresponding functionality is switched on (but not during manual operation).

Restrictions are to be reviewed to reduce them to the minimum necessary to support all key functions, which shall include internationalization and adaptations to allow in principle simultaneous work with diverse compilers, clouds and operating systems (as far as technically possible). Developers of scripts for PHFs are recommended to be cautious when making assumptions about POST names inside of top-level folders (and consider contacting the architect of POST). This caution can be ignored for the folders specified below and for purely manual POSTboxes.

e. All content of the following top-level POST_Cabinet folders shall be guaranteed to remain completely free from any restrictions or required additions. When future POST standards facilitate their automated management, corresponding metadata on content of these free-form folders must be stored elsewhere in their local PHF to maximize free content choice by users as much as relevant clouds and operating systems allow. The FreeFormFolders are:

   II, *GG*, *MM*, *NN*, *OO*, *PP*, *QQ*, *RR*, *SS*, *TT*, and *ZZ*.

   Note that users can turn any of these folders into PHFs or add PHFs of some type; this may make it necessary to distinguish the top PHF's '_POST' and 'ZZ*' folders from those of lower-level folders; a corresponding re-naming-when-active shall be developed as needed.

5.3. Translation in POST folders

Texts that need any sort of translation are generally stored as programmatically accessible stable links in the *AHA* folder which immediately splits into relevant languages (using ISO language codes). While this facilitates the translation of all user documentation and potentially important texts within applications, it does little for translating the POST folder itself, which remains overwhelmingly "English" or rather "abbreviated English". To customize its appearance for English and/or international users, we recommend to create links, similar to the approach of the link-farm manager "Stow" [11].

6. POST specification: current status, stability and future work

The POST specification details in this document define RRv1r0p0, or, omitting zeros, RRv1, both of which are Brief Names for this Explicit Name: *ReviewedRelease* version 1 release 0 patch 0.

Influences that shaped RRv1 include

(i) analyzing key requirements for the efficient management and development of complex software projects as observed by the main architect of POST over 15+ years in professional computational biology research environments;

(ii) surviving intense questioning in multiple rounds of review for more than a year, focusing on the aim to minimize inessential complexity without sacrificing essential functionality;

(iii) integrating a substantial and detailed (albeit still informal) feedback flow from diverse usability and expert reviewers, including practical experiences made by several authors of this study while using manual *POSTboxes* in clouds on different operating systems;

(iv) initial work gathering requirements and implementing file system folder structures in Evolvix with the goal to make them more uniform and easier to manage.

Completeness in all areas is not provided by these efforts as indicated by the reserved names:

(i) Important details about *POSTnet* automation and related substructures of the folders associated with *StabilityCodes* remain to be specified.

(ii) POST reserves *all* folder and file names starting with 'ZZ' for future extensions (briefly summarized as 'ZZ*'), both in the PHF and in its '_POST' folder to meet diverse needs in areas known to be important, but not yet addressed by POST so far (including orderly storage of simulation results in file-systems, clearly defining folders for data import or export, and many more). All these ZZ* codes need to cover certain areas of computational and storage related semantics to facilitate efficient POST automation, including aspects related to swapping the active content of different stability levels, Git, Memory, Backup, renaming, metadata and many other data structures and operations.

(iii) There is a need to define all POST relevant version variant algebra sets and operations to implement the systems that automate storing, increasing and switching version variants.

(iv) As long as POST is at RRv1 any attempts to simplify without loss of expressivity are welcome. While *RR* establishes some interest to keep continuity, radical changes are still possible if a need is found to justify them. This liberty is shrinking fast on the path towards *TT* and will be completely gone once *TT* is reached. Therefore we recommend to simplify as much as possible now to reap the benefits later.

(v) Recommendations are needed for what to require to advance to the next level on the *StabilizingZone* for projects eager to do so. While any levels up to *QQ* are essentially to be managed internally by a project, POST demands that *RR* actually is a public 'release' that has been 'reviewed' by someone with a position that is authorized to do so within the project. However, *RR* does not require outside interactions and thus the quality of *RR* releases is expected to vary substantially among projects. Progress from *RR* to *SS* to *TT* requires increasingly stringent external evidence of substantial long-term stability in the face of repeated rounds of real in-depth review. Details remain to be specified, as well as how to document them automatically.

It is beyond the scope of this work to finalize POST or to discuss more semantic details of the *StabilityCodes* we named below. Our definition of POST is one of several examples from Evolvix development that we could have given to illustrate how the BEST Names concept might facilitate the navigation of complex areas and how it can affect the development of a programming language by enabling filled programming language design approach (which was extensively used to develop POST; this would have been impossible to achieve otherwise). POST is a particularly fitting example for the power of BEST Names, as conflicts in naming requirements for different reasonable use cases of POST *StabilityCodes* are likely to make POST-like systems either so terse or so verbose that beginners or power-users respectively, will find them almost unusable (unless an equivalent of the BEST Names concept is used to disentangle both legitimate needs). We certainly would not have developed POST without the possibility to implement a BEST Names infrastructure.

We used POST here as an example for BEST Names. However, this does not diminish the importance of the POST specification published in this particular publication, which is rated at RRv1 by its architect and current guardian. The BEST Names given in Figures P2-P3 thus mark an important milestone of POST design, which now provides a substantial organizational infrastructure that integrates a broad spectrum of key areas required for managing projects small or large. It has reached a substantial level of refinement that has been deemed enough to merit implementation in the core of Evolvix that aims for long-term stability.

The *StabilizingZone* from *MM* to *TT* is of paramount importance for core Evolvix as it is essential for achieving the goal of reliable long-term backwards compatibility by disentangling the faster development of experimental code innovations from the much slower pace that characterizes the development of robust long-term standards (which highly benefit from many rounds of review and is therefore slow). Given this essential role, it is important to collect experiences with this approach to develop backwards compatible code. Accordingly, great care was taken to structure the POST system presented here in a way that allows its formal features like *StabilityCodes* to remain unchanged as POST is expanding to meet new needs. This allows older code to remain stable as the remaining features are added to POST without removing existing functionality. The internal goal in POST development has been to propose a syntax and semantics that stands a chance to be of *TT* quality, even though it is currently 'only' released as *RRv1* to reserve the possibility to change core aspects of the design in case a new essential need is to be met.

Justification of *RRv1*: Many aspects of POST are much more stable than *RR* (e.g. the concept of a *StabilizingZone* itself and the "double caps keyword" idea have not changed in a long time despite much turnover and no lack of attempts to improve, therefore these are possibly close to or at "*TT*"). However, other POST aspects have not been explored and tested enough to warrant an overall label beyond *RR* for the system as a whole (which in POST is limited by the stability of its weakest elements). The precise nature of the hurdles that need to be overcome by any idea to reach "*SS*" or "*TT*" is an open area of development in POST and beyond the scope of this paper even though it can ultimately be reduced to a naming problem.

RR with the aim to reach *TT* implies the strongest possible invitation to anybody anywhere to join the Quality Quest to scrutinize, test, improve and polish any such proposed standards as much and as soon as possible in all conceivable aspects to improve the chances of long-term stability, usability and usefulness. To ensure an appropriately balanced integration of all received feedback that remains as simple as possible in the context of its bigger system, the POST system requires the existence of one

single guardian brain equally dedicated to protect and expand the simplicity, usability, expressivity, accuracy, and security of a *TT* grade standard that can serve users internationally on the long term. While this guardian is likely to critically depend on external input that may come from a supporting committee that helps to represent many voices and perspectives appropriately, such committees usually do not easily arrive at solutions as elegant and effective as capable guardians that are dedicated to finding the best possible solution to serve the community on the long-term.

Feedback submission: Until more efficient ways of integrating feedback are implemented, please contact the architect of POST (L. Loewe) with any general improvements and in particular with comments on the *StabilizingZone*. To reach long-term stability as fast as possible, an important aim of current POST development is to get the *StabilizingZone* itself as fast as possible to *TT* level stability – albeit without allowing weaknesses to remain in the definitions of names from *MM* to *TT*. Such weaknesses could easily become disastrous for code relying on *TT* stability that would then no longer be able to rely on stable names for labeling stability.

Please consult the POST home on the web at

http://evolvix.org/post

for updates or further details on how to contribute to POST development.

7. References for POST

1. Darwin, C. 1859. *On the origin of species by means of natural selection*. J. Murray. London,.
2. Loewe, L. 2007. Evolution@home: observations on participant choice, work unit variation and low-effort global computing. *Softw Pract Exper*. **37**: 1289-1318.
3. Hucka, M., A. Finney, H.M. Sauro*, et al.* 2003. The systems biology markup language (SBML): a medium for representation and exchange of biochemical network models. *Bioinformatics*. **19**: 524-531.
4. Auffray, C. & L. Hood. 2012. Editorial: Systems biology and personalized medicine - the future is now. *Biotechnol J*. **7**: 938-939.
5. Loewe, L. 2016. "Systems in Evolutionary System Biology". In *Encyclopedia of Evolutionary Biology*, Vol. 4. Richard M. Kliman (Chief) & Hiroshi Akashi (Section), Eds.: pp. 297-318, Oxford: Academic Press (Elsevier). (http://evolutionarysystemsbiology.org/pdf/Loewe-2016-evosysbio.pdf).
6. Ehlert, K. & L. Loewe. 2014. Lazy Updating of hubs can enable more realistic models by speeding up stochastic simulations. *J Chem Phys*. **141**: 204109.
7. McKeown, J.C. & J.M. Smith. 2016. *The Hippocrates code : unraveling the ancient mysteries of modern medical terminology*. Hackett Publishishing Company, Inc. Indianapolis, Indiana.
8. Arp, R., B. Smith & A.D. Spear. 2015. *Building ontologies with Basic Formal Ontology*. Massachusetts Institute of Technology. Cambridge, Massachusetts http://ontology.buffalo.edu/smith/.
9. van Renssen, A.S.H.P. 2005. *Gellish: A Generic Extensible Ontological Language - Design and Application of a Universal Data Structure*. Delft University Press. Delft, Netherlands. http://resolver.tudelft.nl/uuid:de26132b-6f03-41b9-b882-c74b7e34a07d.
10. Raymond, E.S. 2003. *The Art of Unix Programming*. Addison-Wesley (Pearson). Boston, MA, USA. http://www.catb.org/~esr/writings/taoup/.
11. Glickstein, B. & K. Hodgson. 2015-11-09. GNU Stow 2.2.2, a symlink farm manager. https://www.gnu.org/software/stow/.

InfoFlows in a Project Organization Stabilizing Tool (POST) Home Folder (PHF)

POST aims to reduce organizational burdens in info-processing projects of any size by associating frequently used types of meaning with standardized Brief and Explicit *'StabilityCode'* Names chosen to be memorable. You choose the *StabilityCodes* your project needs from the general scaffolding, organizing and stabilizing blueprint given by POST; it does not matter whether you are writing a small text or a big complex nested software system that needs long-term backwards compatibility. **Arrows below** show frequent InfoFlows between *StabilityCodes.* See *Brief Dictionary of* POST and *Intro* for more details.

.../MyPOST_Home/ **POST_Home_Folder** PHF *Here is the Project Home with **activated content** and the active '**_POST**' folder with inactivated and/or management content; all names starting with **ZZ** are reserved*

.../.../_POST/ **'_POST' = POST_Cabinet** ***NEVER RENAME OR MOVE*** *this essential anchor of POST infrastructure that stores most Top POST folders (except ZZ*)*

AnyArrival AA *Quickdrop anything to inbox*

BackBurner BB *Send clutter to sea of ideas*

AnyAimsAdmin AAA *Manage projects, goals, priorities, issues, tasks, workflows, ...*

.../.../.../CC **CollectedContent** CC *Collect core ideas, methods, background, data, refs, tools, coding construct explanations, wider work, etc...*

.../.../.../AHA **AnyHelpArea** AHA *Help for users & developers*

DesignDoc DD *Design methods, systems & specifications; document why & why not* Prepare for *StabilityLadder*

.../.../.../GG **GrandGallery** GG export import via ➔ UU archive ➔ HH *Showcase compilation of all ultimate outcomes from LabLog and elsewhere; polished presentation of key results, changes over time as removal is OK; no long-term stability is expected (see MM➔TT).*

.../.../.../HH

UploadUsed UU *Variants sent or received*

LabLog LL *Chronology of all work done*

EnclosedEffort EE_x *Hide work of NamedGroup x*

FeedbackFlow FF *Review & comment streams*

HistoryHeap HH *Declutter main growth area*

InternalItem II *Personal space free info flow*

JammedJob JJ *Try solve this problem soon*

KnownKiller KK *Don't waste your time here*

ActivatedContent that is used for work here (local control) **(=Set of all files or folders not named '_POST' or 'ZZ*').** Inactivated Sets can be activated from VV, GG, MM, ... TT.

More area-specific _POST Keywords with reserved names for top folders and independent subfolder namespaces are being added, including these:
Def inition, **Est** imate, **Obs** ervation, **Sim** ulation, **Opi** nion, **Res** ult, **Lab** Automation, **Log** Automation, **WebS** ubscribed, **WebP** ublished, **Ma** chine, **Me** mory, **Sp** ace, **Fu** nction, **Te** mplate, **Ro** le, **QAS**, **PBC**, **Data**, **Type**, **Code**, **Idea**; see other area-specific controlled vocabulary lists for more.
Combine 'NameStarters' above with any reserved 'KeyNameFragment' below (or use subfolders; define the full intended meaning in Def): **--- Preprocessed (Obs-)**: **Adj** usted, **Chk** IntegrityCheck, **Maj** ority, **Mis** take, **Mod** ified, **Odd** Outlier, **Raw** AsReceived, **Scd** Scaled, **Try** Risk; **--- Estimation (Est-)**: **Gue** ssed, **Pre** dicted, **Evi** dence, **Upd** Updated, **Lkh** Likelihood, **Xpl** Explained, **Fct** Forecast, **Dst** Distance, **Dif** ference, **Difl** Differential, **Cal** culatedAnalytic, **Sim** ulatedPrediction; **--- ModelResult (Sim-)**: **MFr** ModelFrame, **MSt** ModelStructure, **MVa** ModelVariant, **MRe** ModelRepeat, **MIndx**, ModelResultsIndex; **--- ProblemTreeCompilation (PBC, Automat.)**: **Err** ErrorTree, **Wrn** WarningTree, **Rpt** ReportTree, **Ifo** InfoBlockTree, **Why** WhyCausalTree, **Slo** SlowProgressTree **Dbg** DebugTree, **Ast** AbstractSyntaxTree; **--- QualityAssessmentSummary (QAS, Human.)**: **Pbh** ProblemTreeNotes, **Lim** LimitationsTree, **Cal** CalendarTree, **Mat** MaturityTree, **Exi** ExistenceTree; **Extras: --- Relevance**: **Core** KeyResults, **Keep** InArchives, **Late** stFewResults, **Logg** ingAllResults, **Fail** edTriedRisk, **Bugs** AnyReport **Prob** lemsInModel, **Reso** urceWarning, **Temp** oraryIntermediateResults; **--- SequenceChain**: **Frst** First, **Prev** ious, **Curr** ent, **Next**, **Last**, **CIndx** ChainElementIndex; **--- Compared**: **U** Usual, **Q** Other,, **Wt** Wildtype, **Mt** Mutant, **Wtl** Wildtypelike, **Mtl** Mutantlike,, **Ni** Noise, **Si** Signal,, **Tst** TestExperiment, **Ctr** ControlExperiment. More TBD

MockupModel MM *Quick experiment, pilot study*

NewNonfunctional NN *Foundations, not working yet*

OperatesOften OO *Some parts are now useful*

PreProbing PP *Soon ready for questioning*

QualityQuest QQ *Questioning, under review*

ReviewedRelease RR *Production quality product*

.../.../.../SS **StableSource** SS *Successful popular product*

TrustedTested TT *Long-term stable & reliable*

StabilizingZone

ZenZone

Version Variant VV ***Archive of installations*** *can run the same tasks with code from different versions*

YesYet YY *Automated testing of each given assumed Yes, Yet testing is always limited*

WorkingWheels WW *External tools used locally and assumed to be reliable*

[*NoCode*], **XenoXero** XX *Strange or copied Codes (or missing or implicit, ...)*

.../.../ZZ **ZenZoom** ZZ *Content here is 'inside' but not managed by this PHF*

.../.../ZZA **ZenZoomAnchor** ZZA *Anchors are stable link targets pointing outside*

.../.../ZZ* **ZenZoomAny** ZZ* *All names starting with "ZZ": are reserved*

ZenZoomPOST ZZP* *ZZP* Folders are like ZZ* but stored stable inside of **_POST***

ZenZoomReplace ZZR* *change with content of PHF*

Translation between human languages: the PHF is near-empty to help translate folder names by creating link farms, where stable content remains in PHF/_POST or /ZZ* (mostly for use by code) and translated links for users are added to the PHF (*only* for users, *not for code!*)

Project Organization Stabilizing Tool (POST) system, RRv1r0p0, 2016-09-12 by L. Loewe

Brief Dictionary of the Project Organization Stabilizing Tool (POST) system

Synonymous **Brief**, **Explicit**, and **Summarizing** Names of POST *StabilityCodes* define keywords in Evolvix for fast data annotation

Use Case	Type:POST	Brief	Explicit Name	Summarizing Name of *POSTcode* or *StabilityCode*	Mainly used by ...
Getting organized	AnyZone	**AA**	AnyArrival_*WorkspaceID*	AnyArrival_UsedFor_Inbox_QuickDropping_Items_to_ProcessLater	Developer, Lead, Anybody
	Workflow Zone	**AAA**	AnyAimsAdmin _*WorkspaceID* ≡ Name of Workspace (if needed)	AnyAimsAdmin_UsedFor_Managing_StrategyAims_ProjectPlans__ __Workflows_RoleDefs_Tracking_TasksDeadlinesFunds__ __RecruitingTraining_Delegation_CallBacks_Prioritizing_KeyGoals	Project **Leaders** (Lead) Expert **Developers** (Dev) Any Contributor
		BB	BackBurner_*WorkspaceID*	BackBurner_UsedFor_Storing_SeasOfGoodIdeas_to_RevisitLater	Dev, Lead
Ordering collected info	Info Zone	**AHA**	AnyHelpArea	AnyHelpArea_UsedFor_OnlineHelp_UserManuals_InfoMessages__ __TeachingOutreachMaterial_Translatable_by_CountryCode	New outside **Users** (User) Dev, Lead
		CC	CollectedContent _*WorkspaceID*	CollectedContent_UsedFor_Collecting_External_TopicIdeas_Data__ __Methods_Evidence_Refs_Reviews_Clarity_CodeExplanations	System **Designers** (Design) Lead, Dev
		DD	DesignDoc _*WorkspaceID*	DesignDoc_UsedFor_Describing_ImplementableTechSpecification__ __and_ProsCons_of_DesignDecisions_Methods_AlternativeIdeas	*Produced* by Design, Lead *Used* by Dev,...
Restricting access for reviewing	Review Zone	**EE**	EnclosedEffort _*WorkspaceID*	EnclosedEffort_UsedFor_Collaborating_InClosedGroups__ __with_AccessRestricted_as_AgreedBy_TheNamedGroup	Anybody from named group
		FF	FeedbackFlow _*WorkspaceID*	FeedbackFlow_UsedFor_Logging_AnyInputCommentCritiqueBug__ __Ideas_from_ReviewerExpertUserFriendFoeFans_Worldwide	*Produced* by User, Dev, Lead *Analyzed* by Design, Dev, ...
Keeping instant startups simple	Instant StartUp Zone	**GG**	GrandGallery _*WorkspaceID*	GrandGallery_UsedFor_Presenting_BestExhibits_LatestResults__ __FinalKeyDeliverables_without_NeedForLongTermStability	Anybody as agreed
		HH	HistoryHeap _*WorkspaceID*	HistoryHeap_UsedFor_OldVersionBackup_ProbablyDeletableFiles__ __HoardedDataWithLimitedOrder_KeptFor_PotentialUsefulness	Anybody as agreed
Managing problems Try things	Scratch Zone	**II**	InternalItem _*WorkspaceID*	InternalItem_UsedFor_ScratchPad_AnyChaoticContent_MeantFor__ __YourCreativeMuse_IsReadableUnlessYou_RestrictAccess	Personal (my space has no order anybody can understand)
		JJ	JammedJob _*WorkspaceID*	JammedJob_UsedFor_Documenting_BugReports_DecisionNeeds__ __and_ProgressOn_SolvableProblems _that_NeedSolutionsSoon	Dev, Lead, User, Design, ...
		KK	KnownKiller _*WorkspaceID*	KnownKiller_UsedFor_Documenting_DeprecatedCode_FaultyData__ __FailedIdeas_BadSolutions_etc_ThatKillTimeWhenRevisited	Dev, Design, Lead, Review, ...
		LL	LabLog _*WorkspaceID*	LabLog_UsedFor_Logging_AnyLabLabor_Dated_YYYY_MM_DD__ __ForHistoricSequenceOf_AllOutcomesOf_AllTests_IncludingFails	Dev, Design, Lead, Review, ...
Maximizing stability for projects of **importance**, by climbing inside of the ***Stabilizing Zone*** and becoming ideally: **Stable Forever** Only fully long-term backwards compatible systems shall get to TTv1	**More area specific keywords with reserved top folders** and independent subfolder namespaces are being added, including the following: **Def** inition, **Est** imate, **Obs** ervation, **Sim** ulation, **Opi** nion, **Res** ult, **Lab** Automation, **Log** Automation, **WebS** ubscribed, **WebP** ublished, **Ma** chine, **Me** mory, **Sp** ace, **Fu** nction, **Te** mplate, **Ro** le, **Data**, **Type**, **Code**, **Idea**; see other area-specific controlled vocabulary lists.				
	Stabilizing Zone	**MM**	MockupModel	MockupModel_UsedFor_RapidPrototyping_InformalLearning__ __ExpertimentsToBeThrownAway_StabilizingDesignNotCode	Dev, Design, Lead, ...
		NN	NewNonfunctional	NewNonfunctional_UsedFor_NotYetFunctioning_DeepFoundations__ __ForLargerStableDesigns_ThatDoNotYetWorkForUsers	Dev, Design, Lead, ...
		OO	OperatesOften	OperatesOften_UsedFor_Systems_PartiallyWorkingForEndUsers__ __while_StillMissing_ImportantFeatures_ToBeImplemented	Dev, Design, Lead, ...
		PP	PreProbing	PreProbing_UsedFor_Preparing_PeerReviewAndPublicProbing__ __by_PolishingExistingFeatures_UntilSubmissionFor_Questioning	Dev, Design, Lead, ...
		QQ	QualityQuest	QualityQuest_UsedFor_Questioning_AxiomsDataScienceAccuracy__ __RigorClarityUsability_InMany_ExpertBeginnerReviewRounds	Outside expert and usability **Reviewers**, Dev, Lead
		RR	ReviewedRelease	ReviewedRelease_UsedFor_NewReleasesRecommended_by__ __QualityQuestEditors_after_AnsweringAllReviewerQuestions	User, Dev, ...
		SS	StableSource	StableSource_UsedFor_StunningSoftware_RunningInProduction__ __with_LongTermSuccess_and_VeryRareRevisionRequests	User, Dev (rarely), ...
		TT	TrustedTested	TrustedTested_UsedFor_Marking_VeryLongTermStableDesigns_in__ __WellUnderstoodDomains_AllowingBackwardsCompatibleGrowth	User only; development is over if review worked well, ...
Keeping records repro-ducible	Archival Zone	**UU**	UploadUsed _*WorkspaceID*	UploadUsed_UsedFor_Archiving_ExternalCommunicationLog_of__ __FullUploadsOf_InputTo_or_OutputFrom_GG_etc	Anybody as agreed (Log key imports & exports)
		VV	VersionVariant	VersionVariant_UsedFor_InstallingAllLocalProducts_FullyArchived__ __Checksummed_ReproduciblyWorking_AllFiles_Ready2Publish	All installations are easy to activate and use in situ
		WW	WorkingWheels _*WorkspaceID*	WorkingWheels_UsedFor_AllExternal_CodeProgramToolVersions__ __TrustedEnough_for_Integration_IntoThis_POSTHomeFolder	Dev, Design, Lead, ...
		YY	YesYet _*WorkspaceID*	YesYet_UsedFor_AutomatedTesting_of_EachUnitAssumingYes__ __and_EachFeatureOrWeaknessThatNeeds_YetToBeAutoTested	Dev, Design, Lead, ... most values are auto-produced tests
Missing, conflicting, automated *StabilityCodes*	Flexible Zone	**XX**	XenoXero _*WorkspaceID*	XenoXero_UsedFor_StrangeAndCopiedCodes_DefinedElsewhere_ __AutoGenerated_Temporary_Unspecified_Unclear_or_BadCodes	Automatically, Dev, ...
Managing links, limits, controls, and more.	ZenZone	**ZZ**	ZenZoom	ZenZoom_UsedFor_Nesting_POST_ContentHierarchies_Requiring__ __StorageInsideOf_ButManagementIndependentFrom_ThisPOST	Automatically, ZZ contains independent POST Homes
		ZZA	ZenZoomAnchor	ZenZoomAnchor_UsedFor_Anchoring_StableLinkChainTargets__ __UsedInThisPOST_ToLinkToEasilyAdjustable_OutsideContent	Automatically, Dev, ...
		ZZ*	Names starting with ZZ... are reserved	ZenZoom... All names starting with 'ZZ' are reserved for future POST automation use and must not be used in a POST Home Folder.	Automatically, Dev, ...
		_POST	POST_Cabinet	POST_Cabinet_UsedFor_StableAutomating_and_Decluttering__ __of_POSTHomeFolder_Stowing_AllNonActivatedPOSTContent	Automatically, Dev, ...

Project Organization Stabilizing Tool (POST) system, RRv1r0p0, 2016-09-12 by L. Loewe